



VR-Link

Reference Manual



VR-Link

Reference Manual



*MÄK Technologies
185 Alewife Brook Parkway
Cambridge, MA 02138 USA*

Copyright 1999, MÄK Technologies

All rights Reserved. Printed in the United States.

First Edition: April, 1999

Under copyright laws, no part of this document may be copied or reproduced in any form without prior written consent of MÄK Technologies, Inc.

VR-Link[®] is a registered trademark of MÄK Technologies, Inc.

All other trademarks are owned by their respective companies.

MÄK Technologies
185 Alewife Brook Parkway
Cambridge, MA 02138 USA

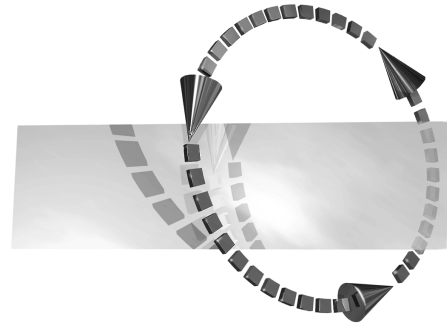
Voice: 617-876-8085

Fax: 617-876-9208

info@mak.com

www.mak.com

Revision VRL-3.4-2-990510



Contents

Preface

How this Guide is Organized	xv
VR-Link Documentation.....	xvi
Other MÄK Products	xvii
How to Contact Us	xviii
Document Conventions	xix

Chapter 1 Guide to VR-Link Header Files

1.1. Convenience Header Files	1-2
1.1.1. Header Files for Protocol Independence	1-2
1.2. Class-to-Header Cross Reference	1-4
1.3. Header-to-Class Cross Reference	1-23
1.4. Other Header Files	1-41

Chapter 2 PDU Class Reference

DtAcknowledgePdu	2-2
DtActionRequestPdu	2-4
DtActionResponsePdu	2-6
DtCollisionPdu	2-8
DtCommentPdu	2-10
DtCreateEntityPdu	2-12
DtDataPdu	2-13
DtDataQueryPdu	2-14
DtDesignatorPdu	2-16
DtDetonationPdu	2-18
DtEmissionPdu	2-21
DtEntityStatePdu	2-26
DtEventReportPdu	2-32
DtFirePdu	2-33

DtLaserPdu	2-35
DtMessagePdu	2-36
DtPduWithData	2-41
DtReceiverPdu	2-48
DtRemoveEntityPdu	2-49
DtRepairCompletePdu	2-50
DtRepairResponsePdu	2-51
DtResupplyCancelPdu	2-52
DtResupplyOfferPdu	2-53
DtResupplyReceivedPdu	2-55
DtSetDataPdu	2-57
DtSignalPdu	2-58
DtSimanPdu	2-60
DtStartResumePdu	2-62
DtStopFreezePdu	2-64
DtTransmitterPdu	2-66
DtViewControlPdu	2-70

Chapter 3 Header File Reference

absNotifyObs.h	3-2
ackPdu.h	3-3
ackRPdu.h	3-5
actionPdu.h	3-7
actReqRPdu.h	3-10
actRespRPdu.h	3-12
aggAggList.h	3-14
aggEntDec.h	3-16
aggEntEnc.h	3-18
aggEntList.h	3-20
aggStatePdu.h	3-22
aggregateSR.h	3-29
antPatRec.h	3-34
approximator.h	3-36
apSpec.h	3-38
arealObjStPdu.h	3-39
artPart.h	3-43
artPartList.h	3-47
artPartThresh.h	3-51
attPartList.h	3-54
baseEmbedDec.h	3-57
baseEmbedEnc.h	3-59
baseEmittrDec.h	3-61
baseEmittrEnc.h	3-63
baseEntDec.h	3-66
baseEntEnc.h	3-69

baseEntitySR.h	3-72
baseExConn.h	3-77
baseHashKey.h	3-79
baseMsg.h	3-81
baseReflAgg.h	3-83
baseReflEnt.h	3-85
beamAntRec.h	3-86
cartView.h	3-89
ClientSocket.h	3-91
collidePdu.h	3-94
collisElasPdu.h	3-96
commentPdu.h	3-99
commentRPdu.h	3-101
createEntRPdu.h	3-103
createPdu.h	3-105
dAggDec.h	3-107
dAggEnc.h	3-108
dAggPub.h	3-110
dataPdu.h	3-111
dataQueryRPdu.h	3-112
dataRPdu.h	3-114
datumIdType.h	3-116
datumMap.h	3-117
datumRecSet.h	3-119
dcm.h	3-126
deadReckoner.h	3-127
decFactory.h	3-131
dEntityDec.h	3-134
dEntityEnc.h	3-136
dEntityPub.h	3-138
designatePdu.h	3-139
designatorDec.h	3-141
designatorEnc.h	3-143
designatorSR.h	3-145
detonatePdu.h	3-147
dExerciseConn.h	3-150
dInteraction.h	3-156
directRotRate.h	3-157
dirRtiAmb.h	3-158
disEnums.h	3-170
disPduFactory.h	3-215
dRefEmSysList.h	3-216
dReflAggList.h	3-218
dReflectedAgg.h	3-221
dReflectedEnt.h	3-224
dReflectedObj.h	3-227

dReflEmitrSys.h	3-229
dReflEntList.h	3-231
dRefObjList.h	3-234
dscvObjCbInfo.h	3-237
dStateMsg.h	3-239
elecEmitPdu.h	3-241
embedSystemSR.h	3-245
emissionPdu.h	3-247
emitterSysDec.h	3-251
emitterSysEnc.h	3-253
emitterSysSR.h	3-255
emitttrBeamSR.h	3-257
emptyFomMap.h	3-261
encFactory.h	3-263
entitySR.h	3-266
EntityTypes.h	3-271
enum2string.h	3-292
envDec.h	3-293
envEnc.h	3-295
enviroTypeRec.h	3-297
envProcPdu.h	3-298
esPdu.h	3-301
esUpdatePdu.h	3-304
Euler.h	3-308
eventId.h	3-309
eventReptPdu.h	3-312
eventReptRPdu.h	3-313
fedReader.h	3-315
fedTarget.h	3-317
filename.h	3-319
firePdu.h	3-321
fom.h	3-323
fomMapper.h	3-326
gedBasFixWing.h	3-332
gedBasRotWing.h	3-335
gedBasSoldier.h	3-338
gedBasVehicle.h	3-341
gedEnhFixWing.h	3-344
gedEnhRotWing.h	3-346
gedEnhSoldier.h	3-348
gedEnhVehicle.h	3-350
gedLogVehicle.h	3-352
genericDec.h	3-355
genericEnc.h	3-356
genericPub.h	3-357
genericSR.h	3-358

geodCoord.h	3-360
globalObjDes.h	3-363
globIdHLst.h	3-364
gridAxiRecFix.h	3-366
gridAxiRecVar.h	3-369
gridDataPdu.h	3-371
gridDataRec.h	3-376
gridDataRecT0.h	3-378
gridDataRecT1.h	3-380
gridDataRecT2.h	3-382
hAckDec.h	3-384
hAckEnc.h	3-386
hAckInter.h	3-387
hActReqDec.h	3-389
hActReqEnc.h	3-391
hActReqInter.h	3-392
hActReqObjDec.h	3-394
hActReqObjEnc.h	3-396
hActReqObjInt.h	3-397
hActRespDec.h	3-399
hActRespEnc.h	3-401
hActRespInter.h	3-403
hActRspObjDec.h	3-405
hActRspObjEnc.h	3-407
hActRspObjInt.h	3-409
hAggPub.h	3-411
hAppRadSigDec.h	3-413
hAppRadSigEnc.h	3-415
hashItem.h	3-417
hashlist.h	3-418
hAttChgReqDec.h	3-422
hAttChgReqEnc.h	3-423
hAttChgReqInt.h	3-424
hAttChgResDec.h	3-427
hAttChgResEnc.h	3-429
hAttChgResInt.h	3-431
hCollideDec.h	3-434
hCollideEnc.h	3-436
hCollideInter.h	3-438
hCommentDec.h	3-440
hCommentEnc.h	3-441
hCommentInter.h	3-442
hCreatEntDec.h	3-444
hCreatEntEnc.h	3-446
hCreatEntInt.h	3-447
hCrtObjReqDec.h	3-449

hCrtObjReqEnc.h	3-450
hCrtObjReqInt.h	3-451
hCrtObjResDec.h	3-454
hCrtObjResEnc.h	3-456
hCrtObjResInt.h	3-458
hDataDec.h	3-460
hDataEnc.h	3-462
hDataInter.h	3-463
hDataQDec.h	3-465
hDataQEnc.h	3-467
hDataQInter.h	3-468
hDbiRadSigDec.h	3-471
hDbiRadSigEnc.h	3-473
hDesignatrPub.h	3-475
hDetDec.h	3-477
hDetEnc.h	3-479
hDetInter.h	3-481
hEmitrBeamPub.h	3-485
hEmittrSysPub.h	3-487
hEncARSigDec.h	3-489
hEncARSigEnc.h	3-491
hEntityPub.h	3-493
hEventRepDec.h	3-494
hEventRepEnc.h	3-496
hEventRepInt.h	3-497
hExerciseConn.h	3-499
hFireDec.h	3-508
hFireEnc.h	3-510
hFireInter.h	3-512
hInteraction.h	3-515
hInterFactory.h	3-519
hInterWData.h	3-521
hIntWEncDec.h	3-527
hlaObj.h	3-530
hlaObjFact.h	3-537
hlaObjWithSR.h	3-539
hLcDec.h	3-542
hLcEnc.h	3-543
hLcInter.h	3-544
hObjPub.h	3-546
hostStructs.h	3-550
hRadioRcvrPub.h	3-557
hRadioXmitPub.h	3-559
hRawBinRSDec.h	3-561
hRawBinRSEnc.h	3-563
hRefBeamList.h	3-565

hRefDesigList.h	3-567
hRefEmSysList.h	3-569
hReflAggList.h	3-573
hReflBeam.h	3-576
hReflDesigntr.h	3-578
hReflectedAgg.h	3-580
hReflectedEnt.h	3-582
hReflectedObj.h	3-584
hRefObjList.h	3-588
hRefEmitrSys.h	3-593
hRefEntList.h	3-595
hRefRadRecvr.h	3-598
hRefRadXmitr.h	3-600
hRefRadRvList.h	3-602
hRefRadXmList.h	3-604
hRemObjReqDec.h	3-606
hRemObjReqEnc.h	3-608
hRemObjReqInt.h	3-609
hRemObjResDec.h	3-611
hRemObjResEnc.h	3-613
hRemObjResInt.h	3-615
hRemovEntDec.h	3-617
hRemovEntEnc.h	3-619
hRemovEntInt.h	3-620
hRepCompDec.h	3-622
hRepCompEnc.h	3-624
hRepCompInter.h	3-625
hRepRespDec.h	3-627
hRepRespEnc.h	3-629
hRepRespInter.h	3-630
hResCancDec.h	3-632
hResCancEnc.h	3-634
hResCancInter.h	3-635
hResOffDec.h	3-637
hResOffEnc.h	3-639
hResOffInter.h	3-640
hResRecDec.h	3-642
hResRecEnc.h	3-644
hResRecInter.h	3-645
hServReqDec.h	3-647
hServReqEnc.h	3-649
hServReqInter.h	3-650
hSetDataDec.h	3-653
hSetDataEnc.h	3-655
hSetDataInter.h	3-656
hSignalInter.h	3-658

hStartDec.h	3-661
hStartEnc.h	3-663
hStartInter.h	3-664
hStateDecoder.h	3-666
hStateEncoder.h	3-670
hStateMsg.h	3-675
hStopDec.h	3-678
hStopEnc.h	3-680
hStopInter.h	3-681
hUnknownInter.h	3-684
hVcDec.h	3-686
hVcEnc.h	3-687
hVcInter.h	3-688
iffAtcNavBeam.h	3-690
iffAtcNavLayr.h	3-692
iffAtcNavOper.h	3-694
iffAtcNavOpr2.h	3-697
iffAtcNavParm.h	3-699
iffAtcNavPdu.h	3-701
iffAtcNavSyID.h	3-705
intClassDesc.h	3-707
intDecFact.h	3-710
intEncFact.h	3-713
intercmCtlPdu.h	3-715
intercmEntDst.h	3-719
intercmGrpAsg.h	3-721
intercmGrpDst.h	3-723
intercmParRec.h	3-725
intercmSigPdu.h	3-727
interDecoder.h	3-730
interEncoder.h	3-733
interMgr.h	3-736
intHashKey.h	3-738
isGroupOfPdu.h	3-739
isPartOfPdu.h	3-744
jammerBeamDec.h	3-747
jammerBeamEnc.h	3-749
laserPdu.h	3-751
lcEnd.h	3-753
lcEofAction.h	3-754
lcHeartbeat.h	3-755
lcPause.h	3-756
lcPdu.h	3-757
lcPlay.h	3-760
lcPlayAction.h	3-761
lcPlayFile.h	3-762

lcQuit.h	3-763
lcRecAction.h	3-764
lcRecFile.h	3-765
lcRecord.h	3-766
lcSetTime.h	3-767
lcSkipTime.h	3-768
lcStart.h	3-769
lcStop.h	3-770
lcTimeScale.h	3-771
lcTimeout.h	3-772
leAppearPdu.h	3-773
leArtPartsPdu.h	3-778
leDeadReck.h	3-781
leDetonPdu.h	3-783
leEntityID.h	3-788
leEntityOri.h	3-790
leEntityVel.h	3-792
leFirePdu.h	3-794
leLocEntity.h	3-799
leLocWorld.h	3-801
leOriError.h	3-803
lePosError.h	3-805
leTspiPdu.h	3-807
LibMatrix.h	3-811
linObjStPdu.h	3-817
linSegParam.h	3-820
locObjMgr.h	3-823
MachineTypes.h	3-825
mathTypes.h	3-827
messagePdu.h	3-828
milEntDec.h	3-831
milEntEnc.h	3-833
milPlatDec.h	3-835
milPlatEnc.h	3-837
mineDataPdu.h	3-839
minePerimeter.h	3-847
mineQueryPdu.h	3-849
mineRsNackPdu.h	3-853
mineStatePdu.h	3-856
munEntDec.h	3-860
munEntEnc.h	3-862
netBoolean.h	3-864
NetSocket.h	3-866
NetStructs.h	3-871
netTimeStamp.h	3-880
NetTypes.h	3-882

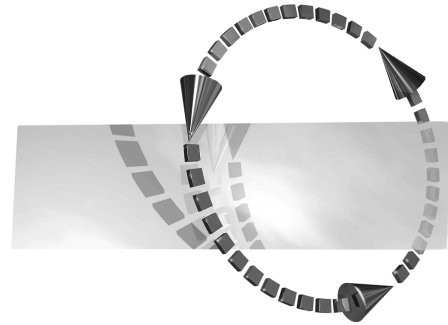
nullFedAmb.h	3-885
objClassDesc.h	3-893
objDesList.h	3-897
objectType.h	3-899
objIdHashKey.h	3-901
objIdHashLst.h	3-902
objRefCbInfo.h	3-904
pairSock.h	3-906
pdu.h	3-907
pduCbInfo.h	3-910
pduCbMgr.h	3-911
pduDatRecSet.h	3-913
pduFactory.h	3-915
pduWithData.h	3-917
physEntDec.h	3-923
physEntEnc.h	3-926
pointObjStPdu.h	3-929
printUtil.h	3-933
ProcControl.h	3-934
queryPdu.h	3-935
radarBeamDec.h	3-937
radarBeamEnc.h	3-939
radioRecvrDec.h	3-941
radioRecvrEnc.h	3-943
radioRecvrSR.h	3-945
radioXmitDec.h	3-947
radioXmitEnc.h	3-949
radioXmitSR.h	3-952
receivePdu.h	3-953
recIntrCbInfo.h	3-955
recordPdu.h	3-957
recordSet.h	3-960
recordSetPdu.h	3-964
recQueryPdu.h	3-967
refAttrCbInfo.h	3-970
refAttrCbList.h	3-972
refGenericObj.h	3-974
refGenList.h	3-976
refObjCbInfo.h	3-978
refObjMgr.h	3-980
regCtlCbInfo.h	3-983
regCtlCbList.h	3-985
remObjCbInfo.h	3-987
removeEntRPdu.h	3-989
removePdu.h	3-991
repairPdu.h	3-993

resupplyPdu.h	3-995
rprDecFact.h	3-999
rprEncFact.h	3-1000
rprFomMap.h	3-1001
rprIDecFact.h	3-1004
rprIEncFact.h	3-1005
rprInterFact.h	3-1006
rprObjFact.h	3-1007
script.h	3-1008
servReqPdu.h	3-1009
setDataPdu.h	3-1011
setDataRPdu.h	3-1012
setRecordPdu.h	3-1014
signalPdu.h	3-1017
simanPdu.h	3-1019
simRprFomMap.h	3-1020
smoother.h	3-1021
smoothOriWDR.h	3-1024
smoothPosWDR.h	3-1025
smoothUtil.h	3-1026
smUtil.h	3-1030
Socket.h	3-1031
soldierDec.h	3-1035
soldierEnc.h	3-1037
startPdu.h	3-1039
startResRPdu.h	3-1041
stateRep.h	3-1043
stIntCbInfo.h	3-1045
stIntCbList.h	3-1047
stopFrzRPdu.h	3-1049
stopPdu.h	3-1051
str.h	3-1053
strHashKey.h	3-1056
strtUpdCbInfo.h	3-1057
strtUpdCbList.h	3-1059
supEmisEsNozz.h	3-1061
supEmisEsPdu.h	3-1063
supEmisEsProp.h	3-1066
sysHashKey.h	3-1068
taitBryan.h	3-1070
tcpForwarder.h	3-1071
tcpSocket.h	3-1072
tcpSocketMgr.h	3-1074
thresolder.h	3-1075
tkJmDesLst.h	3-1077
topoCoord.h	3-1079

Contents

topoView.h	3-1080
trackJamDes.h	3-1081
transCtlPdu.h	3-1083
transmitPdu.h	3-1086
trkJamBeamDec.h	3-1089
trkJamBeamEnc.h	3-1091
tStampType.h	3-1093
undwaAcBeam.h	3-1094
undwaAcEmtSys.h	3-1096
undwaAcPasAct.h	3-1098
undwaAcPdu.h	3-1100
undwaAcShaft.h	3-1104
unknownHObj.h	3-1106
unknownPdu.h	3-1108
updCtlCbInfo.h	3-1109
updCtlCbList.h	3-1111
utmCoord.h	3-1113
utmView.h	3-1117
varDatumRec.h	3-1119
vcPdu.h	3-1122
vlList.h	3-1131
vlprint.h	3-1138
vlTime.h	3-1140
vlutil.h	3-1141
vlVector.h	3-1145
vrlFedAmb.h	3-1147
vrlFedTime.h	3-1153
vrlRtiAmb.h	3-1157

Index



Preface

This manual is a reference for VR-Link programmers. It has information about classes, PDUs, and header files. The bulk of the manual is a listing of the VR-Link header files. This manual is available only in Adobe Portable Document Format (PDF). All entries in the table of contents and index are hyperlinks to the section where the entries are located. Cross references are also live links. PDF files allow full text searching and you can print any portion of the manual.

To view and print PDF documents, you need a copy of the free Acrobat Reader version 3.0 or later. To obtain a copy, visit www.adobe.com.

How this Guide is Organized

The chapters are organized as follows:

Chapter 1, [Guide to VR-Link Header Files](#), lists header files and the classes they contain.

Chapter 2, [PDU Class Reference](#), provides descriptive notes about DIS 2.0.3, 2.0.4, (IEEE1278.1) and 2.1.4 (IEEE 1278.1a) PDUs. This information appeared in many pages in versions of VR-Link prior to version 3.0.

Chapter 3, [Header File Reference](#), contains edited versions of all VR-Link header files.

VR-Link Documentation

The VR-Link documentation set is as follows:

- ♦ *MÄK VR-Link Developer's Guide*. The Developer's Guide explains how to use VR-Link. It includes descriptions of the protocol-independent and protocol-specific APIs, sample code, and descriptions of VR-Link utilities and support applications. This manual is available in printed format and as an online manual in Adobe Portable Document Format (PDF). The PDF file is in the *./doc* directory.
- ♦ *MÄK VR-Link Reference Manual*.
- ♦ Class documentation. Classes are documented in a set of protocol-specific PDF files. HLA classes are documented in *vrl3-xHLAclasses.pdf* and DIS classes are documented in *vrl3-xDISclasses.pdf*. Class hierarchy diagrams are in the files *vrl3-xHLA-classdiagram.pdf* and *vrl3-xDISclassdiagram.pdf*. These files are in the *./doc* directory.
- ♦ Release documentation. Release documentation consists of release notes provided as a text file named *RELNOTES* and in printed format.

Other MÄK Products

VR-Link is a member of the MÄK Technologies line of HLA/DIS software products designed to streamline the process of developing and using networked simulated environments.

In addition to VR-Link, the MÄK Technologies product line includes:

- ♦ The MÄK RTI. An RTI (Run-Time Infrastructure) is required to run applications using the High Level Architecture (HLA). The MÄK RTI is the first commercial run-time infrastructure for HLA simulations. It is optimized for real-time simulations.
- ♦ The MÄK Stealth. The Stealth displays a realistic, three-dimensional view of your virtual world. You can view this world from the inside of a simulated moving vehicle, or place the eyepoint at another moving or stationary location. The Stealth lets you switch rapidly among several predefined viewpoints while the simulation is underway. The Stealth runs on PCs and the SGI computers and supports multi-channel views.
- ♦ The MÄK Logger. The Logger, also called the Data Logger, can record HLA and DIS exercises and play them back for after-action review. You can play a recorded file at speeds above or below normal, or in reverse, and locate areas of interest quickly. The Logger lets you choose between a graphical interface, or a shell-style, command-line interface. You can also control the Logger programmatically with logger control PDUs or state update messages.

The Logger comes with a set of utility programs that let you inspect, combine and edit Logger recordings.

- ♦ The MÄK Plan View Display. The Plan View Display (PVD) provides a two-dimensional, “big picture” of a virtual environment. It shows simulated entities, such as vehicles, soldiers, and tanks moving over a 2D-map display.

Contact MÄK Technologies for more information about these products and their supported platforms.

How to Contact Us

For VR-Link technical support, information about upgrades, and information about other MÄK products, you can reach us in the following ways:

For sales and upgrade information by e-mail: info@mak.com

For technical support by e-mail: support@mak.com

For license key files and information: www.mak.com/support/keyintro.htm

The MÄK web site: www.mak.com

When requesting support, please tell us the product you are using, the version, and the platform on which you are running.

Send postal correspondence to: MÄK Technologies
185 Alewife Brook Parkway
Cambridge, MA, USA 02138

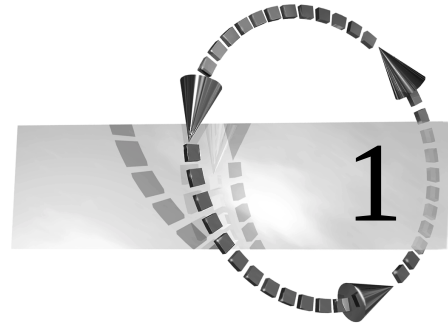
Call or fax us at:	Voice:	617-876-8085 (extension 3 for support)
	Fax:	617-876-9208

Document Conventions

This document uses the following typographic conventions:

Monospaced Type	Indicates commands or values you enter.
<i>Monospaced Italic</i>	Indicates variables that you replace with appropriate values.
[]	Indicates optional arguments.
	Separates options in a command where only one option may be chosen at a time.
/	Indicates a directory. Since MÄK products run on both UNIX and PC platforms, we use the / (slash) for generic discussions of pathnames. If you are running on a PC, substitute a \ (backslash) when you type pathnames.
Bold type	Indicates a new term.
<i>Italic</i>	Indicates a file name or path, or a class name.
sansSerif	Indicates a parameter or argument.
>	Indicates a one-step procedure.
Menu Option	Indicates a menu choice. For example, an instruction to select the Save option from the File menu appears as: Choose File Save.
	Indicates supplemental or clarifying information.
	Indicates additional information that you must observe to ensure the success of a procedure or other task.
	Indicates a warning about potential damage to the program or computer.

Directory names are preceded with dot and slash characters that show their position with respect to the VR-Link home directory. For example, the directory *vrlinkx.x/binRPR* appears in the text as *./binRPR*.



Guide to VR-Link Header Files

This chapter covers the following topics:

Convenience Header Files	1-2
Header Files for Protocol Independence	1-2
Class-to-Header Cross Reference	1-4
Header-to-Class Cross Reference	1-23
Other Header Files	1-41

1.1. Convenience Header Files

Some VR-Link header files exist simply for convenience or to provide protocol independence, and do not require further description in this manual.

The following four files each include the other header files for their respective libraries:

- ♦ *matrixInc.h*
- ♦ *vlInc.h*
- ♦ *mtlInc.h*
- ♦ *vlutilInc.h*.

For convenience, you can include just these general header files in your code. However, you might speed up compilation by including only the header files you need.

The following files group other related header files together for convenience:

- ♦ *mathTypes.h* groups math-related headers, defines math macros
- ♦ *pdu.h* groups all PDU headers.

1.1.1. Header Files for Protocol Independence

Table 1-1 lists header files that are used for protocol-independence.

Table 1-1: Header files for protocol independence

This file:	Provides protocol independence for:
<i>ackInter.h</i>	Acknowledge interactions
<i>actReqInter.h</i>	Action request interactions
<i>actRespInter.h</i>	Action response interactions
<i>aggPub.h</i>	Local aggregates
<i>collideInter.h</i>	Collision interactions
<i>commentInter.h</i>	Comment interactions
<i>createEntInt.h</i>	Entity interactions
<i>dataInter.h</i>	Data interactions
<i>dataQInter.h</i>	Data query interactions
<i>detonateInter.h</i>	Detonation interactions
<i>entityPub.h</i>	Local entities
<i>eventRepInter.h</i>	Event report interactions
<i>exerciseConn.h</i>	Exercise connections
<i>fireInter.h</i>	Fire interactions

Table 1-1: Header files for protocol independence

This file:	Provides protocol independence for:
interaction.h	Base interactions
lclnter.h	Logger control interactions
refEmitLst.h	Reflected emitter lists
reflAggLst.h	Reflected aggregate lists
reflectedAgg.h	Reflected aggregates
reflectedEnt.h	Reflected entities
reflEmittr.h	Reflected emitters
reflEntLst.h	Reflected entity lists
remEntInter.h	Remove entity interactions
reflObjLst.h	Reflected object lists
reflectedObj.h	Reflected objects
repComplInter.h	Repair complete interactions
repRespInter.h	Repair response interactions
resupplyInter.h	Resupply interactions
servReqInter.h	Service request interactions
setDataInter.h	Set data interactions
signalInter.h	Signal interactions
startInter.h	Start and resume interactions
stateMsg.h	State messages
stopInter.h	Stop and freeze interactions
vcInter.h	View control interactions

1.2. Class-to-Header Cross Reference

Table 1-2 contains an alphabetical list of VR-Link classes along with the header file that contains each one. The names of the header files are links to the header file code.

Table 1-2: Classes and their Corresponding Header Files

Class	Header File
DtAbsNotifyObserver	absNotifyObs.h
DtAbsoluteVCPdu	vcPdu.h
DtAcknowledgeDecoder	hAckDec.h
DtAcknowledgeEncoder	hAckEnc.h
DtAcknowledgeInteraction	hAckInter.h
DtAcknowledgePdu	ackPdu.h
DtAcknowledgeRPdu	ackRPdu.h
DtAckPredOpt	smUtil.h
DtActionRequestDecoder	hActReqDec.h
DtActionRequestEncoder	hActReqEnc.h
DtActionRequestInteraction	hActReqInter.h
DtActionRequestPdu	actionPdu.h
DtActionRequestRPdu	actReqRPdu.h
DtActionRequestToObjDecoder	hActReqObjDec.h
DtActionRequestToObjEncoder	hActReqObjEnc.h
DtActionRequestToObjInter	hActReqObjInt.h
DtActionResponseDecoder	hActRespDec.h
DtActionResponseEncoder	hActRespEnc.h
DtActionResponseInteraction	hActRespInter.h
DtActionResponseFromObjDecoder	hActRspObjDec.h
DtActionResponseFromObjEncoder	hActRspObjEnc.h
DtActionResponseFromObjInter	hActRspObjInt.h
DtActionResponsePdu	actionPdu.h
DtActionResponseRPdu	actRespRPdu.h
DtAggregateEntityDecoder	aggEntDec.h

Table 1-2: Classes and their Corresponding Header Files (Continued)

Class	Header File
DtAggregateEntityEncoder	aggEntEnc.h
DtAggregatePublisher	dAggPub.h
DtAggregatePublisher	hAggPub.h
DtAggregateStateDecoder	dAggDec.h
DtAggregateStateEncoder	dAggEnc.h
DtAggregateStateRepository	aggregateSR.h
DtAggregateStatePdu	aggStatePdu.h
DtAntennaPatternRecord	antPatRec.h
DtApproximator	approximator.h
DtAppSpecificRadioSignalDecoder	hAppRadSigDec.h
DtAppSpecificRadioSignalEncoder	hAppRadSigEnc.h
DtArealObjectStatePdu	arealObjStPdu.h
DtArtPart	artPart.h
DtArtPartList	artPartList.h
DtArtPartThresholder	artPartThresh.h
DtAttachedPartList	attPartList.h
DtAttributeChangeReqDec	hAttChgReqDec.h
DtAttributeChangeReqEnc	hAttChgReqEnc.h
DtAttributeChangeReqInter	hAttChgReqInt.h
DtAttributeChangeResDec	hAttChgResDec.h
DtAttributeChangeResEnc	hAttChgResEnc.h
DtAttributeChangeResInter	hAttChgResInt.h
DtBaseEmbeddedSystemDecoder	baseEmbedDec.h
DtBaseEmbeddedSystemEncoder	baseEmbedEnc.h
DtBaseEmitterBeamDecoder	baseEmittDec.h
DtBaseEmitterBeamEncoder	baseEmittEnc.h
DtBaseEntityDecoder	baseEntDec.h
DtBaseEntityEncoder	baseEntEnc.h

Table 1-2: Classes and their Corresponding Header Files (Continued)

Class	Header File
DtBaseEntityStateRepository	baseEntitySR.h
DtBaseExerciseConn	baseExConn.h
DtBaseHashKey	baseHashKey.h
DtBaseMsg	baseMsg.h
DtBaseReflectedAggregate	baseReflAgg.h
DtBaseReflectedEntity	baseReflEnt.h
DtBasicFixedWingAircraftRecord	gedBasFixWing.h
DtBasicGroundCombatSoldierRecord	gedBasSoldier.h
DtBasicGroundCombatVehicleRecord	gedBasVehicle.h
DtBasicRotorWingAircraftRecord	gedBasRotWing.h
DtBeamAntennaPatternRecord	beamAntRec.h
DtBurstDescriptor	hostStructs.h
DtCartesianView	cartView.h
DtClientSocket	ClientSocket.h
DtCollisionDecoder	hCollideDec.h
DtCollisionEncoder	hCollideEnc.h
DtCollisionElasticPdu	collisElasPdu.h
DtCollisionInteraction	hCollideInter.h
DtCollisionPdu	collidePdu.h
DtCommentDecoder	hCommentDec.h
DtCommentEncoder	hCommentEnc.h
DtCommentInteraction	hCommentInter.h
DtCommentPdu	commentPdu.h
DtCommentRPdu	commentRPdu.h
DtCompassVCPdu	vcPdu.h
DtCoordTransform	LibMatrix.h
DtCoordTranslation	LibMatrix.h
DtCreateEntityDecoder	hCreatEntDec.h

Table 1-2: Classes and their Corresponding Header Files (Continued)

Class	Header File
DtCreateEntityEncoder	hCreatEntEnc.h
DtCreateEntityInteraction	hCreatEntInt.h
DtCreateEntityPdu	createPdu.h
DtCreateEntityRPdu	createEntRPdu.h
DtCreateObjReqDecoder	hCrtObjReqDec.h
DtCreateObjReqEncoder	hCrtObjReqEnc.h
DtCreateObjReqInter	hCrtObjReqInt.h
DtCreateObjectResultDecoder	hCrtObjResDec.h
DtCreateObjectResultEncoder	hCrtObjResEnc.h
DtCreateObjectResultInter	hCrtObjResInt.h
DtCsFilterTest	ClientSocket.h
DtDatabaseIndexRadioSignalDecoder	hDbiRadSigDec.h
DtDatabaseIndexRadioSignalEncoder	hDbiRadSigEnc.h
DtDataDecoder	hDataDec.h
DtDataEncoder	hDataEnc.h
DtDataInteration	hDataInter.h
DtDataPdu	dataPdu.h
DtDataQueryDecoder	hDataQDec.h
DtDataQueryEncoder	hDataQEnc.h
DtDataQueryInteraction	hDataQInter.h
DtDataQueryPdu	queryPdu.h
DtDataQueryRPdu	dataQueryRPdu.h
DtDataRPdu	dataRPdu.h
DtDatumMap	datumMap.h
DtDatumMapEntry	datumMap.h
DtDatumRecordSet	datumRecSet.h
DtVarDatumSpec	
DtDcm	dcm.h

Table 1-2: Classes and their Corresponding Header Files (Continued)

Class	Header File
DtDeadReckoner	deadReckoner.h
DtStateDecoderFactory	decFactory.h
DtDesignatorDecoder	designatorDec.h
DtDesignatorEncoder	designatorEnc.h
DtDesignatorPdu	designatePdu.h
DtDesignatorPublisher	hDesignatrPub.h
DtDesignatorRepository	designatorSR.h
DtDetonationDecoder	hDetDec.h
DtDetonationEncoder	hDetEnc.h
DtDetonationInteraction	hDetInter.h
DtDetonationPdu	detonatePdu.h
DtDirectedRotRate	directRotRate.h
DtDirectRtiAmbassador	dirRtiAmb.h
DtDiscoverObjectCbInfo	dscvObjCbInfo.h
DtEntityPublisher	dEntityPub.h
DtExerciseConn	dExerciseConn.h
DtDisPduFactory	disPduFactory.h
DtReflectedEntity	dReflectedEnt.h
DtReflectedEntityList	dReflEntList.h
DtDmeGlobalFcns	datumMap.h
DtElectromagneticEmissionPdu	elecEmitPdu.h
DtEmbeddedSystemRepository	embedSystemSR.h
DtEmissionPdu	emissionPdu.h
DtEmitterBeamPublisher	hEmitrBeamPub.h
DtEmitterBeamRepository	emitttrBeamSR.h
DtEmitterSystemDecoder	emitterSysDec.h
DtEmitterSystemEncoder	emitterSysEnc.h
DtEmitterSystemPublisher	hEmittrSysPub.h

Table 1-2: Classes and their Corresponding Header Files (Continued)

Class	Header File
DtEmitterSystemRepository	emitterSysSR.h
DtEmptyFomMapper	emptyFomMap.h
DtEncodedAudioRadioSignalDecoder	hEncARSigDec.h
DtEncodedAudioRadioSignalEncoder	hEncARSigEnc.h
DtEnhancedFixedWingAircraftRecord	gedEnhFixWing.h
DtEnhancedGroundCombatSoldierRecord	gedEnhSoldier.h
DtEnhancedGroundCombatVehicleRecord	gedEnhVehicle.h
DtEnhancedRotorWingAircraftRecord	gedEnhRotWing.h
DtEnvironmentalEntityDecoder	envDec.h
DtEnvironmentalEntityEncoder	envEnc.h
DtEntityIdentifier	hostStructs.h
DtEntityPublisher	dEntityPub.h hEntityPub.h
DtEntityStateDecoder	dEntityDec.h
DtEntityStateEncoder	dEntityEnc.h
DtEntityStatePdu	esPdu.h
DtEntityStateRepository	entitySR.h
DtEntityStateUpdatePdu	esUpdatePdu.h
DtEntityType	hostStructs.h
DtEnvironmentalProcessPdu	envProcPdu.h
DtEnvironmentTypeRecord	enviroTypeRec.h
DtEventId	eventId.h
DtEventReportDecoder	hEventRepDec.h
DtEventReportEncoder	hEventRepEnc.h
DtEventReportInteraction	hEventReplnt.h
DtEventReportPdu	eventReptPdu.h
DtEventReptRPdu	eventReptRPdu.h
DtExerciseConn	hExerciseConn.h

Table 1-2: Classes and their Corresponding Header Files (Continued)

Class	Header File
DtFederateId	hostStructs.h
DtFedReader	fedReader.h
DtFedTarget	fedTarget.h
DtFilename	filename.h
DtFilterTest	Socket.h
DtFireDecoder	hFireDec.h
DtFireEncoder	hFireEnc.h
DtFireInteraction	hFireInter.h
DtFirePdu	firePdu.h
DtFom	fom.h
DtFomMapper	fomMapper.h
DtFundamentalParameterData	elecEmitPdu.h
DtFundamentalParameterData	emissionPdu.h
DtGenericObjectDecoder	genericDec.h
DtGenericObjectEncoder	genericEnc.h
DtGenericObjectPublisher	genericPub.h
DtGenericStateRepository	genericSR.h
DtGeodeticCoord	geodCoord.h
DtGlobalIdHashList	globldHLst.h
DtGlobalObjectDesignator	globalObjDes.h
DtGlobalObjectDesignatorList	objDesList.h
DtGridAxisRecordFixed	gridAxiRecFix.h
DtGridAxisRecordVariable	gridAxiRecVar.h
DtGridDataRecord	gridDataRec.h
DtGridDataRecordType0	gridDataRecT0.h
DtGridDataRecordType1	gridDataRecT1.h
DtGridDataRecordType2	gridDataRecT2.h
DtGriddedDataPdu	gridDataPdu.h

Table 1-2: Classes and their Corresponding Header Files (Continued)

Class	Header File
DtGroundLogisticsVehicleRecord	gedLogVehicle.h
DtGroupVCPdu	vcPdu.h
DtHashedItem	DtOldHashlist , DtOldIntrusiveHashlist
DtHashItem	hashItem.h
DtHashlist	hashlist.h
DtHlaObject	hlaObj.h
DtHlaObjectFactory	hlaObjFact.h
DtHlaObjectWithStateRep	hlaObjWithSR.h
DtHlaStateDecoder	hStateDecoder.h
DtHlaStateEncoder	hStateEncoder.h
DtIffAtcNavAidsBeamData	iffAtcNavBeam.h
DtIffAtcNavAidsLayerHeader	iffAtcNavLayr.h
DtIffAtcNavAidsOperational	iffAtcNavOper.h
DtIffAtcNavAidsParameter	iffAtcNavParm.h
DtIffAtcNavAidsPdu	iffAtcNavPdu.h
DtIffAtcNavAidsSecondOperational	iffAtcNavOpr2.h
DtIffAtcNavAidsSystemID	iffAtcNavSyID.h
DtInteraction	dInteraction.h
DtInteraction	hInteraction.h
DtInteractionDecoder	interDecoder.h
DtInteractionDecoderFactory	intDecFact.h
DtInteractionEncoder	interEncoder.h
DtInteractionEncoderFactory	intEncFact.h
DtInteractionFactory	hInterFactory.h
DtInteractionManager	interMgr.h
DtInteractionWithData	hInterWData.h
DtInteractionWithEncDec	hIntWEncDec.h
DtInterClassDesc	intClassDesc.h

Table 1-2: Classes and their Corresponding Header Files (Continued)

Class	Header File
DtIntercomControlPdu	intercmCtlPdu.h
DtIntercomEntityDestRec	intercmEntDst.h
DtIntercomGroupAsgmtRec	intercmGrpAsg.h
DtIntercomGroupDestRec	intercmGrpDst.h
DtIntercomParmRec	intercmParRec.h
DtIntercomSignalPdu	intercmSigPdu.h
DtIntHashKey	intHashKey.h
DtIntrusiveList	vList.h
DtIsGroupOfPdu	isGroupOfPdu.h
DtIsPartOfPdu	isPartOfPdu.h
DtJammerBeamDecoder	jammerBeamDec.h
DtJammerBeamEncoder	jammerBeamEnc.h
DtLaserPdu	laserPdu.h
DtLeAppearancePdu	leAppearPdu.h
DtLeArticulatedPartsPdu	leArtPartsPdu.h
DtLeDeadReckoningParameters	leDeadReck.h
DtLeDetonationPdu	leDetonPdu.h
DtLeEntityID	leEntityID.h
DtLeEntityLinearVelocity	leEntityVel.h
DtLeEntityLocationEntityCoordinates	leLocEntity.h
DtLeEntityLocationWorldCoordinates	leLocWorld.h
DtLeEntityOrientation	leEntityOri.h
DtLeFirePdu	leFirePdu.h
DtLeOrientationError	leOriError.h
DtLePositionError	lePosError.h
DtLeTspiPdu	leTspiPdu.h
DtLgrControlDecoder	hLcDec.h
DtLgrControlEncoder	hLcEnc.h

Table 1-2: Classes and their Corresponding Header Files (Continued)

Class	Header File
DtLgrControlInteraction	hLcInter.h
DtLgrControlPdu	lcPdu.h
DtLgrEndPdu	lcEnd.h
DtLgrEofActionPdu	lcEofAction.h
DtLgrHeartbeatPdu	lcHeartbeat.h
DtLgrPausePdu	lcPause.h
DtLgrPlayPdu	lcPlay.h
DtLgrPlayActionPdu	lcPlayAction.h
DtLgrPbFilePdu	lcPlayFile.h
DtLgrQuitPdu	lcQuit.h
DtLgrRecordActionPdu	lcRecAction.h
DtLgrRecFilePdu	lcRecFile.h
DtLgrRecordPdu	lcRecord.h
DtLgrSetTimePdu	lcSetTime.h
DtLgrSkipTimePdu	lcSkipTime.h
DtLgrStartPdu	lcStart.h
DtLgrStopPdu	lcStop.h
DtLgrTimeScalePdu	lcTimeScale.h
DtLgrTimeoutPdu	lcTimeout.h
DtLinearObjectStatePdu	linObjStPdu.h
DtLinearSegmentParameter	linSegParam.h
DtList	vList.h
DtListedObj	vList.h
DtListItem	vList.h
DtLocalHlaObjectManager	locObjMgr.h
DtMessagePdu	messagePdu.h
DtMilitaryEntityDecoder	milEntDec.h
DtMilitaryEntityEncoder	milEntEnc.h

Table 1-2: Classes and their Corresponding Header Files (Continued)

Class	Header File
DtMilitaryPlatformEntityDecoder	milPlatDec.h
DtMilitaryPlatformEntityEncoder	milPlatEnc.h
DtMimicTrackVCPdu	vcPdu.h
DtMimicVCPdu	vcPdu.h
DtMinefieldDataPdu	mineDataPdu.h
DtMinefieldPerimeterRecord	minePerimeter.h
DtMinefieldQueryPdu	mineQueryPdu.h
DtMinefieldResponseNackPdu	mineRsNackPdu.h
DtMinefieldStatePdu	mineStatePdu.h
DtModelessVCPdu	vcPdu.h
DtModulationType	transmitPdu.h
DtMunitionEntityDecoder	munEntDec.h
DtMunitionEntityEncoder	munEntEnc.h
DtNet32Vector	NetStructs.h
DtNet64Vector	NetStructs.h
DtNetBoolean	netBoolean.h
DtNetBurstDescriptor	NetStructs.h
DtNetEntityIdentifier	NetStructs.h
DtNetEntityMarking	NetStructs.h
DtNetEntityType	NetStructs.h
DtNetEulerAngles	NetStructs.h
DtNetEventId	NetStructs.h
DtNetFederateId	NetStructs.h
DtNetFloat32	NetTypes.h
DtNetFloat64	NetTypes.h
DtNetGlobalObjectDesignator	NetStructs.h
DtNetInt16	NetTypes.h
DtNetInt32	NetTypes.h

Table 1-2: Classes and their Corresponding Header Files (Continued)

Class	Header File
DtNetRadioEntityType	NetStructs.h
DtNetSimulationAddress	NetStructs.h
DtNetSocket	NetSocket.h
DtNetTimeStamp	netTimeStamp.h
DtNetU16	NetTypes.h
DtNetU32	NetTypes.h
DtNullFederateAmbassador	nullFedAmb.h
DtObjClassDesc	objClassDesc.h
DtObjectId	hostStructs.h
DtObjectIdHashKey	objIdHashKey.h
DtObjectIdHashlist	objIdHashLst.h
DtObjectPublisher	hObjPub.h
DtObjectReflectAttributeValuesCbInfo	objRefCbInfo.h
DtObjectType	objectType.h
DtOldHashlist, DtOldIntrusiveHashlist	
In VR-Link 3.3, we significantly changed the semantics of our DtHashlist class. The new version is defined in hashlist.h. The older DtHashlist class has been renamed DtOldHashlist, and is still defined here in this file for backwards compatibility. While we suggest that you port your code to the new DtHashlist, which is cleaner and easier to use, if you do not want to make that change, you can just use DtOldHashlist wherever you used to use DtHashlist, and the rest of your old code should work.	
DtOrbitVCPdu	vcPdu.h
DtPdu	pdu.h
DtPduCallbackInfo	pduCbInfo.h
DtPduCallbackManager	pduCbMgr.h
DtPduDatumRecordSet	pduDatRecSet.h
DtPduFactory	pduFactory.h
DtPduWithData	pduWithData.h
DtPhysicalEntityDecoder	physEntDec.h
DtPhysicalEntityEncoder	physEntEnc.h

Table 1-2: Classes and their Corresponding Header Files (Continued)

Class	Header File
DtPointObjectStatePdu	pointObjStPdu.h
DtPropulsionSystem	supEmisEsProp.h
DtRadarBeamDecoder	radarBeamDec.h
DtRadarBeamEncoder	radarBeamEnc.h
DtRadioEntityType	hostStructs.h
DtRadioReceiverDecoder	radioRecvrDec.h
DtRadioReceiverEncoder	radioRecvrEnc.h
DtRadioReceiverPublisher	hRadioRcvrPub.h
DtRadioReceiverRepository	radioRecvrSR.h
DtRadioTransmitterDecoder	radioXmitDec.h
DtRadioTransmitterEncoder	radioXmitEnc.h
DtRadioTransmitterPublisher	hRadioXmitPub.h
DtRadioTransmitterRepository	radioXmitSR.h
DtRawBinaryRadioSignalDecoder	hRawBinRSDec.h
DtRawBinaryRadioSignalEncoder	hRawBinRSEnc.h
DtReceiveInteractionCbInfo	recIntrCbInfo.h
DtReceiverPdu	receivePdu.h
DtRecordPdu	recordPdu.h
DtRecordQueryPdu	recQueryPdu.h
DtRecordSet	recordSet.h
DtRecordSetPdu	recordSetPdu.h
DtReflectAttributeValuesCbInfo	refAttrCbInfo.h
DtReflectAttributeValuesCbList	refAttrCbList.h
DtReflectedAggregate	dReflectedAgg.h
DtReflectedAggregate	hReflectedAgg.h
DtReflectedAggregateList	dReflAggList.h
DtReflectedAggregateList	hReflAggList.h
DtReflectedEmitterBeamList	hRefBeamList.h

Table 1-2: Classes and their Corresponding Header Files (Continued)

Class	Header File
DtReflectedDesignatorList	hRefDesigList.h
DtReflectedEmitterSystemList	hRefEmSysList.h
DtReflectedEmitterBeam	hReflBeam.h
DtReflectedDesignator	hReflDesigntr.h
DtReflectedEmitterSystem	hReflEmitrSys.h dReflEmitrSys.h
DtReflectedEmitterSystemList	dRefEmSysList.h
DtReflectedEntity	dReflectedEnt.h hReflectedEnt.h
DtReflectedEntityList	hReflEntList.h dReflEntList.h
DtReflectedGenericObject	refGenericObj.h
DtReflectedGenericObjectList	refGenList.h
DtReflectedHlaObjectManager	refObjMgr.h
DtReflectedObject	hReflectedObj.h dReflectedObj.h
DtReflectedObjectCallbackInfo	refObjCbInfo.h
DtReflectedObjectList	hRefObjList.h dRefObjList.h
DtReflectedRadioReceiver	hReflRadRecvr.h
DtReflectedRadioTransmitter	hReflRadXmitr.h
DtReflectedRadioRecieverList	hRefRadRvList.h
DtReflectedRadioTransmitterList	hRefRadXmList.h
DtRegControlCbInfo	regCtlCbInfo.h
DtRegControlCbList	regCtlCbList.h
DtRemoveEntityDecoder	hRemovEntDec.h
DtRemoveEntityEncoder	hRemovEntEnc.h
DtRemoveEntityInteraction	hRemovEntInt.h
DtRemoveEntityPdu	removePdu.h

Table 1-2: Classes and their Corresponding Header Files (Continued)

Class	Header File
DtRemoveEntityRPdu	removeEntRPdu.h
DtRemoveObjectCbInfo	remObjCbInfo.h
DtRemoveObjectRequestDecoder	hRemObjReqDec.h
DtRemoveObjectRequestEncoder	hRemObjReqEnc.h
DtRemoveObjectRequestInter	hRemObjReqInt.h
DtRemoveObjectResultDecoder	hRemObjResDec.h
DtRemoveObjectResultEncoder	hRemObjResEnc.h
DtRemoveObjectResultInter	hRemObjResInt.h
DtRepairCompleteDecoder	hRepCompDec.h
DtRepairCompleteEncoder	hRepCompEnc.h
DtRepairCompleteInteraction	hRepComplInter.h
DtRepairCompletePdu	repairPdu.h
DtRepairResponseDecoder	hRepRespDec.h
DtRepairResponseEncoder	hRepRespEnc.h
DtRepairResponseInteraction	hRepResplInter.h
DtRepairResponsePdu	repairPdu.h
DtResupplyCancelDecoder	hResCancDec.h
DtResupplyCancelEncoder	hResCancEnc.h
DtResupplyCancelInteraction	hResCancInter.h
DtResupplyCancelPdu	resupplyPdu.h
DtResupplyOfferDecoder	hResOffDec.h
DtResupplyOfferEncoder	hResOffEnc.h
DtResupplyOfferInteraction	hResOffInter.h
DtResupplyOfferPdu	resupplyPdu.h
DtResupplyReceivedDecoder	hResRecDec.h
DtResupplyReceivedEncoder	hResRecEnc.h
DtResupplyRecInteraction	hResRecInter.h
DtResupplyReceivedPdu	resupplyPdu.h

Table 1-2: Classes and their Corresponding Header Files (Continued)

Class	Header File
DtRprFomMapper	rprFomMap.h
DtRprHlaObjectFactory	rprObjFact.h
DtRprInteractionDecoderFactory	rprlDecFact.h
DtRprInteractionEncoderFactory	rprlEncFact.h
DtRprInteractionFactory	rprInterFact.h
DtRprStateDecoderFactory	rprDecFact.h
DtRprStateEncoderFactory	rprEncFact.h
DtScript	script.h
DtSelectParam	vlutil.h
DtServiceRequestDecoder	hServReqDec.h
DtServiceRequestEncoder	hServReqEnc.h
DtServiceRequestInteraction	hServReqInter.h
DtServiceRequestPdu	servReqPdu.h
DtSetDataDecoder	hSetDataDec.h
DtSetDataEncoder	hSetDataEnc.h
DtSetDataInteraction	hSetDataInter.h
DtSetDataPdu	setDataPdu.h
DtSetDataRPdu	setDataRPdu.h
DtSetRecordPdu	setRecordPdu.h
DtSignalInteraction	hSignalInter.h
DtSignalPdu	signalPdu.h
DtSilentAggregateList	aggAggList.h
DtSilentEntityList	aggEntList.h
DtSimanPdu	simanPdu.h
DtDimpleRprFomMapper	simRprFomMap.h
DtSmoother	smoother.h
DtSmoothOrient	smoothUtil.h
DtSmoothOriWithDr	smoothOriWDR.h

Table 1-2: Classes and their Corresponding Header Files (Continued)

Class	Header File
DtSmoothPos	smoothUtil.h
DtSmoothPosWithDr	smoothPosWDR.h
DtSocket	Socket.h
DtSocketPairSocket	pairSock.h
DtSoldierDecoder	soldierDec.h
DtSoldierEncoder	soldierEnc.h
DtStartResumeDecoder	hStartDec.h
DtStartResumeEncoder	hStartEnc.h
DtStartResumeInteraction	hStartInter.h
DtStartResumePdu	startPdu.h
DtStartResumeRPdu	startResRPdu.h
DtStartStopInterCbInfo	stIntCbInfo.h
DtStartStopInterCbList	stIntCbList.h
DtStateEncoderFactory	encFactory.h
DtStateMsg	dStateMsg.h
DtStateMsg	hStateMsg.h
DtStateRepository	stateRep.h
DtStopFreezeDecoder	hStopDec.h
DtStopFreezeEncoder	hStopEnc.h
DtStopFreezeInteraction	hStopInter.h
DtStopFreezePdu	stopPdu.h
DtStopFreezeRPdu	stopFrzRPdu.h
DtString	str.h
DtStringHashKey	strHashKey.h
DtSuppEmisEntStatePdu	supEmisEsPdu.h
DtSystemHashKey	sysHashKey.h
DtTaitBryan	taitBryan.h
DtTcpForwarder	tcpForwarder.h

Table 1-2: Classes and their Corresponding Header Files (Continued)

Class	Header File
DtTcpSocket	tcpSocket.h
DtTcpSocketMgr	tcpSocketMgr.h
DtTetherTrackVCPdu	vcPdu.h
DtTetherVCPdu	vcPdu.h
DtThresholder	thresholder.h
DtTopoView	topoView.h
DtTrackJamBeamDecoder	trkJamBeamDec.h
DtTrackJamBeamEncoder	trkJamBeamEnc.h
DtTrackJamDesignator	trackJamDes.h
DtTrackJamDesignatorList	tkJmDesLst.h
DtTrackJamTarget	elecEmitPdu.h
DtTrackJamTarget	emissionPdu.h
DtTrackVCPdu	vcPdu.h
DtTransferControlRequestPdu	transCtlPdu.h
DtTransmitterPdu	transmitPdu.h
DtUnderwaterAcousticAddtnlPassiveAct	undwaAcPasAct.h
DtUnderwaterAcousticBeam	undwaAcBeam.h
DtUnderwaterAcousticEmitterSystem	undwaAcEmitSys.h
DtUnderwaterAcousticPdu	undwaAcPdu.h
DtUnderwaterAcousticShaftSpeed	undwaAcShaft.h
DtUnknownHlaObject	unknownHObj.h
DtUnknownInteraction	hUnknownInter.h
DtUnknownPdu	unknownPdu.h
DtUnknownVCPdu	vcPdu.h
DtUpdateControlCbInfo	updCtlCbInfo.h
DtUpdateControlCbList	updCtlCbList.h

Table 1-2: Classes and their Corresponding Header Files (Continued)

Class	Header File
DtUtmCoord	utmCoord.h
DtUtmView	utmView.h
DtVariableDatumRecord	varDatumRec.h
DtVector	vlVector.h
DtVectoringNozzleSystem	supEmisEsNozz.h
DtViewControlDecoder	hVcDec.h
DtViewControlEncoder	hVcEnc.h
DtViewControlInteraction	hVcInter.h
DtViewControlPdu	vcPdu.h
DtVrIFederateAmbassador	vrlFedAmb.h
DtVrIFedTime	vrlFedTime.h
DtVrIRtiAmbassador	vrlRtiAmb.h

1.3. Header-to-Class Cross Reference

Table 1-3 contains an alphabetical list of VR-Link header files, along with the classes they contain. The names of the header files are links to the header file code.

Table 1-3: Header Files and the Classes they Contain

Header File	Class
absNotifyObs.h	DtAbsNotifyObserver
ackPdu.h	DtAcknowledgePdu
ackRPdu.h	DtAcknowledgeRPdu
actionPdu.h	DtActionRequestPdu DtActionResponsePdu
actReqRPdu.h	DtActionRequestRPdu
actRespRPdu.h	DtActionResponseRPdu
aggAggList.h	DtSilentAggregateList
aggEntDec.h	DtAggregateEntityDecoder
aggEntEnc.h	DtAggregatedEntityEncoder
aggEntList.h	DtSilentEntityList
aggregateSR.h	DtAggregateStateRepository
aggStatePdu.h	DtAggregateStatePdu
antPatRec.h	DtAntennaPatternRecord
approximator.h	DtApproximator
arealObjStPdu.h	DtArealObjectStatePdu
artPart.h	DtArtPart
artPartList.h	DtArtPartList
artPartThresh.h	DtArtPartThresholder
attPartList.h	DtAttachedPartList
baseEmbedDec.h	DtBaseEmbeddedSystemDecoder
baseEmbedEnc.h	DtBaseEmbeddedSystemEncoder
baseEmitttrDec.h	DtBaseEmitterBeamDecoder
baseEmitttrEnc.h	DtBaseEmitterBeamEncoder
baseEntDec.h	DtBaseEntityDecoder

Table 1-3: Header Files and the Classes they Contain (Continued)

Header File	Class
baseEntEnc.h	DtBaseEntityEncoder
baseEntitySR.h	DrBaseEntityStateRepository
baseExConn.h	DtBaseExerciseConn
baseHashKey.h	DtBaseHashKey
baseMsg.h	DtBaseMsg
baseReflAgg.h	DtBaseReflectedAggregate
baseReflEnt.h	DtBaseReflectedEntity
beamAntRec.h	DtBeamAntennaPatternRecord
cartView.h	DtCartesianView
ClientSocket.h	DtClientSocket DtCsFilterTest
collidePdu.h	DtCollisionPdu
collisElasPdu.h	DtCollisionElasticPdu
commentPdu.h	DtCommentPdu
commentRPdu.h	DtCommentRPdu
createEntRPdu.h	DtCreateEntityRPdu
createPdu.h	DtCreateEntityPdu
dAggDec.h	DtAggregateStateDecoder
dAggEnc.h	DtAggregateStateEncoder
dAggPub.h	DtAggregatePublisher
dataPdu.h	DtDataPdu
dataQueryRPdu.h	DtDataQueryRPdu
dataRPdu.h	DtDataRPdu
datumMap.h	DtDatumMap DtDatumMapEntry DtDmeGlobalFcns
datumRecSet.h	DtVarDatumSpec DtDatumRecordSet
dcm.h	DtDcm
deadReckoner.h	DtDeadReckoner

Table 1-3: Header Files and the Classes they Contain (Continued)

Header File	Class
decFactory.h	DtStateDecoderFactory
dEntityDec.h	DtEntityStateDecoder
dEntityEnc.h	DtEntityStateEncoder
dEntityPub.h	DtEntityPublisher
designatePdu.h	DtDesignatorPdu
designatorDec.h	DtDesignatorDecoder
designatorEnc.h	DtDesignatorEncoder
designatorSR.h	DtDesignatorRepository
detonatePdu.h	DtDetonationPdu
dExerciseConn.h	DtExerciseConn
dInteraction.h	DtInteraction
directRotRate.h	DtDirectedRotRate
dirRtiAmb.h	DtDirectRtiAmbassador
disPduFactory.h	DtDisPduFactory
dRefEmSysList.h	DtReflectedEmitterSystemList
dRefIAggList.h	DtReflectedAggregateList
dReflectedAgg.h	DtReflectedAggregate
dReflectedEnt.h	DtReflectedEntity
dReflectedObj.h	DtReflectedObject
dRefIEmitrSys.h	DtReflectedEmitterSystem
dRefIEntList.h	DtReflectedEntityList
dRefObjList.h	DtReflectedObjectList
dscvObjCbInfo.h	DtDiscoverObjectCbInfo
dStateMsg.h	DtStateMsg
elecEmitPdu.h	DtElectromagneticEmissionPdu DtFundamentalParameterData DtTrackJamTarget
embedSystemSR.h	DtEmbeddedSystemRepository

Table 1-3: Header Files and the Classes they Contain (Continued)

Header File	Class
emissionPdu.h	DtEmissionPdu DtFundamentalParameterData DtTrackJamTarget
emitterSysDec.h	DtEmitterSystemDecoder
emitterSysEnc.h	DtEmitterSystemEncoder
emitterSysSR.h	DtEmitterSystemRepository
emitttrBeamSR.h	DtEmitterBeamRepository
emptyFomMap.h	DtEmptyFomMapper
encFactory.h	DtStateEncoderFactory
entitySR.h	DtEntityStateRepository
envDec.h	DtEnvironmentalEntityDecoder
envEnc.h	DtEnvironmentalEntityEncoder
enviroTypeRec.h	DtEnvironmentTypeRecord
envProcPdu.h	DtEnvironmentalProcessPdu
esPdu.h	DtEntityStatePdu
esUpdatePdu.h	DtEntityStateUpdatePdu
eventId.h	DtEventId
eventReptPdu.h	DtEventReportPdu
eventReptRPdu.h	DtEventReptRPdu
fedReader.h	DtFedReader
fedTarget.h	DtFedTarget
filename.h	DtFilename
firePdu.h	DtFirePdu
fom.h	DtFom
fomMapper.h	DtFomMapper
gedBasFixWing.h	DtBasicFixedWingAircraftRecord
gedBasRotWing.h	DtBasicRotorWingAircraftRecord
gedBasSoldier.h	DtBasicGroundCombatSoldierRecord
gedBasVehicle.h	DtBasicGroundCombatVehicleRecord

Table 1-3: Header Files and the Classes they Contain (Continued)

Header File	Class
gedEnhFixWing.h	DtEnhancedFixedWingAircraftRecord
gedEnhRotWing.h	DtEnhancedRotorWingAircraftRecord
gedEnhSoldier.h	DtEnhancedGroundCombatSoldierRecord
gedEnhVehicle.h	DtEnhancedGroundCombatVehicleRecord
gedLogVehicle.h	DtGroundLogisticsVehicleRecord
genericDec.h	DtGenericObjectDecoder
genericEnc.h	DtGenericObjectEncoder
genericPub.h	DtGenericObjectPublisher
genericSR.h	DtGenericStateRepository
geodCoord.h	DtGeodeticCoord
globalObjDes.h	DtGlobalObjectDesignator
globldHLst.h	DtGlobalIdHashList
gridAxiRecFix.h	DtGridAxisRecordFixed
gridAxiRecVar.h	DtGridAxisRecordVariable
gridDataPdu.h	DtGriddedDataPdu
gridDataRec.h	DtGridDataRecord
gridDataRecT0.h	DtGridDataRecordType0
gridDataRecT1.h	DtGridDataRecordType1
gridDataRecT2.h	DtGridDataRecordType2
hAckDec.h	DtAcknowledgeDecoder
hAckEnc.h	DtAcknowledgeEncoder
hAckInter.h	DtAcknowledgeInteraction
hActReqDec.h	DtActionRequestDecoder
hActReqEnc.h	DtActionRequestEncoder
hActReqInter.h	DtActionRequestInteraction
hActReqObjDec.h	DtActionRequestToObjDecoder
hActReqObjEnc.h	DtActionRequestToObjEncoder
hActReqObjInt.h	DtActionRequestToObjInter

Table 1-3: Header Files and the Classes they Contain (Continued)

Header File	Class
hActRespDec.h	DtActionResponseDecoder
hActRespEnc.h	DtActionResponseEncoder
hActRespInter.h	DtActionResponseInteraction
hActRspObjDec.h	DtActionResponseFromObjDecoder
hActRspObjEnc.h	DtActionResponseFromObjEncoder
hActRspObjInt.h	DtActionResponseFromObjInter
hAggPub.h	DtAggregatePublisher
hAppRadSigDec.h	DtAppSpecificRadioSignalDecoder
hAppRadSigEnc.h	DtAppSpecificRadioSignalEncoder
hashItem.h	DtHashItem
hashlist.h	DtHashlist
hAttChgReqDec.h	DtAttributeChangeReqDec
hAttChgReqEnc.h	DtAttributeChangeReqEnc
hAttChgReqInt.h	DtAttributeChangeReqInter
hAttChgResDec.h	DtAttributeChangeResDec
hAttChgResEnc.h	DtAttributeChangeResEnc
hAttChgResInt.h	DtAttributeChangeResInter
hCollideDec.h	DtCollisionDecoder
hCollideEnc.h	DtCollisionEncoder
hCollideInter.h	DtCollisionInteraction
hCommentDec.h	DtCommentDecoder
hCommentEnc.h	DtCommentEncoder
hCommentInter.h	DtCommentInteraction
hCreatEntDec.h	DtCreateEntityDecoder
hCreatEntEnc.h	DtCreateEntityEncoder
hCreatEntInt.h	DtCreateEntityInteraction
hCrtObjReqDec.h	DtCreateObjReqDecoder
hCrtObjReqEnc.h	DtCreateObjReqEncoder

Table 1-3: Header Files and the Classes they Contain (Continued)

Header File	Class
hCrtObjReqInt.h	DtCreateObjReqInter
hCrtObjResDec.h	DtCreateObjectResultDecoder
hCrtObjResEnc.h	DtCreateObjectResultEncoder
hCrtObjResInt.h	DtCreateObjectResultInter
hDataDec.h	DtDataDecoder
hDataEnc.h	DtDataEncoder
hDataInter.h	DtDataInteraction
hDataQDec.h	DtDataQueryDecoder
hDataQEnc.h	DtDataQueryEncoder
hDataQInter.h	DtDataQueryInteraction
hDbiRadSigDec.h	DtDatabaseIndexRadioSignalDecoder
hDbiRadSigEnc.h	DtDatabaseIndexRadioSignalEncoder
hDesignatrPub.h	DtDesignatorPublisher
hDetDec.h	DtDetonationDecoder
hDetEnc.h	DtDetonationEncoder
hDetInter.h	DtDetonationInteraction
hEmitrBeamPub.h	DtEmitterBeamPublisher
hEmittrSysPub.h	DtEmitterSystemPublisher
hEncARSigDec.h	DtEncodedAudioRadioSignalDecoder
hEncARSigEnc.h	DtEncodedAudioRadioSignalEncoder
hEntityPub.h	DtEntityPublisher
hEventRepDec.h	DtEventReportDecoder
hEventRepEnc.h	DtEventReportEncoder
hEventReplnt.h	DtEventReportInteraction
hExerciseConn.h	DtExerciseConn
hFireDec.h	DtFireDecoder
hFireEnc.h	DtFireEncoder
hFireInter.h	DtFireInteraction

Table 1-3: Header Files and the Classes they Contain (Continued)

Header File	Class
hInteraction.h	DtInteraction
hInterFactory.h	DtInteractionFactory
hInterWData.h	DtInteractionWithData
hIntWEncDec.h	DtInteractionWithEncDec
hlaObj.h	DtHlaObject
hlaObjFact.h	DtHlaObjectFactory
hlaObjWithSR.h	DtHlaObjectWithStateRep
hLcDec.h	DtLgrControlDecoder
hLcEnc.h	DtLgrControlEncoder
hLcInter.h	DtLgrControlInteraction
hObjPub.h	DtObjectPublisher
hostStructs.h	DtBurstDescriptor DtEntityIdentifier DtEntityType DtFederateId DtObjectId DtRadioEntityType
hRadioRcvrPub.h	DtRadioReceiverPublisher
hRadioXmitPub.h	DtRadioTransmitterPublisher
hRawBinRSDec.h	DtRawBinaryRadioSignalDecoder
hRawBinRSEnc.h	DtRawBinaryRadioSignalEncoder
hRefBeamList.h	DtReflectedEmitterBeamList
hRefDesigList.h	DtReflectedDesignatorList
hRefEmSysList.h	DtReflectedEmitterSystemList
hRefIAggList.h	DtReflectedAggregateList
hRefIBeam.h	DtReflectedEmitterBeam
hRefIDesigntr.h	DtReflectedDesignator
hReflectedAgg.h	DtReflectedAggregate
hReflectedObj.h	DtReflectedObject
hRefIEmitrSys.h	DtReflectedEmitterSystem

Table 1-3: Header Files and the Classes they Contain (Continued)

Header File	Class
hReflectedEnt.h	DtReflectedEntity
hRefEntList.h	DtReflectedEntityList
hRefRadRecvr.h	DtReflectedRadioReceiver
hRefRadXmitr.h	DtReflectedRadioTransmitter
hRefObjList.h	DtReflectedObjectList
hRefRadRvList.h	DtReflectedRadioReceiverList
hRefRadXmList.h	DtReflectedRadioTransmitterList
hRemObjReqDec.h	DtRemoveObjectRequestDecoder
hRemObjReqEnc.h	DtRemoveObjectRequestEncoder
hRemObjReqInt.h	DtRemoveObjectRequestInter
hRemObjResDec.h	DtRemoveObjectResultDecoder
hRemObjResEnc.h	DtRemoveObjectResultEncoder
hRemObjResInt.h	DtRemoveObjectResultInter
hRemovEntDec.h	DtRemoveEntityDecoder
hRemovEntEnc.h	DtRemoveEntityEncoder
hRemovEntInt.h	DtRemoveEntityInteraction
hRepCompDec.h	DtRepairCompleteDecoder
hRepCompEnc.h	DtRepairCompleteEncoder
hRepCompInter.h	DtRepairCompleteInteraction
hRepRespDec.h	DtRepairResponseDecoder
hRepRespEnc.h	DtRepairResponseEncoder
hRepRespInter.h	DtRepairResponseInteraction
hResCancDec.h	DtResupplyCancelDecoder
hResCancEnc.h	DtResupplyCancelEncoder
hResCancInter.h	DtResupplyCancelInteraction
hResOffDec.h	DtResupplyOfferDecoder
hResOffEnc.h	DtResupplyOfferEncoder
hResOffInter.h	DtResupplyOfferInteraction

Table 1-3: Header Files and the Classes they Contain (Continued)

Header File	Class
hResRecDec.h	DtResupplyReceivedDecoder
hResRecEnc.h	DtResupplyReceivedEncoder
hResRecInter.h	DtResupplyRecInteraction
hServReqDec.h	DtServiceRequestDecoder
hServReqEnc.h	DtServiceRequestEncoder
hServReqInter.h	DtServiceRequestInteraction
hSetDataDec.h	DtSetDataDecoder
hSetDataEnc.h	DtSetDataEncoder
hSetDataInter.h	DtSetDataInteraction
hSignalInter.h	DtSignalInteraction
hStartDec.h	DtStartResumeDecoder
hStartEnc.h	DtStartResumeEncoder
hStartInter.h	DtStartResumeInteraction
hStateDecoder.h	DtHlaStateDecoder
hStateEncoder.h	DtHlaStateEncoder
hStateMsg.h	DtStateMsg
hStopDec.h	DtStopFreezeDecoder
hStopEnc.h	DtStopFreezeEncoder
hStopInter.h	DtStopFreezeInteraction
hUnknownInter.h	DtUnknownInteraction
hVcDec.h	DtViewControlDecoder
hVcEnc.h	DtViewControlEncoder
hVcInter.h	DtViewControlInteraction
iffAtcNavBeam.h	DtIffAtcNavAidsBeamData
iffAtcNavLayr.h	DtIffAtcNavAidsLayerHeader
iffAtcNavOper.h	DtIffAtcNavAidsOperational
iffAtcNavOpr2.h	DtIffAtcNavAidsSecondOperational
iffAtcNavParm.h	DtIffAtcNavAidsParameter

Table 1-3: Header Files and the Classes they Contain (Continued)

Header File	Class
iffAtcNavPdu.h	DtIffAtcNav aidsPdu
iffAtcNavSysID.h	DtIffAtcNav aidsSystemID
intClassDesc.h	DtInterClassDesc
intDecFact.h	DtInteractionDecoderFactory
intEncFact.h	DtInteractionEncoderFactory
intercmCtlPdu.h	DtIntercomControlPdu
intercmEntDst.h	DtIntercomEntityDestRec
intercmGrpAsg.h	DtIntercomGroupAsgmtRec
intercmGrpDst.h	DtIntercomGroupDestRec
intercmParRec.h	DtIntercomParmRec
intercmSigPdu.h	DtIntercomSignalPdu
interDecoder.h	DtInteractionDecoder
interEncoder.h	DtInteractionEncoder
interMgr.h	DtInteractionManager
intHashKey.h	DtIntHashKey
isGroupOfPdu.h	DtIsGroupOfPdu
isPartOfPdu.h	DtIsPartOfPdu
jammerBeamDec.h	DtJammerBeamDecoder
jammerBeamEnc.h	DtJammerBeamEncoder
laserPdu.h	DtLaserPdu
lcEnd.h	DtLgrEndPdu
lcEofAction.h	DtLgrEofActionPdu
lcHeartbeat.h	DtLgrHeartbeatPdu
lcPause.h	DtLgrPausePdu
lcPdu.h	DtLgrControlPdu
lcPlay.h	DtLgrPlayPdu
lcPlayAction.h	DtLgrPlayActionPdu
lcPlayFile.h	DtLgrPbFilePdu

Table 1-3: Header Files and the Classes they Contain (Continued)

Header File	Class
IcQuit.h	DtLgrQuitPdu
IcRecAction.h	DtLgrRecordActionPdu
IcRecFile.h	DtLgrRecFilePdu
IcRecord.h	DtLgrRecordPdu
IcSetTime.h	DtLgrSetTimePdu
IcSkipTime.h	DtLgrSkipTimePdu
IcStart.h	DtLgrStartPdu
IcStop.h	DtLgrStopPdu
IcTimeScale.h	DtLgrTimeScalePdu
IcTimeout.h	DtLgrTimeoutPdu
IeAppearPdu.h	DtLeAppearancePdu
IeArtPartsPdu.h	DtLeArticulatedPartsPdu
IeDeadReck.h	DtLeDeadReckoningParameters
IeDetonPdu.h	DtLeDetonationPdu
IeEntityID.h	DtLeEntityID
IeEntityOri.h	DtLeEntityOrientation
IeEntityVel.h	DtLeEntityLinearVelocity
IeFirePdu.h	DtLeFirePdu
IeLocEntity.h	DtLeEntityLocationEntityCoordinates
IeLocWorld.h	DtLeEntityLocationWorldCoordinates
IeOriError.h	DtLeOrientationError
IePosError.h	DtLePositionError
IeTspiPdu.h	DtLeTspiPdu
LibMatrix.h	DtCoordTransform DtCoordTranslation
linObjStPdu.h	DtLinearObjectStatePdu
linSegParam.h	DtLinearSegmentParameter
locObjMgr.h	DtLocalHlaObjectManager

Table 1-3: Header Files and the Classes they Contain (Continued)

Header File	Class
messagePdu.h	DtMessagePdu
milEntDec.h	DtMilitaryEntityDecoder
milEntEnc.h	DtMilitaryEntityEncoder
milPlatDec.h	DtMilitaryPlatformEntityDecoder
milPlatEnc.h	DtMilitaryPlatformEntityEncoder
mineDataPdu.h	DtMinefieldDataPdu
minePerimeter.h	DtMinefieldPerimeterRecord
mineQueryPdu.h	DtMinefieldQueryPdu
mineRsNackPdu.h	DtMinefieldResponseNackPdu
mineStatePdu.h	DtMinefieldStatePdu
munEntDec.h	DtMunitionEntityDecoder
munEntEnc.h	DtMunitionEntityEncoder
netBoolean.h	DtNetBoolean
NetSocket.h	DtNetSocket
NetStructs.h	DtNet32Vector DtNet64Vector DtNetBurstDescriptor DtNetEntityIdentifier DtNetEntityMarking DtNetEntityType DtNetEulerAngles DtNetEventId DtNetFederateId DtNetGlobalObjectDesignator DtNetRadioEntityType DtNetSimulationAddress
netTimeStamp.h	DtNetTimeStamp
NetTypes.h	DtNetFloat32 DtNetFloat64 DtNetInt16 DtNetInt32 DtNetU16 DtNetU32
nullFedAmb.h	DtNullFederateAmbassador
objClassDesc.h	DtObjClassDesc

Table 1-3: Header Files and the Classes they Contain (Continued)

Header File	Class
objDesList.h	DtGlobalObjectDesignatorList
objectType.h	DtObjectType
objIdHashKey.h	DtObjectIdHashKey
objIdHashLst.h	DtObjectIdHashlist
objRefCbInfo.h	DtObjectReflectAttributeValuesCbInfo
oldHashList.h	In VR-Link 3.3, we significantly changed the semantics of our DtHashlist class. The new version is defined in hashlist.h. The older DtHashlist class has been renamed DtOldHashlist, and is still defined here in this file for backwards compatibility. While we suggest that you port your code to the new DtHashlist, which is cleaner and easier to use, if you do not want to make that change, you can just use DtOldHashlist wherever you used to use DtHashlist, and the rest of your old code should work.
pairSock.h	DtSocketPairSocket
pdu.h	DtPdu
pduCbInfo.h	DtPduCallbackInfo
pduCbMgr.h	DtPduCallbackManager
pduDatRecSet.h	DtPduDatumRecordSet
pduFactory.h	DtPduFactory
pduWithData.h	DtPduWithData
physEntDec.h	DtPhysicalEntityDecoder
physEntEnc.h	DtPhysicalEntityEncoder
pointObjStPdu.h	DtPointObjectStatePdu
queryPdu.h	DtDataQueryPdu
radarBeamDec.h	DtRadarBeamDecoder
radarBeamEnc.h	DtRadarBeamEncoder
radioRecvrDec.h	DtRadioReceiverDecoder
radioRecvrEnc.h	DtRadioReceiverEncoder
radioRecvrSR.h	DtRadioReceiverRepository
radioXmitDec.h	DtRadioTransmitterDecoder

Table 1-3: Header Files and the Classes they Contain (Continued)

Header File	Class
radioXmitEnc.h	DtRadioTransmitterEncoder
radioXmitSR.h	DtRadioTransmitterRepository
receivePdu.h	DtReceiverPdu
recIntrCbInfo.h	DtReceiveInteractionCbInfo
recordPdu.h	DtRecordPdu
recordSet.h	DtRecordSet
recordSetPdu.h	DtRecordSetPdu
recQueryPdu.h	DtRecordQueryPdu
refAttrCbInfo.h	DtReflectAttributeValuesCbInfo
refAttrCbList.h	DtReflectAttributeValuesCbList
refGenericObj.h	DtReflectedGenericObject
refGenList.h	DtReflectedGenericObjectList
refObjCbInfo.h	DtReflectedObjectCallbackInfo
refObjMgr.h	DtReflectedHlaObjectManager
regCtlCbInfo.h	DtRegControlCbInfo
regCtlCbList.h	DtRegControlCbList
remObjCbInfo.h	DtRemoveObjectCbInfo
removeEntRPdu.h	DtRemoveEntityRPdu
removePdu.h	DtRemoveEntityPdu
repairPdu.h	DtRepairCompletePdu DtRepairResponsePdu
resupplyPdu.h	DtResupplyCancelPdu DtResupplyOfferPdu DtResupplyReceivedPdu
rprDecFact.h	DtRprStateDecoderFactory
rprEncFact.h	DtRprStateEncoderFactory
rprFomMap.h	DtRprFomMapper
rprlDecFact.h	DtRprlInteractionDecoderFactory
rprlEncFact.h	DtRprlInteractionEncoderFactory

Table 1-3: Header Files and the Classes they Contain (Continued)

Header File	Class
rprInterFact.h	DtRprInteractionFactory
rprObjFact.h	DtRprHlaObjectFactory
script.h	DtScript
servReqPdu.h	DtServiceRequestPdu
setDataPdu.h	DtSetDataPdu
setDataRPdu.h	DtSetDataRPdu
setRecordPdu.h	DtSetRecordPdu
signalPdu.h	DtSignalPdu
simanPdu.h	DtSimanPdu
simRprFomMap.h	DtDimpleRprFomMapper
smoother.h	DtSmoother
smoothOriWDR.h	DtSmoothOriWithDr
smoothPosWDR.h	DtSmoothPosWithDr
smoothUtil.h	DtSmoothOrient DtSmoothPos
smUtil.h	DtAckPredOpt
Socket.h	DtFilterTest DtSocket
soldierDec.h	DtSoldierDecoder
soldierEnc.h	DtSoldierEncoder
startPdu.h	DtStartResumePdu
startResRPdu.h	DtStartResumeRPdu
stateRep.h	DtStateRepository
stIntCbInfo.h	DtStartStopInterCbInfo
stIntCbList.h	DtStartStopInterCbList
stopFrzRPdu.h	DtStopFreezeRPdu
stopPdu.h	DtStopFreezePdu
str.h	DtString
strHashKey.h	DtStringHashKey

Table 1-3: Header Files and the Classes they Contain (Continued)

Header File	Class
supEmisEsNozz.h	DtVectoringNozzleSystem
supEmisEsPdu.h	DtSuppEmisEntStatePdu
supEmisEsProp.h	DtPropulsionSystem
sysHashKey.h	DtSystemHashKey
taitBryan.h	DtTaitBryan
tcpForwarder.h	DtTcpForwarder
tcpSocket.h	DtTcpSocket
tcpSocketMgr.h	DtTcpSocketMgr
threshold.h	DtThresholder
tkJmDesLst.h	DtTrackJamDesignatorList
topoView.h	DtTopoView
trackJamDes.h	DtTrackJamDesignator
transCtlPdu.h	DtTransferControlRequestPdu
transmitPdu.h	DtModulationType DtTransmitterPdu
trkJamBeamDec.h	DtTrackJamBeamDecoder
trkJamBeamEnc.h	DtTrackJamBeamEncoder
tStampType.h	DtTimeStampType
undwaAcBeam.h	DtUnderwaterAcousticBeam
undwaAcEmitSys.h	DtUnderwaterAcousticEmitterSystem
undwaAcPasAct.h	DtUnderwaterAcousticAddtnlPassiveAct
undwaAcPdu.h	DtUnderwaterAcousticPdu
undwaAcShaft.h	DtUnderwaterAcousticShaftSpeed
unknownHObj.h	DtUnknownHlaObject
unknownPdu.h	DtUnknownPdu
updCtlCbInfo.h	DtUpdateControlCbInfo
updCtlCbList.h	DtUpdateControlCbList
utmCoord.h	DtUtmCoord

Table 1-3: Header Files and the Classes they Contain (Continued)

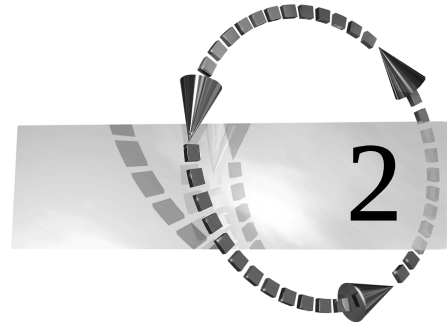
Header File	Class
utmView.h	DtUtmView
varDatumRec.h	DtVariableDatumRecord
vcPdu.h	DtAbsoluteVCPdu DtCompassVCPdu DtGroupVCPdu DtMimicTrackVCPdu DtMimicVCPdu DtModelessVCPdu DtOrbitVCPdu DtTetherTrackVCPdu DtTetherVCPdu DtTrackVCPdu DtUnknownVCPdu DtViewControlPdu
vList.h	DtIntrusiveList DtList DtListItem DtListedObj
vlutil.h	DtSelectParam
vlVector.h	DtVector
vrlFedAmb.h	DtVrlFederateAmbassador
vrlFedTime.h	DtVrlFedTime
vrlRtiAmb.h	DtVrlRtiAmbassador

1.4. Other Header Files

Table 1-4 lists header files that do not contain classes.

Table 1-4: Other Header Files

Header File	Contents
datumIdType.h	Definitions and functions that map from DtDatumParam to type enumerations.
disEnums.h	Contains enumerations specified by the DIS protocol and RPR FOM.
EntityType.h	Contains enumerations of entity-type fields.
enum2string.h	Contains functions that map from DtPduKind enumerations to strings.
Euler.h	Contains functions that convert between Euler angles and transformation matrices.
MachineTypes.h	Defines primitive data types in a machine-independent manner.
mathTypes.h	Contains math macros, includes other math-related headers.
printUtil.h	Defines printing functions.
ProcControl.h	Defines functions for sleeping and polling stdin.
topoCoord.h	Defines functions to generate matrices that convert between geodetic and topographic orientation.
vlprint.h	Contains functions related to the printing of notification messages.
vlTime.h	Contains functions related to time.



PDU Class Reference

This appendix contains descriptive notes pertaining to DIS 2.0.3, 2.0.4 (IEEE1278.1) and 2.1.4 (1278.1a) PDUs. For information about class derivation of PDU classes, see the class documentation and class hierarchy diagram in the *./doc* directory.

<i>DtAcknowledgePdu</i>	2-2	<i>DtPduWithData</i>	2-41
<i>DtActionRequestPdu</i>	2-4	<i>DtReceiverPdu</i>	2-48
<i>DtActionResponsePdu</i>	2-6	<i>DtRemoveEntityPdu</i>	2-49
<i>DtCollisionPdu</i>	2-8	<i>DtRepairCompletePdu</i>	2-50
<i>DtCommentPdu</i>	2-10	<i>DtRepairResponsePdu</i>	2-51
<i>DtCreateEntityPdu</i>	2-12	<i>DtResupplyCancelPdu</i>	2-52
<i>DtDataPdu</i>	2-13	<i>DtResupplyOfferPdu</i>	2-53
<i>DtDataQueryPdu</i>	2-14	<i>DtResupplyReceivedPdu</i>	2-55
<i>DtDesignatorPdu</i>	2-16	<i>DtSetDataPdu</i>	2-57
<i>DtDetonationPdu</i>	2-18	<i>DtSignalPdu</i>	2-58
<i>DtEmissionPdu</i>	2-21	<i>DtSimanPdu</i>	2-60
<i>DtEntityStatePdu</i>	2-26	<i>DtStartResumePdu</i>	2-62
<i>DtEventReportPdu</i>	2-32	<i>DtStopFreezePdu</i>	2-64
<i>DtFirePdu</i>	2-33	<i>DtTransmitterPdu</i>	2-66
<i>DtLaserPdu</i>	2-35	<i>DtViewControlPdu</i>	2-70
<i>DtMessagePdu</i>	2-36		

DtAcknowledgePdu

Name

DtAcknowledgePdu (DIS Acknowledge PDU)

Description

Inspecting and mutating PDU header data is done through *DtPdu* member functions.

Constructors

Standard constructors derived from *DtPdu*:

- ♦ Blank PDU constructor
- ♦ From-network-representation constructor.

Class-specific constructors:

- ♦ The constructor creates a blank *DtAcknowledgePdu* using `buffer`, then sets the originating entity ID to “from”, the receiving entity ID to “to”, the acknowledge flag to “aFlag” and the response flag to “rFlag”.

Functions

Function	Description
senderId() receiverId() setSenderId() setReceiverId()	Inherited from <i>DtSimanPdu</i> .
acknowledgeFlag() responseFlag()	Returns the value of the flags.
setAcknowledgeFlag()	Sets the acknowledge flag to “flag”.
setResponseFlag()	Sets the response flag to “flag”.
requestId() setRequestId()	Inspect and set the request ID field.



requestId() and setRequestId() are unavailable when linking with *libv13*.

See Also

[DtSimanPdu](#), this PDU's header file ([ackPdu.h](#)), and the description of *DtPdu* in Chapter 7, "The DIS-Specific Interface" in the *VR-Link Developer's Guide*.

DtActionRequestPdu

Name

DtActionRequestPdu (DIS Action Request PDU)

Description

Inspecting and mutating PDU header data is done through *DtPdu* member functions.

Constructors

Standard constructors derived from *DtPdu*:

- ♦ Blank PDU constructor
- ♦ From-network-representation constructor.

Functions

Function	Description
senderId() receiverId() setSenderId() setReceiverId()	Inherited from DtSimanPdu .
numFixedFields() numVarFields() setNumFixedFields() setNumVarFields() setDatumParam() setVarDataBytes() initDataSection() datumSizeBits() datumSizeBytes() datumId() getIndex() fixedDatumId() varDatumId() datumVal() functions setDatumVal() functions	Inherited from DtPduWithData .
requestId() actionId() setRequestId() setActionId()	Inspect and set their PDU fields

See Also

[DtSimanPdu](#), [DtPduWithData](#), this PDU's header file ([actionPdu.h](#)), and the description of *DtPdu* in Chapter 7, "The DIS-Specific Interface" in the *VR-Link Developer's Guide*.

DtActionResponsePdu

Name

DtActionResponsePdu (DIS Action Response PDU)

Description

Inspecting and mutating PDU header data is done through *DtPdu* member functions.

Constructors

Standard constructors derived from *DtPdu*:

- ♦ Blank PDU constructor
- ♦ From-network-representation constructor.

Functions

Function	Description
senderId() receiverId() setSenderId() setReceiverId()	Inherited from DtSimanPdu .
numFixedFields() numVarFields() setNumFixedFields() setNumVarFields() setDatumParam() setVarDataBytes() initDataSection() datumSizeBits() datumSizeBytes() datumId() getIndex() fixedDatumId() varDatumId() datumVal() functions setDatumVal() functions	Inherited from DtPduWithData .
requestId() requestStatus() setRequestId() setRequestStatus()	Inspect and set their PDU fields

See Also

[DtPduWithData](#), [DtSimanPdu](#), this PDU's header file ([actionPdu.h](#)), and the description of *DtPdu* in Chapter 7, "The DIS-Specific Interface" in the *VR-Link Developer's Guide*.

DtCollisionPdu

Name

DtCollisionPdu (DIS Collision PDU)

Description

Inspecting and mutating PDU header data is done through *DtPdu* member functions.

Constructors

Standard constructors derived from *DtPdu*:

- ♦ Blank PDU constructor
- ♦ From-network-representation constructor.

Functions

Function	Description
issueld() collidelid() eventld() velocity() mass() entityCoords() setIssuelid() setCollidelid() setEventld() setVelocity() setMass() setEntityCoords()	Returns the value of the appropriate data.
collisionType() setCollisionType()	Inspect and set the value of the collision type field. This field was introduced in the IEEE 1278.1-1995 version of the DIS protocol, replacing 32 bits of padding.

Notes:

- ♦ Since C and C++ do not allow assignment to arrays, you must use *DtVecCopy* for copying the values returned by *velocity()* and *entityCoords()* into local variables:

```
DtCollisionPdu pdu;  
DtVector vec;  
vec = pdu.velocity();           /* ILLEGAL!!! */  
DtVecCopy(pdu.velocity(), vec); /* CORRECT    */
```

- ♦ `collisionType()` and `setCollisionType()` are not available when using DIS 2.0.3 (linking with *libvl3.a*). When linking with *libvl.a*, when the protocol version of a *DtCollisionPdu* is 4 (indicating DIS 2.0.4), these functions print a warning as they access a place in the PDU structure that is supposed to be padding in DIS 2.0.4.

See Also

This PDU's header file ([collidePdu.h](#)), and the description of *DtPdu* in Chapter 7, "The DIS-Specific Interface" in the *VR-Link Developer's Guide*.

DtCommentPdu

Name

DtCommentPdu (DIS Comment PDU)

Description

Inspecting and mutating PDU header data is done through *DtPdu* member functions.

The Comment PDU was introduced in DIS version 5 (1278.1-1995). It was previously known as the Message PDU. When using DIS 2.0.3 or 2.0.4, *DtMessagePdu* should be instead, as *DtCommentPdu* is unavailable.

Constructors

Standard constructors derived from *DtPdu*:

- ♦ Blank PDU constructor
- ♦ From-network-representation constructor.

Class-specific constructors:

- ♦ The constructor takes a sender ID, receiver ID, and a text message. The PDU created will have one variable datum record which contains the message.

Functions

Function	Description
senderId() receiverId() setSenderId() setReceiverId()	Inherited from DtSimanPdu .
numFixedFields() numVarFields() setNumFixedFields() setNumVarFields() setDatumParam() setVarDataBytes() initDataSection() datumSizeBits() datumSizeBytes() datumId() getIndex() fixedDatumId() varDatumId() datumVal() functions setDatumVal() functions	Inherited from DtPduWithData .

Function	Description
<code>setComment()</code>	Takes a text message and a size of this message as arguments. After a call to this function, the PDU has one variable length datum record that contains the message.

See Also

[DtSimanPdu](#), [DtPduWithData](#), [DtMessagePdu](#), this PDU's header file ([commentPdu.h](#)), and the description of *DtPdu* in Chapter 7, "The DIS-Specific Interface" in the *VR-Link Developer's Guide*.

DtCreateEntityPdu

Name

DtCreateEntityPdu (DIS Create Entity PDU)

Description

Inspecting and mutating PDU header data is done through *DtPdu* member functions.

Constructors

Standard constructors derived from *DtPdu*:

- ♦ Blank PDU constructor
- ♦ From-network-representation constructor.

Class-specific constructors:

- ♦ The constructor creates a blank *DtCreateEntityPdu* using `buffer`, then sets the originating entity ID to “from”, and the receiving entity ID to “to”.

Functions

Function	Description
<code>senderId()</code> <code>receiverId()</code> <code>setSenderId()</code> <code>setReceiverId()</code>	Inherited from DtSimanPdu .
<code>requestId()</code> <code>setRequestId()</code>	Inspect and set this field.



`requestId()` and `setRequestId()` are unavailable when linking with *libv13*.

See Also

[DtSimanPdu](#), this PDU’s header file ([createPdu.h](#)), and the description of *DtPdu* in Chapter 7, "The DIS-Specific Interface" in the *VR-Link Developer’s Guide*.

DtDataPdu

Name

DtDataPdu (DIS Data PDU)

Description

Inspecting and mutating PDU header data is done through *DtPdu* member functions.

Constructors

Standard constructors derived from *DtPdu*:

- ♦ Blank PDU constructor
- ♦ From-network-representation constructor.

Functions

Function	Description
senderId() receiverId() setSenderId() setReceiverId()	Inherited from DtSimanPdu .
numFixedFields() numVarFields() setNumFixedFields() setNumVarFields() setDatumParam() setVarDataBytes() initDataSection() datumSizeBits() datumSizeBytes() datumId() getIndex() fixedDatumId() varDatumId() datumVal() functions setDatumVal() functions	Inherited from DtPduWithData .
requestId() setRequestId()	Inspect and set this field.

See Also

[DtSimanPdu](#), [DtPduWithData](#), this PDU's header file ([dataPdu.h](#)), and the description of *DtPdu* in Chapter 7, "The DIS-Specific Interface" in the *VR-Link Developer's Guide*.

DtDataQueryPdu

Name

DtDataQueryPdu (DIS Data Query PDU)

Description

Inspecting and mutating PDU header data is done through *DtPdu* member functions.

Constructors

Standard constructors derived from *DtPdu*:

- ♦ Blank PDU constructor
- ♦ From-network-representation constructor.

Functions

Function	Description
senderId() receiverId() setSenderId() setReceiverId()	Inherited from DtSimanPdu .
requestId() timeInterval()	Returns the value of those data. The value returned by <code>timeInterval()</code> is in seconds.
numFixedFields()	Returns the number of fixed datum IDs in the PDU.
numVarFields()	Returns the number of variable datum IDs in the PDU.
fixedDatumId()	Returns the <i>n</i> th fixed datum ID in the PDU, where <i>n</i> = <code>index</code> . <code>DtDatumParam(0)</code> is returned if <code>index</code> is either less than 1 or greater than the number of fixed datum IDs in the PDU.
varDatumId()	Returns the <i>n</i> th variable datum ID in the PDU, where <i>n</i> = <code>index</code> . <code>DtDatumParam(0)</code> is returned if <code>index</code> is either less than 1 or greater than the number of variable datum IDs in the PDU.
setRequestId() setTimeInterval()	Sets the value of those data in the PDU.
initDatumIds()	Sets the number of fixed datum IDs in the PDU to <code>numFixedFields</code> and the number of variable datum IDs to <code>numVarFields</code> . It changes the size of the PDU to accommodate the new number of fixed and variable fields. It also sets all fixed and variable datum IDs to <code>DtDatumParam(0)</code> .
setFixedDatumId()	Sets the <i>n</i> th fixed datum ID to <code>id</code> , where <i>n</i> = <code>index</code> .

Function	Description
setVarDatumId()	Sets the <i>n</i> th variable datum ID to <i>id</i> , where <i>n</i> = <i>index</i> .

See Also

[DtSimanPdu](#), this PDU's header file ([queryPdu.h](#)), and the description of *DtPdu* in Chapter 7, "The DIS-Specific Interface" in the *VR-Link Developer's Guide*.

DtDesignatorPdu

Name

DtDesignatorPdu (DIS Designator PDU)

Description

Inspecting and mutating PDU header data is done through *DtPdu* member functions.



This class is unavailable when linking with *libv13*.

Constructors

Standard constructors derived from *DtPdu*:

- ♦ Blank PDU constructor
- ♦ From-network-representation constructor.

Functions

Function	Description
designatingEntityId() codeName() designatedEntityId() designatorCode() power() wavelength() designatorSpotRelative() designatorSpotWorld()	Returns the value of the appropriate data.
setDesignatingEntityId() setCodeName() setDesignatedEntityId() setDesignatorCode() setPower() setWavelength() setDesignatorSpotRelative() setDesignatorSpotWorld()	Sets the value of the appropriate data.
deadReckoningAlgorithm() setDeadReckoningAlgorithm() linearAcceleration() setLinearAcceleration()	Inspect and set fields that were introduced in the IEEE 1278.1-1995 version of the DIS protocol.

i

The dead reckoning and linear acceleration functions are not available when using DIS 2.0.3 (linking with *libvl3.a*). When linking with *libvl.a*, when the protocol version of a *DtCollisionPdu* is 4 (indicating DIS 2.0.4), these functions print a warning as they access fields that do not exist in DIS 2.0.4. In this case, `deadReckoningAlgorithm()` returns 0, and `linearAcceleration()` returns {0,0,0}. The corresponding mutator functions have no effect on the PDU.

See Also

[*DtLaserPdu*](#), this PDU's header file ([designatePdu.h](#)), and the description of *DtPdu* in Chapter 7, "The DIS-Specific Interface" in the *VR-Link Developer's Guide*.

DtDetonationPdu

Name

DtDetonationPdu (DIS Detonation PDU)

Description

Inspecting and mutating PDU header data is done through *DtPdu* member functions.

Constructors

Standard constructors derived from *DtPdu*:

- ♦ Blank PDU constructor
- ♦ From-network-representation constructor.

Class-specific constructors:

- ♦ The constructor creates a blank *DtDetonationPdu* using **buffer**, then calls *DtDetonationPdu* mutator functions to set the values of each of the data items specified as arguments. It also sets the number of articulation parameters and initializes those parameters by passing **specList** to **setArtParts()**.

Functions

Function	Description
attackerId() targetId() munitionId() eventId() velocity() worldLocation() burst() entityLocation() result() numArtParams()	Returns the value of the appropriate data.
artPartSpecs()	Returns an array of <i>DtArtPartSpecs</i> , representing the articulation parameters contained in the PDU (up to 25 articulated parts). The array is an exact copy of the last array of <i>DtArtPartSpecs</i> with which the PDU was initialized (either through a call to setArtParts() , or by passing it to a <i>DtDetonationPdu</i> constructor).

Function	Description
getArtParamVal()	The two overloaded versions of getArtParamVal() fills value with the value of a particular articulation parameter. The first is used to select an articulation parameter by index (the <i>n</i>th parameter in the PDU, where <i>n</i> = paramIndex); the second is used to select a parameter by part type and parameter type. The first function returns zero if paramIndex is out of bounds (that is, less than 1 or greater than the number of parameters in this PDU), one otherwise. The second returns zero if no articulation parameter exists in this PDU with the specified part type and parameter type, otherwise it returns one.
setAttackerId() setTargetId() setMunitionId() setEventId() setVelocity() setWorldLocation() burst() setEntityLocation() setResult()	Sets the appropriate PDU data to the values passed as arguments.
setArtParts()	Changes the size of the PDU and sets the number of articulation parameters to reflect specList . It also sets the part type, and part-attached-to fields of each articulation record appropriately.
setArtParamVal()	The two overloaded versions of setArtParamVal() set the value of a particular articulation parameter to value . Using the first, an articulation parameter is chosen by part type and parameter type. In the second, the <i>n</i> th articulation parameter is chosen, where <i>n</i> = paramIndex .

specList is an array of *DtArtPartSpecs*, each of which describes one articulated part (which may have several articulation parameters associated with it).

A *DtArtPartSpec* has the following form:

```
typedef struct
{
    int partType;
    int attachedType;
    int params[DtMaxNumArtParts];
} DtArtPartSpec;
```

where **partType** is the DIS enumeration for the part type, **attachedType** is the DIS enumeration for the parent part type, (or the token *DtAttachedToBase* if the part is attached to the entity base), and **params** is a NULL terminated list of articulation parameters that are to be used for this part.

The last element of the array of *DtArtPartSpecs* must be *DtNullArtPartSpec*, indicating that the previous *DtArtPartSpec* described the last articulated part. If **specList** is NULL, the list contains no articulated parts.

The array of *DtArtPartSpecs* can be defined statically, for example:

```
int numParts = 2;
static DtArtPartSpec specList[numParts +1] =
{
    {
        DtPrimaryTurret1,
        DtAttachedToBase,
        {DtApAzimuth, DtApAzimuthRate, 0}
    },
    {
        DtPrimaryGun1,
        DtPrimaryTurret1,
        {DtApElevation, 0}
    },
    {DtNullArtPartSpec}
};
```

See Also

[artPart.h](#), this PDU's header file ([detonatePdu.h](#)), and the description of *DtPdu* in Chapter 7, "The DIS-Specific Interface" in the *VR-Link Developer's Guide*.

DtEmissionPdu

Name

DtEmissionPdu (DIS Emission PDU)

Description

Inspecting and mutating PDU header data is done through *DtPdu* member functions.



This class is only available when linking with *libvl3*. It is replaced by *DtElectromagneticEmissionPdu* in *libvl*.

Constructors

Standard constructors derived from *DtPdu*:

- ♦ Blank PDU constructor
- ♦ From-network-representation constructor.

Functions

Function	Description
emittingEntityId() eventId() stateUpdateIndicator() numSystems()	Returns the value of the appropriate data.
systemDataWords() numBeams() systemName() systemFunction() emitterId() location()	Inspects data associated with a particular emitter system record. The emitter system to inspect is dictated by the sysNum argument to each of these functions. Emitter system records are numbered 0 through <code>numSystem() - 1</code>. For the meaning of the retcode argument, see “Return Codes” on page 2-25.

Function	Description
beamDataWords() beamId() beamParameterIndex() parameterData() beamFunction() numTargetsInBeam() trackingAllTargets() jammingModeSequence()	Inspects data associated with a particular beam belonging to a particular emitter system (each emitter system record may contain one or more beam records). The particular emitter system record to inspect is dictated by the sysNum argument to each of these functions. The particular beam record within the specified emitter system record to inspect is dictated by the beamNum argument. Emitter system records are numbered 0 through numSystems() - 1. Beam records within a particular emitter system record sysNum are numbered 0 through numBeams(sysNum). For the meaning of the retcode argument, see "Return Codes" on page 2-25.
trackJamTarget()	A particular emitter beam record may contain one or more track/jam targets records. The function trackJamTarget() is used to inspect a particular target record. sysNum and beamNum identify the particular emitter system and beam within that system whose target we would like to inspect. targetNum identifies a particular target within the beam record. Target records within a particular beam record indexed as described above by sysNum and beamNum , are indexed 0 through numTargetsInBeam(sysNum, beamNum). For the meaning of the retcode argument, see "Return Codes" on page 2-25.
setEmittingEntityId() setEventId() setStateUpdateIndicator()	Sets the value of the appropriate fields.
setNumSystems()	Resizes the PDU to accommodate num emitter system records. If num is less than the number of emitter system records previously in the PDU, trailing system records are deleted. Otherwise, additional emitter system records are created at the end of the PDU. The newly created emitter system records will each contain zero emitter beam records, and the system data size fields are set to reflect this.
setSystemName() setSystemFunction() setEmitterId() setLocation()	Sets the appropriate fields of the emitter system record indexed by sysNum . sysNum has the same meaning as in the corresponding inspector functions described above. For the meaning of the retcode argument, see "Return Codes" on page 2-25.

Function	Description
setNumBeams()	Resizes the PDU, and the emitter system record indexed by sysNum to accommodate the new number of beam records. If numBeams is less than the previous number of beams in the system record, trailing beam records are deleted. Otherwise, the appropriate number of new beam records are created following the last existing beam record. These new beam records will contain zeros in all fields, and will have no track/jam target records. The data sizes in the beam records, system records, and PDU header will all remain consistent after a call to setNumBeams() . For the meaning of the retcode argument, see "Return Codes" on page 2-25.
setBeamId() setBeamParameterIndex() setParameterData() setBeamFunction() setTrackingAllTargets() setJammingModeSequence()	Sets the appropriate values with a beam records indexed by sysNum and beamNum() . sysNum and beamNum have the same meanings as in the corresponding inspector functions described above. For the meaning of the retcode argument, see "Return Codes" on page 2-25. The <i>DtFundamentalParameterData</i> class is described above. The individual components of the object may be set directly before it is passed to setParameterData() .
setNumTargetsInBeam()	Resizes the PDU, beam record, and emitter system record to accommodate the new number of track/jam target records. If num is less than the previous number of targets, trailing target records are deleted. Otherwise additional target records are inserted, following the last existing record. New track/jam target records created will have all values set to zero. The data sizes in the beam records, system records, and PDU header will all remain consistent after a call to setNumTargetsInBeam() . For the meaning of the retcode argument, see "Return Codes" on page 2-25.
setTrackJamTarget()	Sets the values of a particular track/jam target record to the data contained in targ . The <i>DtTrackJamTarget</i> class is described above. Its individual components may be set directly before it is passed to setTrackJamTarget() . sysNum , beamNum and targetNum have the same meanings as in trackJamTarget() . For the meaning of the retcode argument, see "Return Codes" on page 2-25.

Function	Description
status()	Returns one of the following values: DtSTATUS_OK if the values of all of the system data lengths, beam data lengths, number of targets and PDU length are all consistent. DtINCONSISTENT_SYS_LENGTHS_PDU_LENGTH if the system record data lengths are inconsistent with the PDU length. DtINCONSISTENT_BEAM_LENGTHS_SYS_LENGTH if for any emitter system record, the beam data lengths are inconsistent with the system data length. DtINCONSISTENT_NUM_TARGETS_BEAM_LENGTH if for any beam record, the number of targets is inconsistent with the beam record length.

The value returned by parameterData() is a reference to an object of type *DtFundamentalParameterData*. *DtFundamentalParameterData* has the following definition, and the values of the individual components can be inspected directly:

```
class DtFundamentalParameterData
{
    public:
        float frequency;
        float frequencyRange;
        float effectiveRadiatedPower;
        float pulseRepetitionFrequency;
        float pulseWidth;
        float beamAzCenter;
        float beamAzSweep;
        float beamElCenter;
        float beamElSweep;
        float beamSweepSync;
};
```

The value returned by trackJamTarget() is a reference to an object of type *DtTrackJamTarget*. Its definition appears below, and individual components may be inspected directly:

```
class DtTrackJamTarget
{
    public:
        DtEntityIdentifier entityId;
        int emitterId;
        int beamId;
};
```

Return Codes

Many inspector and mutator functions have an optional **retcode** argument. If it is non-NULL, the integer to which it points is set to indicate whether the inspection or mutation of the PDU was successful. For inspector functions, a return code of DtEMISSION_SUCCESS indicates that the value returned is valid. For mutator functions, a return code of DtEMISSION_SUCCESS indicates that the PDU has been successfully mutated. Any other value to which the integer to which **retcode** points might be set, indicates failure for one of the reasons cited below.

For functions that have a **sysNum** argument, DtBAD_SYS_NUM indicates that **sysNum** is not a valid emitter system record index for the PDU. DtBAD_SYS_ADDR indicates that although **sysNum** is within the legal range, system data lengths are inconsistent with the PDU length, and the specified emitter system records could not be found.

For functions with a **beamNum** argument, DtBAD_BEAM_NUM indicates that **sysNum** is a valid emitter system record index for the PDU but **beamNum** is not a valid beam record index for that emitter system. DtBAD_SYS_ADDR indicates that although **sysNum** and **beamNum** are within the legal ranges, and the specified emitter system record was found, the length of the data in the beam records is inconsistent with either the PDU length or the enclosing emitter system record length, and the specified beam record could not be found.

For functions with **sysNum**, **beamNum** and **targetNum** arguments, DtBAD_TARGET_NUM indicates that while **sysNum** and **beamNum** are valid, **targetNum** is not valid. DtBAD_SYS_ADDR indicates that although **sysNum**, **beamNum** and **targetNum** are within the legal ranges, and the specified beam record was found, the number of targets in this beam record is inconsistent with either the PDU length or the enclosing beam record length, and the specified target record could not be found.

See Also

This PDU's header file ([emissionPdu.h](#)) and the description of *DtPdu* in Chapter 7, "The DIS-Specific Interface" in the *VR-Link Developer's Guide*.

DtEntityStatePdu

Name

DtEntityStatePdu (DIS Entity State PDU)

Description

Inspecting and mutating PDU header data is done through *DtPdu* member functions.

Constructors

Standard constructors derived from *DtPdu*:

- ♦ Blank PDU constructor
- ♦ From-network-representation constructor.

Class-specific constructors:

- ♦ The constructors create a blank *DtEntityStatePdu* using `buffer`, then call *DtEntityStatePdu* mutator functions to set the values of each of the data items specified as arguments. For details on the meanings of each of the arguments, see the sections on the mutator functions `setId()`, `setEntityType()`, `setGuise()`, `setAlgorithm()`, `setForceId()`, and `setArtAndAttachedParts()` in this section.

After construction from a network representation, the status of the *DtEntityStatePdu* as reported by `DtPdu::status()` is `DtSTATUS_OK` unless the number of articulation parameters is inconsistent with the size of the PDU.

Functions

Function	Description
<code>id()</code>	Returns the current value of those data.
<code>forceId()</code>	All of these functions return by value or reference, so their values can be copied into local objects as follows: <pre>DtEntityStatePdu pdu; DtEntityType type = pdu.type; DtEntityIdentifier id = pdu.id();</pre> Alternatively, the data can be used immediately, as in: <pre>DtEntityStatePdu pdu1; DtEntityStatePdu pdu2; pdu2.setEntityType(pdu1.entityType());</pre> or <pre>printf("Entity Type: %s\n", pdu1.entityType().string());</pre>
<code>entityType()</code>	
<code>guise()</code>	
<code>orientation()</code>	
<code>appearance()</code>	
<code>algorithm()</code>	
<code>markingCharSet()</code>	
<code>capabilities()</code>	

Function	Description
velocity() location() acceleration() rotationalVelocity()	Returns a reference to <i>DtConstVectors</i> (which is a <i>DtVector</i>) whose components cannot be modified. Because C and C++ prohibit assignment to arrays, the following is illegal: <pre>DtEntityStatePdu pdu1; DtVector vec = pdu.location();</pre> You must use <i>DtVecCopy</i> when you need to copy the value of one of these data into a local object: <pre>DtEntityStatePdu pdu1; DtVector vec; DtVecCopy(pdu.location(), vec);</pre> You can also use the returned value directly by passing it to any function that expects a <i>DtConstVector</i> or reference to one (actually, <i>DtVecCopy</i> is just such a function).
markingText()	Returns a pointer to memory internal to <i>DtEntityStatePdu</i>. Do not save a pointer to this buffer, since it may be updated by the <i>DtEntityStatePdu</i> at any time, and will contain garbage after the <i>DtEntityStatePdu</i> is deleted. If the value of the marking text is not to be used immediately, use <code>strcpy</code> to copy it into a local buffer. The marking text will never be longer than 12 characters, and will always contain a trailing NULL character: <pre>char mark[12]; strcpy(mark, pdu.markingText());</pre>
appearanceAttribute()	Returns the current value of appearance attribute <code>attr</code>. A list of constants for many appearance attributes and values is in disEnums.h. For example: <pre>DtAppearAttrVal val = pdu.appearanceAttribute(DtAppHatch); if (val == DtHatchPopped) printf("Popped\n"); . .</pre>
numArtParams()	Returns the number of articulation parameter records in the PDU.
numAttachedParams()	Returns the number of attached part records in the PDU.
numArtPlusAttachedParams()	Returns the number of attached part records plus articulation parameter records in the PDU.

Function	Description
artPartSpecs()	Returns an array of <i>DtArtPartSpecs</i> , representing the articulation parameters contained in the PDU (up to 25 articulated parts). The array is an exact copy of the last array of <i>DtArtPartSpecs</i> with which the PDU was initialized (either through a call to <i>setArtAndAttachedParts()</i> , or by passing it to a <i>DtEntityStatePdu</i> constructor).
attachedPartStations()	Returns a null-terminated array of integers representing the attached part stations possessed by this entity. The array is an exact copy of the last array of station numbers with which the PDU was initialized (either through a call to <i>setArtAndAttachedParts()</i> or by passing it to the constructor).
getArtParamVal()	The two overloaded versions of <i>getArtParamVal()</i> fills value with the value of a particular articulation parameter. The first is used to select an articulation parameter by index (the <i>n</i>th parameter in the PDU, where <i>n</i> = paramIndex); the second is used to select a parameter by part type and parameter type. The first function returns zero if paramIndex is out of bounds (that is, less than 0 or greater than the number of parameters in this PDU), otherwise it returns one. The second returns zero if no articulation parameters exist in this PDU with the specified part type and parameter type, otherwise it returns one.
attachedPartChange()	Returns the value of the change indicator in the attached part record associated with station . If there is no record with station station , DtNO_SUCH_ATTACH_STATION is returned.
getAttachedType()	Fills type with the entity type in the type field of the attached part record whose station number is equal to station . If no such record exists, DtNO_SUCH_ATTACH_STATION is returned. Otherwise, DtATTACHED_PART_SUCCESS is returned.

Function	Description
setId() setForceId() setEntityType() setGuise() setVelocity() setLocation() setOrientation() setAppearance() setAlgorithm() setAcceleration() setRotationalVelocity() setMarkingCharSet() setCapabilities()	Sets the value of the data to their arguments.
setAppearanceAttribute()	Sets the value of appearance attribute attr to val . A list of constants for many appearance attributes and values may be found in disEnums.h . For example: <pre>pdu.setAppearanceAttribute(DtAppHatch, DtHatchPopped);</pre>
setMarkingText()	Sets the marking text to be the string contained in text and sets the character set to be charSet . The default value for charSet is 1.
setArtAndAttachedParts()	Changes the size of the PDU and sets the number of articulation parameters plus attached part records to reflect specList and stationList . It also sets the part type, and part-attached-to fields of each articulation record appropriately, and sets the station number of each attached part record appropriately. See further discussion following this table.
setArtParamVal()	The two overloaded versions of setArtParamVal() set the value of a particular articulation parameter to value . Using the first, an articulation parameter is chosen by part type and parameter type. In the second, the <i>n</i> th articulation parameter is chosen, where <i>n</i> = paramIndex . The change argument indicates the value to be placed into the Change field of the articulation record. Passing the value DtINCREMENT (which is the default), indicates that the value of the Change field should be set to one greater than its current value.
setAttachedPartChange()	Sets the value of the change indicator in the attached part record whose station is station to change . If no such record exists, DtNO_SUCH_ATTACH_STATION is returned, otherwise DtATTACHED_PART_SUCCESS is returned.

Function	Description
setAttachedPartChange()	Sets the value of the type field in the attached part record whose station is station to type . If no such record exists, DtNO_SUCH_ATTACH_STATION is returned, otherwise DtATTACHED_PART_SUCCESS is returned.

stationList is a null terminated list of attached part station numbers.

specList is an array of *DtArtPartSpecs*, each of which describes one articulated part (which may have several articulation parameters associated with it).

A *DtArtPartSpec* has the following form:

```
typedef struct
{
    int partType;
    int attachedType;
    int params[DtMaxNumArtParts];
} DtArtPartSpec;
```

where **partType** is the DIS enumeration for the part type, **attachedType** is the DIS enumeration for the parent part type, (or the token *DtAttachedToBase* if the part is attached to the entity base), and **params** is a NULL terminated list of articulation parameters that are to be used for this part.

The last element of the array of *DtArtPartSpecs* must be *DtNullArtPartSpec*, indicating that the previous *DtArtPartSpec* described the last articulated part. If **specList** is NULL, the list contains no articulated parts.

The array of *DtArtPartSpecs* can be defined statically, for example:

```
int numParts = 2;
static DtArtPartSpec specList[numParts +1] =
{
    {
        DtPrimaryTurret1,
        DtAttachedToBase,
        {DtApAzimuth, DtApAzimuthRate, 0}
    },
    {
        DtPrimaryGun1,
        DtPrimaryTurret1,
        {DtApElevation, 0}
    },
    {DtNullArtPartSpec}
};
```

See Also

[artPart.h](#), this PDU's header file ([esPdu.h](#)), and the description of *DtPdu* in Chapter 7, "The DIS-Specific Interface" in the *VR-Link Developer's Guide*.

DtEventReportPdu

Name

DtEventReportPdu (DIS Event Report PDU)

Description

Inspecting and mutating PDU header data is done through *DtPdu* member functions.

Constructors

Standard constructors derived from *DtPdu*:

- ♦ Blank PDU constructor
- ♦ From-network-representation constructor.

Functions

Function	Description
senderId() receiverId() setSenderId() setReceiverId()	Inherited from DtSimanPdu .
numFixedFields() numVarFields() setNumFixedFields() setNumVarFields() setDatumParam() setVarDataBytes() initDataSection() datumSizeBits() datumSizeBytes() datumId() getIndex() fixedDatumId() varDatumId() datumVal() functions setDatumVal() functions	Inherited from DtPduWithData .
eventType() setEventType()	Inspect and set that field in the PDU.

See Also

[DtSimanPdu](#), this PDU's header file ([eventReptPdu.h](#)), and the description of *DtPdu* in Chapter 7, "The DIS-Specific Interface" in the *VR-Link Developer's Guide*.

DtFirePdu

Name

DtFirePdu (DIS Fire PDU)

Description

Inspecting and mutating PDU header data is done through *DtPdu* member functions.

Constructors

Standard constructors derived from *DtPdu*:

- ♦ Blank PDU constructor
- ♦ From-network-representation constructor.

Class-specific constructor:

The constructor creates a blank *DtFirePdu* using *buffer*, then calls *DtFirePdu* mutator functions to set the values of each of the data items specified as arguments.

Functions

Function	Description
attacker() target() munition() eventId() location() burst velocity() range()	Returns the value of the appropriate data.
setAttacker() setTarget() setMunition() setEventId() setLocation() setBurst() setVelocity() setRange()	set the appropriate PDU data to the values passed as arguments.
fireMissionIndex() setFireMissionIndex()	Inspect and set the value of the fire mission field. This field was introduced in the IEEE 1278.1-1995 version of the DIS protocol, replacing 32 bits of padding.

Notes:

- Since C and C++ do not allow assignment to arrays, you must use *DtVecCopy* for copying the values returned by *location()* and *velocity()* into local variables:

```
DtFirePdu pdu;  
DtVector vec;  
vec = pdu.location();           /* ILLEGAL!!! */  
DtVecCopy(pdu.location(), vec); /* CORRECT    */
```

- *fireMissionIndex()* and *setFireMissionIndex()* are not available when using DIS 2.0.3 (linking with *libvl3.a*). When linking with *libvl.a*, when the protocol version of a *DtCollisionPdu* is 4 (indicating DIS 2.0.4), these functions print a warning as they access a place in the PDU structure that is supposed to be padding in DIS 2.0.4.

See Also

This PDU's header file ([firePdu.h](#)) and the description of *DtPdu* in Chapter 7, "The DIS-Specific Interface" in the *VR-Link Developer's Guide*.

DtLaserPdu

Name

DtLaserPdu (DIS Laser PDU)

Description

This class is only available when linking with *libvl3*. It is replaced by *DtDesignatorPdu* in *libvl*.

Constructors

Standard constructors derived from *DtPdu*:

- ♦ Blank PDU constructor
- ♦ From-network-representation constructor.

Functions

Function	Description
lasingEntityId() codeName() lasedEntityId() laserCode() power() wavelength() lasedSpotRelative() lasedSpotWorld()	Returns the value of the appropriate data.
setLasingEntityId() setCodeName() setLasedEntityId() setLaserCode() setPower() setWavelength() setLasedSpotRelative() setLasedSpotWorld()	Sets the value of the appropriate data.

See Also

[DtDesignatorPdu](#), this PDU's header file ([laserPdu.h](#)), and the description of *DtPdu* in Chapter 7, "The DIS-Specific Interface" in the *VR-Link Developer's Guide*.

DtMessagePdu

Name

DtMessagePdu (DIS Message PDU)

Description

Inspecting and mutating PDU header data is done through *DtPdu* member functions.

The *DtMessagePdu* class described here is available only when linking with *libvl3.a* (using DIS 2.0.3). In DIS version 5 (1278.1-1995) the Message PDU was replaced by the Comment PDU. When linking with *libvl.a*, and using DIS 2.0.4, *DtMessagePdu* is defined as:

```
typedef DtCommentPdu DtMessagePdu;
```

Constructors

Standard constructors derived from *DtPdu*:

- ♦ Blank PDU constructor
- ♦ From-network-representation constructor.

Class-specific constructors:

- ♦ The constructor can be used when the PDU is meant to convey a single text message. **message** is a null terminated character string, which is the message to be sent. The PDU will have a single variable length datum record whose length is equal to `strlen(message) + 1`. The datum parameter in the variable datum record is 0, or *DtNullDatumParam*.

Functions

Function	Description
<code>senderId()</code> <code>receiverId()</code> <code>setSenderId()</code> <code>setReceiverId()</code>	Inherited from DtSimanPdu .
<code>numVarFields()</code>	Returns the number of variable length datum fields in the PDU.
<code>datumSizeBits()</code>	Returns the size of the <code>n</code> th variable datum field is returned, where <code>n = Index</code> . If <code>Index</code> is out of bounds, 0 is returned.
<code>datumSizeBytes()</code>	Returns the value of <code>datumSizeBits()</code> , after converting to bytes.

Function	Description
setNumVarFields()	Resizes the PDU to accommodate the new number of variable length datum records. If newNum is less than the previous number of records, trailing records are deleted. If newNum is greater than the previous number of records, additional records are created after the last existing record. The newly created records will have a data size of zero, and datum parameter ID of 0.
setDatumParam()	Sets the <i>n</i> th variable datum ID in the PDU to param , where <i>n</i> = Index . Index should be between 1 and the number of variable datum fields in the PDU.
setVarDataBytes()	Changes the data size in a particular variable length datum record. The PDU is resized to reflect the change. If the new value is less than the old, trailing bytes of the data are deleted. If the new value is greater than the old, bytes containing zeros are added after the last existing byte in the record. A PDU might be constructed and filled in as follows: <pre> DtMessagePdu messagePdu; messagePdu.setNumVarFields(3); messagePdu.setDatumParam(1, DtDatumXpos); messagePdu.setDatumParam(2, DtDatumYpos); messagePdu.setDatumParam(3, DtDatumZpos); messagePdu.setVarDatumBytes(1, 8); messagePdu.setVarDatumBytes(2, 8); messagePdu.setVarDatumBytes(3, 8); /* Now use the setDatumVal functions described below to set the actual datum values */ </pre>
setMessage()	Used when the PDU is meant to contain only one variable length datum record, which contains a text message. The size of the field is set to size , and the data is set to message .

Inspecting and Setting Data Values

DtMessagePdu has many inspector and mutator functions that return and set the data value of a particular datum parameter respectively. The inspector functions' names start with "datumVal", and the mutator functions' names start with "setDatumVal()". The values of different datum parameters in a PDU can be of different data types. For each of the possible data types, we provide an inspector function that will return the value of a particular datum parameter, when that value is interpreted as belonging to that type, and a mutator function that sets the value of a particular datum parameter to a value of that type. The correct data type to be used for a particular datum parameter may be obtained using *DtDatumId2Type* (see "Data Types" on page 2-39).

Particular datum records can be identified either by datum parameter ID, or by Index in the PDU. Indices range from 1 to numVarFields().

You can go back and forth between datum parameter IDs and indices using the functions `datumId()` and `paramIndex()`. `paramIndex()` will return the index of the variable datum record where the given datum parameter ID may be found. `datumId()` returns the *n*th variable datum ID in the PDU, where *n* = `Index`. `DtNullDatumParam` is returned when `Index` is less than 1 or greater than the number of variable datum fields in the PDU.

If no datum record with datum parameter ID `param` is found in the PDU, `paramIndex()` will return 0, and the integer to which `retcode` points is set to the value `DtDATUM_BAD_ID` (provided `retcode` is not NULL). If `paramIndex()` is successful, the value pointed to by `retcode` is set to `DtDATUM_VALID`.

Below are several examples of usage of the `datumVal()` inspectors, the `setDatumVal()` mutators, and the `getIndex()` function.

```
double xval = messagePdu.datumValFloat64(DtDatumXpos);
```

will assign to `xval` the value contained in the datum record of the PDU whose datum parameter ID is `DtDatumXpos()`, interpreting that value as a 64 bit floating point number.

An alternative way of doing the same thing is:

```
int Index, retcode;
/* first get the Index of the desired datum record */
Index = messagePdu.paramIndex(DtDatumXpos, &retcode);
/* now get the value contained in that datum record */
if (retcode == DtDATUM_VALID)
    double xval = messagePdu.datumValFloat64(Index);
messagePdu.setDatumValInt32(DtDatumKind, 12);
```

will set the value of the kind parameter to 12, using a 32 bit integer representation.

```
int Index, retcode;
/* first get the Index of the desired datum record */
Index = messagePdu.paramIndex(DtDatumMass, &retcode);
/* now set that datum record's value to 2000 */
if (retcode == DtDATUM_VALID)
{
    messagePdu.setDatumValFloat32(Index, 2000.0);
}
```

i

Setting or inspecting datum values by `Index` is more efficient than by parameter ID (the latter type method first looks up the `Index` of the ID, then calls the former type of inspector (or mutator) with that `Index`). So if you plan to look up the value of the same ID more than once, it is more efficient to look up the `Index` once, then call the inspector with that `Index`.

Choosing among the different inspector and mutator functions for data types is fairly straightforward. However, the `datumValByteArray()` and `setDatumValByteArray()` functions can always be used to get a bitwise copy of a particular data value. The value returned by `datumValByteArray()` is a pointer to internal memory. If the value is not to be used immediately, the data should be copied into local memory:

```
{
    char marking[12];
    memcpy(marking,
           messagePdu.datumValByteArray(Index, 12),
           messagePdu.datumSizeBytes(Index));
}
```

Return Codes

All of the datum value inspectors and mutators have an optional return code pointer argument, **retcode**. If this pointer is non-NULL, a return code for the function is stored in **retcode**. In all cases, the return code is `DtDATUM_VALID` when the function call was successful. This means that specified datum parameter record exists in the PDU, and its value was able to be interpreted as the data type whose inspector function was used, or set to a value of the data type whose mutator function was used.

The return code is set to `DtDATUM_BAD_LENGTH` when the length of the datum value field does not match the size of the data type whose function you are using.

The return code is set to `DtDATUM_BAD_INDEX` when **Index** is out of range.

The return code is set to `DtDATUM_BAD_ID` when no datum record exists in the PDU whose parameter ID is **param**.

Data Types

VR-Link maintains a table of mappings between *DtDatumIds* and data types. The function *DtDatumId2Type* is used to look up the data type associated with a particular datum ID.

The different data type values that may be returned by *DtDatumId2Type* are:

- ♦ `DtTYPE_BYTE_ARRAY`
- ♦ `DtTYPE_FLOAT32`
- ♦ `DtTYPE_FLOAT64`
- ♦ `DtTYPE_INT32`
- ♦ `DtTYPE_UNSIGNED32`
- ♦ `DtTYPE_STRING`.

If the table does not contain a mapping for **id**, then the value `DtTYPE_NOT_DEFINED` is returned.

All DIS 2.0.3 datum IDs are entered in this table by default along with reasonable data types. For instance, *DtDatumId2Type(DtDatumXpos)* will return DtTYPE_FLOAT64 while *DtDatumId2Type(DtDatumForceId)* will return DtTYPE_FLOAT32. This function is used by, among other things, *DtMessagePdu*'s implementation of the *DtPdu* virtual function *printData()*, which needs to know the type of each datum value, so it can properly format them for printing:

```
for (int i = 1; i <= numVarFields(); i++)
{
    DtDatumParam param = datumId(i);
    switch (DtDatumId2Type(param))
    {
        case DtTYPE_FLOAT32:
            DtDumpFloat(DtEnumString(param),
                datumValFloat32(i));
            break;
        case DtTYPE_FLOAT64:
            DtDumpFloat(DtEnumString(param),
                datumValFloat32(i));
            break;
        case DtTYPE_INT32:
            DtDumpInteger(DtEnumString(param),
                datumValInt32(i));
            break;
        .
        .
        .
    }
}
```

The data type of a particular datum ID may be changed, or a mapping for a new datum ID may be added to the table, with the function *DtAddDatumId2TypeMapping()*. This function will return DtTYPE_NOT_DEFINED if there was no previous data type mapping for *id* in the table. Otherwise, it returns the old value. If the optional **warn** argument is non-zero, then a warning message is printed when an existing mapping is being overridden.

See Also

[DtSimanPdu](#), [DtPduWithData](#), [DtCommentPdu](#), this PDU's header file ([messagePdu.h](#)), and the description of *DtPdu* in Chapter 7, "The DIS-Specific Interface" in the *VR-Link Developer's Guide*.

DtPduWithData

Name

DtPduWithData

Description

DtPduWithData is an abstract class that encapsulates the functionality common to all PDUs that contain fixed and variable length datum records. Specifically, *DtPduWithData* provides accessor functions to inspect and set the values in the “data” section of these PDUs.

After an instance of a class derived from *DtPduWithData* is constructed, inspecting and mutating PDU header data is done through *DtPdu* member functions. Similarly, inspecting and mutating of data common to all simulation management PDUs is done through *DtSimanPdu* member functions, and inspecting and mutating pieces of the “data” section of the PDU is done through *DtPduWithData* member functions.

Constructors

Standard constructors derived from *DtPdu*:

- ♦ Blank PDU constructor
- ♦ From-network-representation constructor.

Functions

Function	Description
<code>numFixedFields()</code>	Returns the number of fixed length datum fields in the PDU.
<code>numVarFields()</code>	Returns the number of variable length datum fields in the PDU.
<code>setNumVarFields()</code> <code>setnumFixedFields()</code>	resize the PDU to accommodate the new number of datum records. If <code>newNum</code> is less than the previous number of records, trailing records are deleted. If <code>newNum</code> is greater than the previous number of records, additional records are created after the last existing record. Newly created variable length records will have a data size of zero. Both fixed and variable data records will have a datum parameter ID of 0.
<code>setDatumParam()</code>	Sets the <code>n</code> th fixed or variable datum ID in the PDU to <code>param</code> , where <code>n</code> = Index, depending on whether <code>fixedOrVar</code> is <code>DtFIXED</code> or <code>DtVAR</code> . Index should be between 1 and the number of fields in the PDU.

Function	Description
setVarDataBytes()	Changes the data size in a particular variable length datum record. The PDU is resized to reflect the change. If the new value is less than the old, trailing bytes of the data are deleted. If the new value is greater than the old, bytes containing zeros are added after the last existing byte in the record.
initDataSection()	Initializes all of the datum records in the PDU at once, so that calls to setnumFixedFields() and setNumVarFields(), and repeated calls to setDatumParam() and setNumVarDatumBytes() do not need to be made. Changes the size of the PDU to reflect the total size of the fixed and variable data records described by params and specList. Sets the number of fixed records, and number of variable records appropriately. fixedParams is an array of DtDatumParams to be used as fixed datum parameter IDs. The last element of the array must be <i>DtNullDatumParam</i>. specList is an array of <i>DtVarDatumSpecs</i>, each of which describes the datum parameter ID, and data size (in bits) of one variable datum record. The last element of the array must be <i>DtNullVarDatumSpec</i>. See additional discussion following this table.
datumSizeBits()	Returns the size of the datum field specified by fixedOrVar and Index . That is, if fixedOrVar is DtFIXED, 32 is returned, and if fixedOrVar is DtVAR, the size of the <i>n</i> th variable datum field is returned, where <i>n</i> = Index . In either case, if Index is out of bounds, 0 is returned. Indices range from 1 to numFixedFields() or numVarFields().
datumSizeBytes()	Returns the value of datumSizeBits(), after converting to bytes.

A *DtVarDatumSpec* looks like this:

```
typedef struct
{
    DtDatumParam param;
    int length;
} DtVarDatumSpec;
```

For example:

```
static DtDatumParam fixed[] = {DtDatumPsi, DtDatumTheta,
                               DtDatumPhi, DtNullDatumParam};
static DtVarDatumSpec var[] =
{
    {DtDatumXpos, 64},
    {DtDatumYpos, 64},
    {DtDatumZpos, 64},
    {DtNullVarDatumSpec}
```

```
};  
pdu.initDataSection(fixed, var);
```

i

The *DtNullDatumParam* and *DtNullVarDatumSpec* serve only to mark the end of the list of records, and are not meant to indicate extra records. In the example above, the number of fixed records in the PDU is 3, and the number of variable length records in the PDU is 3.

Inspecting and Setting Data Values

DtPduWithData has many inspector and mutator functions that return and set the data value of a particular datum parameter respectively. The names of inspector functions start with “datumVal”, and the names of mutator functions start with “setDatumVal()”. The values of different datum parameters in a PDU can be of different data types. For each of the possible data types, we provide an inspector function that returns the value of a particular datum parameter, when that value is interpreted as belonging to that type, and a mutator function that sets the value of a particular datum parameter to a value of that type. The correct data type to be used for a particular datum parameter can be obtained using *DtDatumId2Type* (see “Data Types” on page 2-45).

Particular datum records can be identified either by datum parameter ID, or by Index in the PDU. We use Index in a general sense here to mean a pair of indices which specify first whether you are talking about a fixed or variable datum field, and secondly the value of *n* in the expression “the *n*th fixed datum field”. Indices range from 1 to *numFixedFields* or *numVarFields*.

You can go back and forth between datum parameter IDs and indices using the functions *datumId()* and *getIndex()*. *getIndex()* fills in the arguments *fixedOrVar* and *Index* with *DtFIXED* or *DtVAR*, and an index, respectively. Taken together, these two values determine the slot in the PDU where the datum record associated with this datum parameter ID can be found.

fixedDatumId() returns the *n*th fixed datum ID in the PDU, where *n* = *Index*.

DtNullDatumParam is returned when *Index* is less than 1, or greater than the number of fixed datum fields in the PDU.

varDatumId() returns the *n*th variable datum ID in the PDU, where *n* = *Index*.

DtNullDatumParam is returned when *Index* is less than 1, or greater than the number of variable datum fields in the PDU.

datumId() returns *fixedDatumId(Index)* or *varDatumId(Index)*, depending on whether *fixedOrVar* is *DtFIXED* or *DtVAR*.

If no datum record with datum parameter ID *param* is found in the PDU, the values in *Index* and *fixedOrVar* are invalid, and the value of the pointer passed as the *retcode* argument is set to the value *DtDATUM_BAD_ID* (provided *retcode* is not NULL.) If *getIndex()* is successful, *retcode* the value pointed to by *retcode* is set to *DtDATUM_VALID*.

Several examples of usage of the datumVal() inspectors, the setDatumVal() mutators, and the getIndex() function follow:

```
double xval = dataPdu.datumValFloat64(DtDatumXpos);
```

assigns to xval the value contained in the datum record of the PDU whose datum parameter ID is *DtDatumXpos*, interpreting that value as a 64 bit floating point number.

An alternative way of doing the same thing is:

```
int fixedOrVar, Index, retcode;
/* first get the Index of the desired datum record */
pdu.getIndex(DtDatumXpos, &fixedOrVar, &Index, &retcode);
/* now get the value contained in that datum record */
if (retcode == DtDATUM_VALID)
    double xval = dataPdu.datumValFloat64(fixedOrVar,
                                           Index);
dataPdu.setDatumValInt32(DtDatumKind, 12);
```

will set the value of the kind parameter to 12, using a 32 bit integer representation.

```
int fixedOrVar, Index, retcode;
/* first get the Index of the desired datum record */
pdu.getIndex(DtDatumMass, &fixedOrVar, &Index, &retcode);
/* now set that datum record's value to 2000 */
if (retcode == DtDATUM_VALID)
{
    dataPdu.setDatumValFloat32(fixedOrVar, Index, 2000.0);
}
```

i

Setting or inspecting datum values by Index is more efficient than by parameter ID (the latter type method first looks up the Index of the ID, then calls the former type of inspector (or mutator) with that Index). So if you plan to look up the value of the same ID more than once, it is more efficient to look up the Index once, then call the inspector with that Index.

Choosing among the different inspector and mutator functions for the data types is fairly straightforward. However, the datumValByteArray() and setDatumValByteArray() functions may always be used to get a bitwise copy of a particular data value. The value returned by datumValByteArray() is a pointer to internal memory. If the value is not to be used immediately, copy the data into local memory:

```
{
    char marking[12];
    memcpy(marking,
           dataPdu.datumValByteArray(fixedOrVar, Index, 12),
           dataPdu.datumSizeBytes(fixedOrVar, Index));
}
```


Return Codes

All of the datum value inspectors and mutators have an optional return code pointer argument, **retcode**. If this pointer is non-NULL, a return code for the function is stored in **retcode**. In all cases, the return code is DtDATUM_VALID when the function call was successful. This means that a specified datum parameter record exists in the PDU, and its value was able to be interpreted as the data type whose inspector function was used, or set to a value of the data type whose mutator function was used.

The return code is set to DtDATUM_BAD_LENGTH when the length of the datum value field (32 bits for fixed data, varies for variable data) does not match the size of the data type whose function you are using. For example:

```
int retcode;
double val = pdu.datumValFloat64(DtFIXED, 1, &retcode);
```

Here, **retcode** is set to DtDATUM_BAD_LENGTH since the 32 bit fixed data field cannot be interpreted as a 64 bit floating point number.

The return code is set to DtDATUM_BAD_INDEX when **index** is out of range. For instance, if there are only 3 fixed datum records,

```
int retcode;
pdu.setDatumValFloat64(DtFIXED, 4, 725.0, &retcode);
```

causes **retcode** to contain DtDATUM_BAD_INDEX, since we have asked for the value of the 4th fixed record.

The return code is set to DtDATUM_BAD_ID when no datum record exists in the PDU whose parameter ID is **param**.

Data Types

VR-Link maintains a table of mappings between *DtDatumIds* and data types. The function *DtDatumId2Type* is used to look up the data type associated with a particular datum ID.

The different data type values that can be returned by *DtDatumId2Type* are:

- ♦ DtTYPE_BYTE_ARRAY
- ♦ DtTYPE_FLOAT32
- ♦ DtTYPE_FLOAT64
- ♦ DtTYPE_INT32
- ♦ DtTYPE_UNSIGNED32
- ♦ DtTYPE_STRING.

If the table does not contain a mapping for **id**, then the value DtTYPE_NOT_DEFINED is returned.

All DIS 2.0.3 datum IDs are entered in this table by default along with reasonable data types. For instance, *DtDatumId2Type(DtDatumXpos)* returns *DtTYPE_FLOAT64* while *DtDatumId2Type(DtDatumForceId)* returns *DtTYPE_FLOAT32*. This function is used by, among other things, *DtPduWithData*'s implementation of the *DtPdu* virtual function *printData()*, which needs to know the type of each datum value, so it can properly format them for printing:

```
for (int i = 1; i <= numVarFields(); i++)
{
    DtDatumParam param = datumId(i);
    switch (DtDatumId2Type(param))
    {
        case DtTYPE_FLOAT32:
            DtDumpFloat(DtEnumString(param),
                datumValFloat32(i));
            break;
        case DtTYPE_FLOAT64:
            DtDumpFloat(DtEnumString(param),
                datumValFloat32(i));
            break;
        case DtTYPE_INT32:
            DtDumpInteger(DtEnumString(param),
                datumValInt32(i));
            break;
        .
        .
        .
    }
}
```

The data type of a particular datum ID can be changed, or a mapping for a new datum ID can be added to the table with the function *DtAddDatumId2TypeMapping*. This function returns *DtTYPE_NOT_DEFINED* if there was no previous data type mapping for *id* in the table. Otherwise, it returns the old value. If the optional *warn* argument is non-zero, then a warning message is printed when an existing mapping is being overridden.

The following classes in VR-Link are derived from *DtPduWithData*:

- ♦ *DtActionRequestPdu*
- ♦ *DtActionResponsePdu*
- ♦ *DtDataPdu*
- ♦ *DtEventReportPdu*
- ♦ *DtMessagePdu*
- ♦ *DtSetDataPdu*.

See Also

[*DtSimanPdu*](#), this PDU's header file ([*pduWithData.h*](#)), and the description of *DtPdu* in Chapter 7, "The DIS-Specific Interface" in the *VR-Link Developer's Guide*.

DtReceiverPdu

Name

DtReceiverPdu (DIS Receiver PDU)

Description

Inspecting and mutating PDU header data is done through *DtPdu* member functions.

Constructors

Standard constructors derived from *DtPdu*:

- ♦ Blank PDU constructor
- ♦ From-network-representation constructor.

Functions

Function	Description
entityId() radioid() receiverState() receivedPower() transmitEntityId() transmitRadioid()	Returns the value of the appropriate data.
setEntityId() setRadioid() setReceiverState() setReceivedPower() setTransmitEntityId() setTransmitRadioid()	Sets the appropriate PDU data to the values passed as arguments.

See Also

This PDU's header file ([receivePdu.h](#)) and the description of *DtPdu* in Chapter 7, "The DIS-Specific Interface" in the *VR-Link Developer's Guide*.

DtRemoveEntityPdu

Name

DtRemoveEntityPdu (DIS Remove Entity PDU)

Description

Inspecting and mutating PDU header data is done through *DtPdu* member functions.

Constructors

Standard constructors derived from *DtPdu*:

- ♦ Blank PDU constructor
- ♦ From-network-representation constructor.

Class-specific constructors:

- ♦ The constructor creates a blank *DtRemoveEntityPdu* using **buffer**, then sets the originating entity ID to **from**, and the receiving entity ID to **to**.

Functions

Function	Description
senderId() receiverId() setSenderId() setReceiverId()	Inherited from DtSimanPdu .
requestId() setRequestId()	Inspect and set this field.

i

requestId() and setRequestId() are unavailable when linking with *libv13*.

See Also

[DtSimanPdu](#), this PDU's header file ([receivePdu.h](#)), and the description of *DtPdu* in Chapter 7, "The DIS-Specific Interface" in the *VR-Link Developer's Guide*.

DtRepairCompletePdu

Name

DtRepairCompletePdu (DIS RepairComplete PDU)

Description

Inspecting and mutating PDU header data is done through *DtPdu* member functions.

Constructors

Standard constructors derived from *DtPdu*:

- ♦ Blank PDU constructor
- ♦ From-network-representation constructor.

Functions

Function	Description
receivingId() repairingId() repairType()	Returns the value of the appropriate data.
setReceivingId() setRepairingId() setRepairType()	Sets the value of the appropriate data.

See Also

This PDU's header file ([repairPdu.h](#)) and the description of *DtPdu* in Chapter 7, "The DIS-Specific Interface" in the *VR-Link Developer's Guide*.

DtRepairResponsePdu

Name

DtRepairResponsePdu (DIS RepairResponse PDU)

Description

Inspecting and mutating PDU header data is done through *DtPdu* member functions.

Constructors

Standard constructors derived from *DtPdu*:

- ♦ Blank PDU constructor
- ♦ From-network-representation constructor.

Functions

Function	Description
receivingId() repairingId() repairResult()	Returns the value of the appropriate data.
setReceivingId() setRepairingId() setRepairResult()	Sets the value of the appropriate data.

See Also

This PDU's header file ([repairPdu.h](#)) and the description of *DtPdu* in Chapter 7, "The DIS-Specific Interface" in the *VR-Link Developer's Guide*.

DtResupplyCancelPdu

Name

DtResupplyCancelPdu (DIS ResupplyCancel PDU)

Description

Inspecting and mutating PDU header data is done through *DtPdu* member functions.

Constructors

Standard constructors derived from *DtPdu*:

- ♦ Blank PDU constructor
- ♦ From-network-representation constructor.

Class-specific constructors:

- ♦ The constructor constructs a blank Resupply Cancel PDU, then sets the receiving and supplying entity IDs to the IDs passed as arguments.

Functions

Function	Description
receivingId() supplyingId()	Returns the value of the appropriate data.
setReceivingId() setSupplyingId()	Sets the value of the appropriate data.

See Also

This PDU's header file ([resupplyPdu.h](#)) and the description of *DtPdu* in Chapter 7, "The DIS-Specific Interface" in the *VR-Link Developer's Guide*.

DtResupplyOfferPdu

Name

DtResupplyOfferPdu (DIS ResupplyOffer PDU)

Description

Inspecting and mutating PDU header data is done through *DtPdu* member functions.

Constructors

Standard constructors derived from *DtPdu*:

- ♦ Blank PDU constructor
- ♦ From-network-representation constructor.

Class-specific constructors:

- ♦ The constructor may be used when the PDU needs to convey information about just one supply type. It constructs a Resupply Offer PDU with receiving and supplying entity IDs set to **receiver** and **supplier** respectively. The PDU will have one supply type record, with type **supply** and quantity **quantity**.

Functions

Function	Description
receivingId() supplyingId() numSupplyTypes()	Returns the value of the appropriate data.
setReceivingId() setSupplyingId()	Sets the value of the appropriate data.
nthSupply()	Returns the entity type in the <i>n</i> th supply type record, where <i>n</i> is supplyIndex . Records are indexed from 0 to numSupplyTypes() - 1. If quantity is a non-NULL pointer, the integer to which it points is set to the quantity in which the <i>n</i> th supply type is being offered. If supplyIndex is not a valid index for the PDU, a <i>DtEntityType</i> with all fields set to zero is returned, and the integer pointed to by retCode (if retCode is non-NULL) is set to DtSUPPLY_BAD_INDEX. If the index is valid, the integer pointed to by retCode is set to DtSUPPLY_VALID.
supplyQuantity()	Returns the quantity in which the supply type supply is being offered. If there is no supply type record whose supply type is supply , zero is returned.

Function	Description
setNumSupplyTypes()	Resizes the PDU to accommodate numTypes supply type records. If numTypes is greater than the previous number of supply type records, trailing records are deleted. Otherwise, supply type records with types and quantities consisting of all zeros are added to the end of the PDU.
setNthSupply()	Sets the supply type and quantity that appears in the <i>n</i> th supply type record, where <i>n</i> is supplyIndex . Records are indexed from 0 to numSupplyTypes() - 1. The integer pointed to by retCode (if retCode is non-NULL) is set to DtSUPPLY_BAD_INDEX if supplyIndex is not in this range, and DtSUPPLY_VALID if it is. In the former case, the PDU remains unchanged.

See Also

This PDU's header file ([resupplyPdu.h](#)) and the description of *DtPdu* in Chapter 7, "The DIS-Specific Interface" in the *VR-Link Developer's Guide*.

DtResupplyReceivedPdu

Name

DtResupplyReceivedPdu (DIS ResupplyReceived PDU)

Description

Inspecting and mutating PDU header data is done through *DtPdu* member functions.

Constructors

Standard constructors derived from *DtPdu*:

- ♦ Blank PDU constructor
- ♦ From-network-representation constructor.

Class-specific constructors:

- ♦ The constructor may be used when the PDU needs to convey information about just one supply type. It constructs a Resupply Received PDU with receiving and supplying entity IDs set to **receiver** and **supplier** respectively. The PDU has one supply type record, with type **supply** and quantity **quantity**.

Functions

Function	Description
receivingId() supplyingId() numSupplyTypes()	Returns the value of the appropriate data.
setReceivingId() setSupplyingId()	Sets the value of the appropriate data.
nthSupply()	Returns the entity type in the <i>n</i> th supply type record, where <i>n</i> is supplyIndex . Records are indexed from 0 to numSupplyTypes() - 1. If quantity is a non-NULL pointer, the integer to which it points is set to the quantity in which the <i>n</i> th supply type is being offered. If supplyIndex is not a valid index for the PDU, a <i>DtEntityType</i> with all fields set to zero is returned, and the integer pointed to by retCode (if retCode is non-NULL) is set to DtSUPPLY_BAD_INDEX. If the index is valid, the integer pointed to by retCode is set to DtSUPPLY_VALID.
supplyQuantity()	Returns the quantity in which the supply type supply is being offered. If there is no supply type record whose supply type is supply , zero is returned.

Function	Description
setNumSupplyTypes()	Resizes the PDU to accommodate numTypes supply type records. If numTypes is greater than the previous number of supply type records, trailing records are deleted. Otherwise, supply type records with types and quantities consisting of all zeros are added to the end of the PDU.
setNthSupply()	Sets the supply type and quantity that appears in the <i>n</i> th supply type record, where <i>n</i> is supplyIndex . Records are indexed from 0 to <code>numSupplyTypes() - 1</code> . The integer pointed to by retCode (if retCode is non-NULL) is set to <code>DtSUPPLY_BAD_INDEX</code> if supplyIndex is not in this range, and <code>DtSUPPLY_VALID</code> if it is. In the former case, the PDU remains unchanged.

See Also

This PDU's header file ([resupplyPdu.h](#)) and the description of *DtPdu* in Chapter 7, "The DIS-Specific Interface" in the *VR-Link Developer's Guide*.

DtSetDataPdu

Name

DtSetDataPdu (DIS SetData PDU)

Description

Inspecting and mutating PDU header data is done through *DtPdu* member functions.

Constructors

Standard constructors derived from *DtPdu*:

- ♦ Blank PDU constructor
- ♦ From-network-representation constructor.

Functions

Function	Description
senderId() receiverId() setSenderId() setReceiverId()	Inherited from DtSimanPdu .
numFixedFields() numVarFields() setNumFixedFields() setNumVarFields() setDatumParam() setVarDataBytes() initDataSection() datumSizeBits() datumSizeBytes() datumId() getIndex() fixedDatumId() varDatumId() datumVal() functions setDatumVal() functions	Inherited from DtPduWithData .
requestId() setRequestId()	Inspect and set this field.

See Also

[DtPduWithData](#), [DtSimanPdu](#), this PDU's header file ([setDataPdu.h](#)), and the description of *DtPdu* in Chapter 7, "The DIS-Specific Interface" in the *VR-Link Developer's Guide*.

DtSignalPdu

Name

DtSignalPdu (DIS Signal PDU)

Description

Inspecting and mutating PDU header data is done through *DtPdu* member functions.

Constructors

Standard constructors derived from *DtPdu*:

- ♦ Blank PDU constructor
- ♦ From-network-representation constructor.

Functions

Function	Description
entityId() radioId() sampleRate() numSamples()	Returns the value of the appropriate data.
encodingClass()	Returns the value of the upper 2 bits of the encoding scheme, one of the following: ♦ <i>DtSignalClassEncodedVoice</i> ♦ <i>DtSignalClassRawBinary</i> ♦ <i>DtSignalClassApplicationSpecific</i> ♦ <i>DtSignalClassPreRecorded</i>
encodingType()	Returns the value in the lower 14 bits of the encoding scheme.
tIdType() setTIdType()	Inspect and set the TDL type respectively.
numDataBytes()	Returns the size of the data section in bytes, while numDataBits() returns the size in bits.
dataByte()	Returns the <i>n</i> th byte of the data section of the PDU, where <i>n</i> is byteNum . Data byte indices range from 0 to numDataBytes().
getData()	is used to copy the entire content of the data section of the PDU into the buffer buff . buff must be large enough to accommodate all of the data, that is, it should be at least numDataBytes() long.

Function	Description
setEntityId() setRadioId() setSampleRate() setNumSamples()	Sets the appropriate PDU data to the values passed as arguments.
setEncodingClass()	Sets the value of the top two bits of the encoding scheme to one of the following: ♦ <i>DtSignalClassEncodedVoice</i> ♦ <i>DtSignalClassRawBinary</i> ♦ <i>DtSignalClassApplicationSpecific</i> ♦ <i>DtSignalClassPreRecorded</i>.
setEncodingType()	Sets the value in the lower 14 bits of the encoding scheme to codeType .
setNumDataBits() setNumDataBytes()	Resizes the PDU to accommodate the new size of the data section of the PDU. The PDU length and data length fields of the PDU are updated accordingly. If the new length is larger than the current length of the PDU, all new data bytes created are set to 0. If the new length is less than the old length, trailing data bytes are deleted.
setData()	will copy numBytes bytes starting from data into the data section of the PDU, effectively enabling setting the whole data section at one.
setDataByte()	may be used to set bytes in the data section one at a time. Bytes are indexed from 0 to numDataBytes() - 1.

i

tdlType() and setTdlType() are not available when linking with *libv3*.

See Also

This PDU's header file ([signalPdu.h](#)) and the description of *DtPdu* in Chapter 7, "The DIS-Specific Interface" in the *VR-Link Developer's Guide*.

DtSimanPdu

Name

DtSimanPdu (DIS Simulation Management PDU)

Description

DtSimanPdu is an abstract class that encapsulates the functionality common to all simulation management PDUs. Specifically, *DtSimanPdu* provides access to the sender and receiver IDs in instances of PDU classes that are derived from *DtSimanPdu*.

DtSimanPdu is derived from *DtPdu*, so that when an instance of a class derived from *DtSimanPdu* is constructed, inspecting and mutating PDU header data is done through *DtPdu* member functions. Since *DtSimanPdu* is an abstract class, an instance of *DtSimanPdu* cannot be directly constructed.

Constructors

Standard constructors derived from *DtPdu*:

- ♦ Blank PDU constructor
- ♦ From-network-representation constructor.

Functions

Function	Description
senderId() and receiverId()	Returns the value of the appropriate data.
setSenderId() and setReceiverId()	set the appropriate PDU data to the values passed as arguments.

Notes

The following classes in VR-Link are derived directly from *DtSimanPdu*:

- ♦ *DtAcknowledgePdu*
- ♦ *DtCreateEntityPdu*
- ♦ *DtDataQueryPdu*
- ♦ *DtRemoveEntityPdu*
- ♦ *DtStartResumePdu*
- ♦ *DtStopFreezePdu*.

In addition, the following classes are derived from *DtPduWithData*, which is in turn derived from *DtSimanPdu*:

- ♦ *DtActionRequestPdu*
- ♦ *DtActionResponsePdu*
- ♦ *DtDataPdu*
- ♦ *DtEventReportPdu*
- ♦ *DtMessagePdu*
- ♦ *DtSetDataPdu*.

See Also

This PDU's header file ([simanPdu.h](#)) and the description of *DtPdu* in Chapter 7, "The DIS-Specific Interface" in the *VR-Link Developer's Guide*.

DtStartResumePdu

Name

DtStartResumePdu (DIS Start/Resume PDU)

Description

Inspecting and mutating PDU header data is done through *DtPdu* member functions.

Constructors

Standard constructors derived from *DtPdu*:

- ♦ Blank PDU constructor
- ♦ From-network-representation constructor.

Class-specific constructors:

The constructor creates a blank *DtStartResumePdu* using *buffer*, then sets the originating entity ID to *from*, the receiving entity ID to *to*, the value of the real world time to *realTime*, and the value of the simulation time to *simTime*.

Functions

Function	Description
senderId() receiverId() setSenderId() setReceiverId()	Inherited from <i>DtSimanPdu</i> .
realWorldTime()	Returns the value of the real world time encoded in the PDU. The return value is a floating point number representing seconds since January 1, 1970.
simulationTime()	Returns the value of the simulation time encoded in the PDU. The return value is a floating point number representing seconds since January 1, 1970.
setRealWorldTime(setSimulationTime()	Sets the values of those data. <i>Time</i> is a floating point number representing seconds since January 1, 1970.
requestId() setRequestId()	Inspect and set this field.



requestId() and setRequestId() are unavailable when linking with *libv13*.

See Also

[DtSimanPdu](#), this PDU's header file ([startPdu.h](#)), and the description of *DtPdu* in Chapter 7, "The DIS-Specific Interface" in the *VR-Link Developer's Guide*.

DtStopFreezePdu

Name

DtStopFreezePdu (DIS Stop/Freeze PDU)

Description

Inspecting and mutating PDU header data is done through *DtPdu* member functions.

Constructors

Standard constructors derived from *DtPdu*:

- ♦ Blank PDU constructor
- ♦ From-network-representation constructor.

Class-specific constructors:

- ♦ The constructor creates a blank *DtStopFreezePdu* using **buffer**, then sets the originating entity ID to **from**, the receiving entity ID to **to**, the value of the real world time to **realTime**, and the stop reason to **reason**.

Functions

Function	Description
senderId() receiverId() setSenderId() setReceiverId()	Inherited from DtSimanPdu .
realWorldTime()	Returns the value of the real world time encoded in the PDU. The return value is a floating point number representing seconds since January 1, 1970.
setRealWorldTime()	Sets the real world time to Time , which is a floating point number representing seconds since January 1, 1970.
frozenBehavior() setFrozenBehavior() requestId() setRequestId()	Inspect and set the appropriate fields.
reason()	Returns the reason for stopping
setReason()	Sets the reason for stopping.



These functions are unavailable when linking with *libvl3*.

See Also

[DtSimanPdu](#), this PDU's header file ([stopPdu.h](#)), and the description of *DtPdu* in Chapter 7, "The DIS-Specific Interface" in the *VR-Link Developer's Guide*.

DtTransmitterPdu

Name

DtTransmitterPdu (DIS Transmitter PDU)

Description

Inspecting and mutating PDU header data is done through *DtPdu* member functions.

Constructors

Standard constructors derived from *DtPdu*:

- ♦ Blank PDU constructor
- ♦ From-network-representation constructor.

Functions

Function	Description
entityId() radioId() radioEntityType() transmitState() inputSource() worldAntennaLoc() relativeAntennaLoc() antennaPatternType() antennaParamsLength() frequency() bandwidth() power() cryptoSystem() cryptoKeyId() modulationParamsLength()	Returns the value of the appropriate data.
radioEntityType()	Returns a <i>DtEntityType</i> when using DIS version 2.0.3, (linked with <i>libvl3.a</i>), and a <i>DtRadioEntityType</i> when using DIS version 4 or 5 (linked with <i>libvl.a</i>).
antennaParamsLength() and modulationParamsLength()	Returns values in bytes. As 64 bit integers are not supported on most platforms frequency has an optional argument to indicate whether one wishes to inspect the upper or lower 32 bit word of the 64 bit frequency field. The upperLower argument defaults to DtFREQ_LOWER, but may also be set to DtFREQ_UPPER.

Function	Description
modulationType()	Returns a reference to a <i>DtModulationType</i> object. The <i>DtModulationType</i> member functions spectrum(), majorModType(), detail(), and systemType() may be used to obtain individual pieces of data from this object. For more information see the text following this table.
setEntityId() setRadioID() setRadioEntityType() setTransmitState() setInputSource() setWorldAntennaLoc() setRelativeAntennaLoc() setAntennaPatternType() setFrequency() setBandwidth() setPower() setModulationType() setCryptoSystem() setCryptoKey()	Sets the appropriate PDU data to the values passed as arguments. See text for further discussion

modulationType() *Example*

```
int detail = transPdu.modulationType().detail();
```

The layout for the modulation parameters portion of the Transmitter PDU depends on the modulation type and is not described in the DIS standard. Therefore, VR-Link gives the user access to the bitwise network representation of this portion of the PDU. The function getNetModulationParams() copies this data into the buffer provided as the **destination** argument to the function. This buffer must be at least modulationParamsLength() long.

Similarly, one can obtain the bitwise network representation of the antenna pattern parameters, using getNetAntennaParams(). The size of the buffer passed as **destination** must be at least antennaParamsLength() bytes long.

If the antenna pattern type for a particular transmitter PDU is a beam antenna pattern (that is, antennaPatternType() returns *DtBeam*), the function antennaParams() can be used to interpret the antenna pattern parameters as a beam antenna pattern record. A reference to a *DtBeamAntennaPatternRecord* is returned. The *DtBeamAntennaPatternRecord* member functions can be used to inspect the values of its individual fields. They are:

- ♦ angles()
- ♦ azimuthBeamwidth()
- ♦ elevationBeamwidth()
- ♦ referenceSystem()
- ♦ electricFieldZ()

- ♦ electricFieldX()
- ♦ phase().

For example:

```
float phase = pdu.antennaParams().phase();
```

setRadioEntityType()

When linking with *libvl.a* (using DIS 2.0.4 or 1278.1-1995), *setRadioEntityType()* is overloaded to take either a *DtRadioEntityType* or a *DtEntityType*. However, use *DtRadioEntityType* in version 1278.1-1995 PDUs; use *DtEntityType* in DIS 2.0.4 PDUs.



When linking with *libvl3.a*, (using DIS 2.0.3), only the version of *setRadioEntityType()* that takes a *DtEntityType* is available.

setFrequency()

There are two versions of *setFrequency()*. One allows you to set both the upper and lower words of the 64-bit frequency field. The other may be used when the frequency will fit completely in the lower word.

setModulationType()

The **type** argument to *setModulationType()* is reference to a *DtModulationType* object. The *DtModulationType* member functions *setSpectrum()*, *setMajor()*, *setDetail()*, and *setSystemType()* may be used to set the individual components of the object. Alternately, all of these data may be passed to the *DtModulationType* constructor. For example:

```
pdu.setModulationType(DtModulationType(spec, maj, det,
                                     sys));
```

DIS does not specify the layout of the modulation parameters record. The layout, in fact, depends on the modulation type. VR-Link allows you to set the exact network representation for the PDU's modulation parameter information using *setModulationParams()*. Typically, this would be used by creating an appropriate struct, filling out all of the fields, and passing it to this function. The **size** argument indicates the length, in bytes, of the modulation parameters.

setAntennaPatternParams()

Similarly, the function *setAntennaPatternParams()* may be used to set the network representation of the antenna pattern parameters. **size** indicates the length, in bytes, of the antenna pattern parameters.

If the antenna pattern type is a beam antenna pattern, the antenna pattern parameters may be represented as a *DtBeamAntennaPatternRecord*. An overloaded version of `setAntennaPatternParams()` takes a reference to an object of this type, and sets the PDU's antenna pattern parameter record appropriately. The *DtBeamAntennaPatternRecord* member functions can be used to set the individual fields of this object. They are:

- ♦ `setAngles()`
- ♦ `setBeamwidth()`
- ♦ `setReferenceSystem()`
- ♦ `setElectricField()`
- ♦ `setPhase()`.

See Also

This PDU's header file ([transmitPdu.h](#)) and the description of *DtPdu* in Chapter 7, "The DIS-Specific Interface" in the *VR-Link Developer's Guide*.

DtViewControlPdu

Name

DtViewControlPdu

Description

A *DtViewControlPdu* represents an instance of MÄK's ViewControl PDU. Inspecting and mutating PDU header data is done through *DtPdu* member functions.

When sent to the MÄK Stealth, View Control PDUs cause changes in the Stealth's View State.

There are currently 10 different types of View Control PDU that the MÄK Stealth accepts - one each for nine of its View Modes and one to indicate a change of attached entity or view group without setting the current mode. Each View Mode requires different parameters to describe the View State. A subclass of *DtViewControlPdu* represents each type of View Control PDU:

- ♦ *DtAbsoluteVCPdu*
- ♦ *DtTetherVCPdu*
- ♦ *DtCompassVCPdu*
- ♦ *DtMimicVCPdu*
- ♦ *DtOrbitVCPdu*
- ♦ *DtTrackVCPdu*
- ♦ *DtTetherTrackVCPdu*
- ♦ *DtMimicTrackVCPdu*
- ♦ *DtGroupVCPdu*
- ♦ *DtModelessVCPdu*.

In addition, the derived class *DtUnknownVCPdu* may be used to represent new experimental or unknown kinds of View Control PDU.

Data that is common to all types of View Control PDUs may be accessed through the base *DtViewControlPdu* mutators and inspectors.

All View Control PDUs have the same DIS PDU kind (*DtViewControlPduKind* = 202), but each type has its own Viewer Command. *viewerCommand()* returns the PDU's Viewer Command. (See [disEnums.h](#) for the *DtViewerCommand* enumeration.)

The Viewer ID in a View Control PDU is the entity ID of the Stealth to which the PDU is directed. A value of 0xFFFF may be used as a wildcard in any field of the entity ID. *viewerId()* and *setViewerId()* access the PDU's Viewer Id.

Constructors

Standard constructors derived from *DtPdu*:

- ♦ Blank PDU constructor
- ♦ From-network-representation constructor

Functions

Function	Description
senderId() and setSenderId()	Inspect and set the entity ID of the sender of PDU.
ignoreFlags() and setIgnoreFlags()	Inspect and set ignore flags.

All View Control PDUs have an Ignore Flags field that is used to indicate to the Stealth that certain viewing parameters should remain as they were, and should not be set from this PDU. The different Ignore commands that may be OR'ed into the Ignore bitfield are:

- ♦ DtIGNORE_EAST
- ♦ DtIGNORE_PITCH_OFFSET
- ♦ DtIGNORE_NORTH
- ♦ DtIGNORE_FORWARD
- ♦ DtIGNORE_HEIGHT
- ♦ DtIGNORE_RIGHT
- ♦ DtIGNORE_HEADING
- ♦ DtIGNORE_UP
- ♦ DtIGNORE_PITCH
- ♦ DtIGNORE_BODY_PITCH
- ♦ DtIGNORE_ROLL
- ♦ DtIGNORE_BODY_ROLL
- ♦ DtIGNORE_EAST_OFFSET
- ♦ DtIGNORE_AZIMUTH_RATE
- ♦ DtIGNORE_NORTH_OFFSET
- ♦ DtIGNORE_RADIAL_OFFSET
- ♦ DtIGNORE_HEIGHT_OFFSET
- ♦ DtIGNORE_PRIMARY_ID
- ♦ DtIGNORE_RADIUS
- ♦ DtIGNORE_SECONDARY_ID
- ♦ DtIGNORE_BODY_HEADING

- ♦ DtIGNORE_GROUP_IDS
- ♦ DtIGNORE_HEADING_OFFSET.

The values of the parameters in a view Control PDU can be represented with an ASCII view command string. These strings are used in *viewsave.mtl* - the file written by the Stealth whenever the saved view states are to be written, and also in Stealth Scripting files, which record a sequence of View State changes. **string** returns a string representing the data in the PDU, and **setFromString()** sets the data in the PDU to the values in the string.

DtNewViewControlPdu creates an instance of the appropriate derived *DtViewControlPdu* based on the value of **command**.

Each derived *DtViewControlPdu* has individual accessor functions to inspect and mutate the parameters associated with its View Mode.

See Also

This PDU's header file ([vcPdu.h](#)) and the description of *DtPdu* in Chapter 7, "The DIS-Specific Interface" in the *VR-Link Developer's Guide*.



Header File Reference

This chapter contains abbreviated versions of the VR-Link header files.

This appendix appears only in the on-line PDF version of the use Acrobat's *search* and *bookmark* features to find information quickly. In addition, the cross-reference tables on pages [1-4](#) through [1-41](#) contain hypertext links that allow easy navigation of all header files and classes in this appendix.

absNotifyObs.h

```
#include "MachineTypes.h"
#include "observee.h"

// class DtAbsNotifyObserver:
//
// DtAbsNotifyObserver is an abstract base class.
// Instances of DtAbsNotifyObserver are used by DtObservee to
// notify an observer object that something has happened to the
// observed object.  Observers should subclass from DtAbsNotifyOb-
// server
// to receive notification for their class.

class DT_DLL_VLUTIL DtAbsNotifyObserver
{
public:

    // destructor
    virtual ~DtAbsNotifyObserver();

    // assignment operator
    DtAbsNotifyObserver& operator=(const DtAbsNotifyObserver& orig);

    // Called by DtObservee when something has happened to the
    // observed object.
    // Default does nothing.  This should be overridden.
    virtual DtBoolean notifyFromObserved(DtObservee* observed,
        const DtObservee::DtWhyNotifyObserver& why);

protected:

    // Constructors are protected since this is an abstract base
    class.

    // default constructor
    DtAbsNotifyObserver();

    // copy constructor
    DtAbsNotifyObserver(const DtAbsNotifyObserver& orig);
};
```

ackPdu.h

```

#if !DtHLA

#include "hostStructs.h"
#include "simanPdu.h"

#define DtAcknowledgeStaticSize 32

class DT_DLL_VL DtAcknowledgePdu;
typedef DtAcknowledgePdu DtAcknowledgeInteraction;
typedef void (*DtAcknowledgeInteractionCb)
    (DtAcknowledgeInteraction *pdu, void *usr);

class DT_DLL_VL DtAcknowledgePdu : public DtSimanPdu
{
public:

/***** STANDARD CONSTRUCTORS *****/

    DtAcknowledgePdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtAcknowledgePdu(const DtNetAcknowledgePdu *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    static DtPdu *create(const DtNetPacket *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    static void addCallback(DtExerciseConn *conn,
        DtAcknowledgeInteractionCb cb, void *usr);
    static void removeCallback(DtExerciseConn *conn,
        DtAcknowledgeInteractionCb cb, void *usr);

/***** ADDITIONAL CONSTRUCTORS *****/

    DtAcknowledgePdu(const DtEntityIdentifier &to,
        const DtEntityIdentifier &from,
        DtAcknowledgeFlag aFlag,
        DtResponseFlag rFlag,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

/***** INSPECTOR FUNCTIONS *****/

    DtAcknowledgeFlag acknowledgeFlag() const;
    DtResponseFlag responseFlag() const;
#if DtDISVERS != 3
    int requestId() const;
#endif

/***** MUTATOR FUNCTIONS *****/

```

```
    void setAcknowledgeFlag(DtAcknowledgeFlag flag);
    void setResponseFlag(DtResponseFlag flag);
#if DtDISVERS != 3
    void setRequestId(int id);
#endif

    void printData() const;

protected:
    DtPduKind internalGetPduKind() const {return DtAcknowledgePdu-
Kind;}
};
#endif
```


ackRPdu.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "hostStructs.h"
#include "simanPdu.h"
#include "packets/ackRP.h"
#include "exerciseConn.h"

// DtAcknowledgeRPdu callback function signature.
typedef void (*DtAcknowledgeRPduCb)(DtAcknowledgeRPdu* pdu, void*
usr);

// class DtAcknowledgeRPdu:
//
// Instances of DtAcknowledgeRPdu are used to send reliable acknowl-
// edge
// PDUs.

class DT_DLL_VL DtAcknowledgeRPdu : public DtSimanPdu
{
public:

    // Constructors
    DtAcknowledgeRPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtAcknowledgeRPdu(const DtNetAcknowledgeRPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtAcknowledgeRPdu(const DtAcknowledgeRPdu& orig);

    // destructor
    virtual ~DtAcknowledgeRPdu();

    // Assignment operator
    DtAcknowledgeRPdu& operator=(const DtAcknowledgeRPdu& orig);

    // Set/Get the acknowledge flag.
    virtual void setAcknowledgeFlag(DtAcknowledgeFlag flag);
    virtual DtAcknowledgeFlag acknowledgeFlag() const;

    // Set/Get the response flag.
    virtual void setResponseFlag(DtResponseFlag flag);
    virtual DtResponseFlag responseFlag() const;

    // Set/Get the request id.
    virtual void setRequestId(int id);
    virtual int requestId() const;

    // Print the data contained by this PDU

```

```
    virtual void printData() const;

public:
    // Create a DtAcknowledgeRPdu.
    // Caller is responsible for deletion.
    static DtPdu* create(const DtNetPacket* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // Add (register)/Remove (deregister) function to be called when
    // pdu is received.
    static void addCallback(DtExerciseConn* conn,
        DtAcknowledgeRPduCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtAcknowledgeRPduCb cb, void* usr);

protected:
    // Returns the internal PDU kind
    virtual DtPduKind internalGetPduKind() const;

protected:
    // Size of PDU with zero data bytes.
    static int theStaticSize;

};
#endif
#endif
```

actionPdu.h

```

#if !DtHLA

#include "hostStructs.h"
#include "pduWithData.h"
#include "datumIdType.h"

/*****
/***** ACTION REQUEST PDU *****/
/*****/

#if DtDISVERS != 3
#define DtActionRequestStaticSize 40
#else
#define DtActionRequestStaticSize 48
#endif

typedef DtActionRequestPdu DtActionRequestInteraction;
typedef void (*DtActionRequestInteractionCb)
    (DtActionRequestInteraction *pdu, void *usr);

class DT_DLL_VL DtActionRequestPdu : public DtPduWithData
{
public:

/***** STANDARD CONSTRUCTORS *****/

    DtActionRequestPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtActionRequestPdu(const DtNetActionRequestPdu *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    static DtPdu *create(const DtNetPacket *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    static void addCallback(DtExerciseConn *conn,
        DtActionRequestInteractionCb cb, void *usr);
    static void removeCallback(DtExerciseConn *conn,
        DtActionRequestInteractionCb cb, void *usr);

/***** INSPECTOR FUNCTIONS *****/

    unsigned long requestId() const;
    unsigned long actionId() const;

/***** MUTATOR FUNCTIONS *****/

    void setRequestId(unsigned long id);
    void setActionId(unsigned long id);

```

```
void printData() const;

protected:

    DtPduKind internalGetPduKind() const {return DtActionRequestPdu-
Kind;}
    DtNetDataPduTail *dataSection() const;
};
/*****
/***** ACTION RESPONSE PDU *****/
/*****/

#if DtDISVERS != 3
#define DtActionResponseStaticSize 40
#else
#define DtActionResponseStaticSize 48
#endif

typedef DtActionResponsePdu DtActionResponseInteraction;
typedef void (*DtActionResponseInteractionCb)(DtActionResponsePdu
*pdu, void *usr);

class DT_DLL_VL DtActionResponsePdu : public DtPduWithData
{
public:

/***** STANDARD CONSTRUCTORS *****/

    DtActionResponsePdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtActionResponsePdu(const DtNetActionResponsePdu *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    static DtPdu *create(const DtNetPacket *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    static void addCallback(DtExerciseConn *conn,
        DtActionResponseInteractionCb cb, void *usr);
    static void removeCallback(DtExerciseConn *conn,
        DtActionResponseInteractionCb cb, void *usr);

/***** INSPECTOR FUNCTIONS *****/

    unsigned long requestId() const;
    DtRequestStatus requestStatus() const;

/***** MUTATOR FUNCTIONS *****/

    void setRequestId(unsigned long id);
    void setRequestStatus(DtRequestStatus rStatus);

    void printData() const;
```

```
protected:

    DtPduKind internalGetPduKind() const {return DtActionResponsePdu-
Kind;}
    DtNetDataPduTail *dataSection() const;
};

#endif
```

actReqRPdu.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "hostStructs.h"
#include "pduWithData.h"
#include "datumIdType.h"
#include "packets/actReqRP.h"
#include "exerciseConn.h"

// class DtActionRequestRPdu:
//
// Instances of DtActionRequestRPdu are used to send/receive
// reliable action request PDUs.

// DtActionRequestRPdu callback function signature.
typedef void (*DtActionRequestRPduCb)(DtActionRequestRPdu* pdu,
void* usr);

class DT_DLL_VL DtActionRequestRPdu : public DtPduWithData
{
public:

    // Constructors
    DtActionRequestRPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtActionRequestRPdu(const DtNetActionRequestRPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtActionRequestRPdu(const DtActionRequestRPdu& orig);

    // destructor
    virtual ~DtActionRequestRPdu();

    // Assignment operator
    DtActionRequestRPdu& operator=(const DtActionRequestRPdu& orig);

    // Set/Get the request id.
    virtual void setRequestId(unsigned long id);
    virtual unsigned long requestId() const;

    // Set/Get the action id.
    virtual void setActionId(unsigned long id);
    virtual unsigned long actionId() const;

    // Set/Get the required reliability service.
    virtual void setReqRelService(DtRequiredReliabilityService
value);
    virtual DtRequiredReliabilityService reqRelService() const;
```

```
// Print the data contained by this PDU
virtual void printData() const;

public:

    // Create a DtActionRequestRPdu.
    // Caller is responsible for deletion.
    static DtPdu* create(const DtNetPacket* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // Add (register)/Remove (deregister) function to be called when
    // pdu is received.
    static void addCallback(DtExerciseConn* conn,
        DtActionRequestRPduCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtActionRequestRPduCb cb, void* usr);

protected:

    // Returns the internal PDU kind
    virtual DtPduKind internalGetPduKind() const;

    // Returns the data portion of the PDU
    virtual DtNetDataPduTail* dataSection() const;

protected:

    static int theStaticSize;

};
#endif
#endif
```

actRespRPdu.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "hostStructs.h"
#include "pduWithData.h"
#include "datumIdType.h"
#include "packets/actRespRP.h"
#include "exerciseConn.h"

// class DtActionResponseRPdu:
//
// Instances of DtActionResponseRPdu are used to send/receive
// reliable action response PDUs.

// DtActionResponseRPdu callback function signature.
typedef void (*DtActionResponseRPduCb)(DtActionResponseRPdu* pdu,
void* usr);

class DT_DLL_VL DtActionResponseRPdu : public DtPduWithData
{
public:

    // Constructors
    DtActionResponseRPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtActionResponseRPdu(const DtNetActionResponseRPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtActionResponseRPdu(const DtActionResponseRPdu& orig);

    // destructor
    virtual ~DtActionResponseRPdu();

    // Assignment operator
    DtActionResponseRPdu& operator=(const DtActionResponseRPdu&
orig);

    // Set/Get the request id.
    virtual void setRequestId(unsigned long id);
    virtual unsigned long requestId() const;

    // Set/Get the request status.
    virtual void setRequestStatus(DtRequestStatus rStatus);
    virtual DtRequestStatus requestStatus() const;

    // Print the data contained by this PDU
    virtual void printData() const;

public:
```



```
// Create a DtActionResponseRPdu.  
// Caller is responsible for deletion.  
static DtPdu* create(const DtNetPacket* initial,  
    DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);  
  
// Add (register)/Remove (deregister) function to be called when  
// pdu is received.  
static void addCallback(DtExerciseConn* conn,  
    DtActionResponseRPduCb cb, void* usr);  
static void removeCallback(DtExerciseConn* conn,  
    DtActionResponseRPduCb cb, void* usr);  
  
protected:  
  
    // Returns the internal PDU kind  
    virtual DtPduKind internalGetPduKind() const;  
  
    // Returns the data portion of the net structure.  
    virtual DtNetDataPduTail* dataSection() const;  
  
protected:  
  
    static int theStaticSize;  
  
};  
#endif  
#endif
```

aggAggList.h

```
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"
#include "hostStructs.h"

#include "packets/aggStateP.h"

// class DtSilentAggregateList:
//
// DtSilentAggregateList is a class which contains the information
// needed
// to describe a list of silent aggregates for the aggregate state
// PDU.

class DT_DLL_VL DtSilentAggregateList
{
public:

    // Constructors
    DtSilentAggregateList();
    DtSilentAggregateList(const DtSilentAggregateList& orig);

    // Destructor
    virtual ~DtSilentAggregateList();

    // Assignment operator
    DtSilentAggregateList& operator=(const DtSilentAggregateList&
orig);
    DtSilentAggregateList& operator=(const DtNetSilentAggregateList&
orig);

    // Comparison operators
    int operator==(const DtSilentAggregateList& otherguy) const;
    int operator!=(const DtSilentAggregateList& otherguy) const;

    // Set/Get the number of aggregates
    virtual void setNumberOfAggregates(const DtU16 value);
    virtual DtU16 numberOfAggregates() const;

    // Set/Get the aggregate type
    virtual void setAggregateType(const DtEntityType& type);
    virtual const DtEntityType& aggregateType() const;

    // Print the record
    virtual void print() const;
```

```
// Load/unload data from a packet record
virtual void unloadNetRecord(const DtNetSilentAggregateList&
record);
virtual void loadNetRecord(DtNetSilentAggregateList& record)
const;

public:

    // Get the length of the network representation of a record
    static int byteLengthOfNetworkRecord();

protected:

    // Data members:
    DtU16                myNumberOfAggregates;
    DtEntityType          myEntityType;

};
#endif
```

aggEntDec.h

```
#if DtHLA

#include "baseEntDec.h"

// class DtAggregateEntityDecoder:
//
// Instances of DtAggregateEntityDecoder are used to decode an
// aggregate state message into a state repository in an HLA
// (as opposed to DIS) environment.

class DT_DLL_VL DtAggregateEntityDecoder : public DtBaseEntityDe-
coder
{
public:

    // constructor
    DtAggregateEntityDecoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtAggregateEntityDecoder();

    // copy constructor
    DtAggregateEntityDecoder(const DtAggregateEntityDecoder& orig);

    // assignment operator
    DtAggregateEntityDecoder& operator=(const DtAggregateEntityDe-
coder& orig);

public:

    // Create a DtAggregateEntityDecoder.
    // User is responsible for deletion.
    static DtHlaStateDecoder* create(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

protected:

    // Add decoding functions for the attributes.
    // Note: Non-virtual since meant to be called from constructors
    to
    //      add decoders.
    void addMyDecoders();

protected:
```

```

    // static decode<AttributeName> member functions.
    DtDECLARE_AGGREGATE_ATTR_DECODER(AggregateMarking);
    DtDECLARE_AGGREGATE_ATTR_DECODER(AggregateState);
    DtDECLARE_AGGREGATE_ATTR_DECODER(Dimensions);
    DtDECLARE_AGGREGATE_ATTR_DECODER(EntityIDs);
    DtDECLARE_AGGREGATE_ATTR_DECODER(ForceID);
    DtDECLARE_AGGREGATE_ATTR_DECODER(Formation);
    DtDECLARE_AGGREGATE_ATTR_DECODER(NumberOfEntities);
    DtDECLARE_AGGREGATE_ATTR_DECODER(SilentAggregates);
    DtDECLARE_AGGREGATE_ATTR_DECODER(SilentEntities);
    DtDECLARE_AGGREGATE_ATTR_DECODER(SubAggregateIDs);
    DtDECLARE_AGGREGATE_ATTR_DECODER(VariableDatums);
};
// These macros are used to define simple attribute decoder func-
// tions
// (members of a derived DtHlaStateDecoder that decode
// individual attributes). The simple decoding is just to
// convert the value pulled out of the ahvps to a host type using
// the cast expression castExpr, and pass it to the mutator func-
// tion.
#define DtDEFINE_SIMPLE_AGGREGATE_ATTR_DECODER_WITH_CAST( \
    attrName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_ATTR_DECODER_WITH_CAST( \
        DtAggregateEntityDecoder, DtAggregateStateRepository, \
        attrName, netType, mutator, castExpr)

#define DtDEFINE_SIMPLE_AGGREGATE_ATTR_DECODER( \
    attrName, netType, inspector) \
    DtDEFINE_SIMPLE_AGGREGATE_ATTR_DECODER_WITH_CAST( \
        attrName, netType, inspector, (netType))

#endif

```

aggEntEnc.h

```
#if DtHLA

#include "baseEntEnc.h"
#include "aggregateSR.h"

// class DtAggregateEntityEncoder:
//
// Instances of DtAggregateEntityEncoder are used to encode an
// aggregate entity's state repository into a state message in an
// HLA environment.

class DT_DLL_VL DtAggregateEntityEncoder : public DtBaseEntityEncoder
{
public:

    // constructor
    DtAggregateEntityEncoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtAggregateEntityEncoder();

    // copy constructor
    DtAggregateEntityEncoder(const DtAggregateEntityEncoder& orig);

    // assignment operator
    DtAggregateEntityEncoder& operator=(const DtAggregateEntityEncoder& orig);

public:

    // Create a DtAggregateEntityEncoder.
    // Caller is responsible for deletion.
    static DtHlaStateEncoder* create(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

protected:

    // static need<Attribute> member functions.
    DtDECLARE_AGGREGATE_ATTR_CHECKER(AggregateMarking);
    DtDECLARE_AGGREGATE_ATTR_CHECKER(AggregateState);
    DtDECLARE_AGGREGATE_ATTR_CHECKER(Dimensions);
    DtDECLARE_AGGREGATE_ATTR_CHECKER(EntityIDs);
    DtDECLARE_AGGREGATE_ATTR_CHECKER(ForceID);
```

```

DtDECLARE_AGGREGATE_ATTR_CHECKER(Formation);
DtDECLARE_AGGREGATE_ATTR_CHECKER(NumberOfEntities);
DtDECLARE_AGGREGATE_ATTR_CHECKER(SilentAggregates);
DtDECLARE_AGGREGATE_ATTR_CHECKER(SilentEntities);
DtDECLARE_AGGREGATE_ATTR_CHECKER(SubAggregateIDs);
DtDECLARE_AGGREGATE_ATTR_CHECKER(VariableDatums);

// static decode<Attribute> member functions.
DtDECLARE_AGGREGATE_ATTR_ENCODER(AggregateMarking);
DtDECLARE_AGGREGATE_ATTR_ENCODER(AggregateState);
DtDECLARE_AGGREGATE_ATTR_ENCODER(Dimensions);
DtDECLARE_AGGREGATE_ATTR_ENCODER(EntityIDs);
DtDECLARE_AGGREGATE_ATTR_ENCODER(ForceID);
DtDECLARE_AGGREGATE_ATTR_ENCODER(Formation);
DtDECLARE_AGGREGATE_ATTR_ENCODER(NumberOfEntities);
DtDECLARE_AGGREGATE_ATTR_ENCODER(SilentAggregates);
DtDECLARE_AGGREGATE_ATTR_ENCODER(SilentEntities);
DtDECLARE_AGGREGATE_ATTR_ENCODER(SubAggregateIDs);
DtDECLARE_AGGREGATE_ATTR_ENCODER(VariableDatums);
};
// This macro is used to define simple attribute encoder functions
// (members of a derived DtHlaStateEncoder that encode
// individual attributes). The simple encoding is just to implicitly
// convert the value returned by the inspector to a netType,
// and add it to the ahvps.
#define DtDEFINE_SIMPLE_AGG_ENTITY_ATTR_ENCODER( \
    attrName, netType, inspector) \
    DtDEFINE_SIMPLE_ATTR_ENCODER(DtAggregateEntityEncoder, \
        DtAggregateStateRepository, \
        attrName, netType, inspector)

// This macro is used to define simple attribute checker functions
// (members of a derived DtHlaStateEncoder that checker whether
// a particular attribute is out of date. The simple implementation
// is to just compare the values returned by the two StateReps'
// inspector functions.
#define DtDEFINE_SIMPLE_AGG_ENTITY_ATTR_CHECKER( \
    attrName, inspector) \
    DtDEFINE_SIMPLE_ATTR_CHECKER(DtAggregateEntityEncoder, \
        DtAggregateStateRepository, \
        attrName, inspector)

#endif

```

aggEntList.h

```
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"
#include "hostStructs.h"

#include "packets/aggStateP.h"

// class DtSilentEntityList:
//
// DtSilentEntityList is a class which contains the information
// needed
// to describe a list of silent entities for the aggregate state
// PDU.

class DT_DLL_VL DtSilentEntityList
{
public:

    // Constructors
    DtSilentEntityList();
    DtSilentEntityList(const DtSilentEntityList& orig);

    // Destructor
    virtual ~DtSilentEntityList();

    // Assignment operator
    DtSilentEntityList& operator=(const DtSilentEntityList& orig);
    DtSilentEntityList& operator=(const DtNetSilentEntityList& orig);

    // Comparison operators
    int operator==(const DtSilentEntityList& otherguy) const;
    int operator!=(const DtSilentEntityList& otherguy) const;

    // Set/Get the number of aggregates
    virtual void setNumberOfEntities(const DtU16 value);
    virtual DtU16 numberOfEntities() const;

    // Set/Get the entity type
    virtual void setEntityType(const DtEntityType& type);
    virtual const DtEntityType& entityType() const;

#if !DtHLA
    // Set/Get the number of appearance records. Set returns DtFalse
    // if the
```



```

    // value is too large, old values are not retained when the number
    changes.
    virtual DtBoolean setNumberOfAppearanceRecords(const DtU16
value);
    virtual DtU16 numberOfAppearanceRecords() const;

    // Set/Get the appearance of an entity (as a 32 bit integer).
Returns
    // DtFalse if the index is out of range. Set will erase all
previous
    // values.
    virtual DtBoolean setAppearance(int index, DtU32 app);
    virtual DtBoolean getAppearance(int index, DtU32& app) const;

    // Set/Get appearance by attribute. Returns DtFalse if the index
    // is out of range.
    virtual DtBoolean setAppearanceAttribute(int index,
        const DtAppearAttribute attr, const DtAppearAttrVal val);
    virtual DtBoolean getAppearanceAttribute(int index,
        const DtAppearAttribute attr, DtAppearAttrVal& val) const;
#endif

    // Print the record
    virtual void print() const;

    // Load/unload data from a packet record
    virtual void unloadNetRecord(const DtNetSilentEntityList&
record);
    virtual void loadNetRecord(DtNetSilentEntityList& record) const;

    // Get the network byte length of this record
    virtual int networkByteLength() const;

public:

    // Get the length of the network representation of a record
    static int byteLengthOfNetworkRecord(DtNetSilentEntityList*
recPtr);

protected:

    // Data members:
    DtU16                                myNumberOfEntities;
    DtEntityType                        myEntityType;
#ifdef !DtHLA
    DtU16                                myNumberOfAppearanceRecords;
    DtU32*                             myEntityAppearanceList;
#endif
};
#endif

```

aggStatePdu.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "hostStructs.h"
#include "stateMsg.h"
#include "exerciseConn.h"
#include "aggEntList.h"
#include "aggAggList.h"
#include "varDatumRec.h"

#include "mathTypes.h"

#include "packets/aggStateP.h"

// class DtAggregateStatePdu:
//
// Instances of DtAggregateStatePdu are used to communicate informa-
// tion about
// aggregating entities.

// DtAggregateStatePdu callback function signature.
typedef void (*DtAggregateStatePduCb)(DtAggregateStatePdu* pdu,
void* usr);

class DT_DLL_VL DtAggregateStatePdu : public DtStateMsg
{
public:

    // Constructors
    DtAggregateStatePdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtAggregateStatePdu(const DtNetAggregateStatePdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtAggregateStatePdu(const DtAggregateStatePdu& orig);

    // destructor
    virtual ~DtAggregateStatePdu();

    // Assignment operator
    DtAggregateStatePdu& operator=(const DtAggregateStatePdu& orig);

    // Set/Get aggregate entity identifier
    virtual void setAggregateID(const DtEntityIdentifier& entity);
    virtual const DtEntityIdentifier& aggregateID() const;

    // Set/Get the force ID
```

```

virtual void setForceID(DtForceType force);
virtual DtForceType forceID() const;

// Set/Get the aggregate state
virtual void setAggregateState(DtAggregateState value);
virtual DtAggregateState aggregateState() const;

// Set/Get the aggregate type
virtual void setAggregateType(const DtEntityType& type);
virtual const DtEntityType& aggregateType() const;

// Set/Get the formation
virtual void setFormation(DtAggregateFormation value);
virtual DtAggregateFormation formation() const;

// Set/Get marking character set
virtual void setMarkingCharacterSet(DtCharacterSet value);
virtual DtCharacterSet markingCharacterSet() const;

// Set/Get marking text (copies 31 bytes)
virtual void setMarkingText(DtU8* copyFrom);
virtual void getMarkingText(DtU8* copyTo) const;

// Set/Get the dimensions
virtual void setDimensions(DtConstVector dims);
virtual DtConstVector dimensions() const;

// Set/Get orientation
virtual void setOrientation(const DtTaitBryan& orientation);
virtual const DtTaitBryan& orientation() const;

// Set/Get the location of center of mass
virtual void setLocation(DtConstVector coords);
virtual DtConstVector location() const;

// Set/Get the velocity
virtual void setVelocity(DtConstVector coords);
virtual DtConstVector velocity() const;

// Set/Get the number of DIS sub-aggregates. Set returns DtFalse
if the
// requested number is too large.
virtual DtBoolean setNumberOfDIS_Subaggregates(DtU16 newNumberOf-
Aggregates);
virtual DtU16 numberOfDIS_Subaggregates() const;

// Set/Get the number of DIS entities. Set returns DtFalse if the
// requested number is too large.
virtual DtBoolean setNumberOfDIS_Entities(DtU16 newNumberOfEnti-
ties);

```

```
virtual DtU16 numberOfDIS_Entities() const;

// Set/Get the number of silent aggregates. Set returns DtFalse
if the
// requested number is too large.
virtual DtBoolean setNumberOfSilentSubaggregates(
    DtU16 newNumberOfAggregates);
virtual DtU16 numberOfSilentSubaggregates() const;

// Set/Get the number of silent entities. Set returns DtFalse if
the
// requested number is too large.
virtual DtBoolean setNumberOfSilentEntities(DtU16 newNumberOfEn-
tities);
virtual DtU16 numberOfSilentEntities() const;

// Set/Get a DIS sub-aggregate ID. Returns DtFalse if the index
// is invalid.
virtual DtBoolean setSubaggregateID(int index,
    const DtEntityIdentifier& aggID);
virtual DtBoolean getSubaggregateID(int index,
    DtEntityIdentifier& aggID) const;

// Set/Get a DIS entity ID. Returns DtFalse if the index is
invalid.
virtual DtBoolean setEntityID(int index,
    const DtEntityIdentifier& entID);
virtual DtBoolean getEntityID(int index,
    DtEntityIdentifier& entID) const;

// Set/Get a silent aggregate list. Returns DtFalse if the index
is
// invalid.
virtual DtBoolean setSilentAggregateList(int index,
    const DtSilentAggregateList& aggList);
virtual DtBoolean getSilentAggregateList(int index,
    DtSilentAggregateList& aggList) const;

// Set/Get a silent entity list. Returns DtFalse if the index is
// invalid.
virtual DtBoolean setSilentEntityList(int index,
    const DtSilentEntityList& entList);
virtual DtBoolean getSilentEntityList(int index,
    DtSilentEntityList& entList) const;

// Set/Get the number of datum records. Set returns DtFalse
// if the requested number is too large.
virtual DtBoolean setNumberOfDatumRecords(DtU32 newNumberOf-
Records);
virtual DtU32 numberOfDatumRecords() const;
```

```
// Set/Get a variable datum record. Returns DtFalse if the index
is
// invalid.
virtual DtBoolean setVariableDatumRecord(int index,
    const DtVariableDatumRecord& rec);
virtual DtBoolean getVariableDatumRecord(int index,
    DtVariableDatumRecord& rec) const;

// Get a constant reference to a variable datum record
virtual const DtVariableDatumRecord& variableDatumRecordConstRef(
    int index) const;

// Print the data contained by this PDU
virtual void printData() const;

// WARNING: These methods are obsolete but have been retained for
backward
// compatibility. These methods may be removed in a future
release.
// Instead, the methods setVariableDatumRecord, getVariableDatum-
Record,
// and variableDatumRecordConstRef should be used.

// Set/Get the datum length (in bits). Returns DtFalse if
// the index is invalid. Set may destroy data in the record.
virtual DtBoolean setDatumBitLength(int index, DtU32 newRecord-
BitLength);
virtual DtBoolean getDatumBitLength(int index, DtU32& length)
const;

// Set/Get the variable datum ID. Returns DtFalse if the index
// is invalid.
virtual DtBoolean setDatumID(int index, DtDatumParam value);
virtual DtBoolean getDatumID(int index, DtDatumParam& value)
const;

// Set/Get datum value as a DtFloat64. Returns DtFalse if the
index
// is invalid or if the field length is not 64 bits.
virtual DtBoolean setDatumValFloat64(int index, DtFloat64 value);
virtual DtBoolean getDatumValFloat64(int index, DtFloat64& value)
const;

// Set/Get datum value as a DtFloat32. Returns DtFalse if the
index
// is invalid or if the field length is not 32 bits.
virtual DtBoolean setDatumValFloat32(int index, DtFloat32 value);
virtual DtBoolean getDatumValFloat32(int index, DtFloat32& value)
const;
```

```
// Set/Get datum value as a DtInt32. Returns DtFalse if the index
// is invalid or if the field length is not 32 bits.
virtual DtBoolean setDatumValInt32(int index, DtInt32 value);
virtual DtBoolean getDatumValInt32(int index, DtInt32& value)
const;

// Set/Get datum value as a DtU32. Returns DtFalse if the index
// is invalid or if the field length is not 32 bits.
virtual DtBoolean setDatumValU32(int index, DtU32 value);
virtual DtBoolean getDatumValU32(int index, DtU32& value) const;

// Set/Get datum value as a DtInt16. Returns DtFalse if the index
// is invalid or if the field length is not 16 bits.
virtual DtBoolean setDatumValInt16(int index, DtInt16 value);
virtual DtBoolean getDatumValInt16(int index, DtInt16& value)
const;

// Set/Get datum value as a DtU16. Returns DtFalse if the index
// is invalid or if the field length is not 16 bits.
virtual DtBoolean setDatumValU16(int index, DtU16 value);
virtual DtBoolean getDatumValU16(int index, DtU16& value) const;

// Set/Get datum value as a DtU8. Returns DtFalse if the index
// is invalid or if the field length is not 8 bits.
virtual DtBoolean setDatumValU8(int index, DtU8 value);
virtual DtBoolean getDatumValU8(int index, DtU8& value) const;

// Set/Get datum value as a byte array. Returns DtFalse if the
index
// is invalid. Data is copied to and from the network PDU struc-
ture
// according to the datum length of the record.
virtual DtBoolean setDatumValByteArray(int index, const DtU8*
setFrom);
virtual DtBoolean getDatumValByteArray(int index, DtU8* copyTo)
const;

// Get a pointer to datum value. Returns DtFalse if the index is
invalid.
virtual DtBoolean getDatumValPtr(int index, DtNetU8*& datumPtr);

public:

// Create a DtAggregateStatePdu.
// Caller is responsible for deletion.
static DtPdu* create(const DtNetPacket* initial,
DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

// Add (register)/Remove (deregister) function to be called when
```

```
// pdu is received.
static void addCallback(DtExerciseConn* conn,
    DtAggregateStatePduCb cb, void* usr);
static void removeCallback(DtExerciseConn* conn,
    DtAggregateStatePduCb cb, void* usr);
```

protected:

```
// Returns the internal PDU kind
virtual DtPduKind internalGetPduKind() const;

// Casts the netPdu element to our network struct
// Non-virtual since you can't override based on return type
virtual DtNetAggregateStatePdu* netPduStruct() const;

// Casts the 'this' pointer for changing elements in a const
function
// Non virtual since you can't override based on return type
DtAggregateStatePdu* self() const;

// Returns a pointer to the first byte of variable length data
virtual char* var() const;

// Returns an offset from the beginning of variable length data
to
// a particular subaggregate ID.
virtual int subaggregateID_Offset(int index) const;

// Returns an offset from the beginning of variable length data
to
// a particular entity ID.
virtual int entityID_Offset(int index) const;

// Returns an offset from the beginning of variable length data
to
// the padding following the entity ID records.
virtual int paddingOffset() const;

// Returns the number of padding bytes which must follow the
entity
// ID records.
virtual int paddingBytes() const;

// Returns an offset from the beginning of variable length data
to
// a particular silent aggregate list.
virtual int silentAggregateOffset(int index) const;

// Returns an offset from the beginning of variable length data
to
```

```
// a particular silent entity system list.
virtual int silentEntityOffset(int index) const;

// Returns an offset from the beginning of variable length data
to
// the number of variable datum records field.
virtual int numberOfDatumRecordsOffset() const;

// Returns an offset from the beginning of variable length data
to
// a particular variable datum record.
virtual int datumRecordOffset(int index) const;

protected:

// Size of PDU with zero data bytes.
static int theStaticSize;

protected:

// Temporary objects for reference returns
DtEntityIdentifier myEntityID;
DtEntityType myEntityType;
DtVector myVector;
DtTaitBryan myAngles;
DtVariableDatumRecord myRefRec;
DtVariableDatumRecord myDumRec;
};
#endif
#endif
```


aggregateSR.h

```

#if DtDISVERS >= 5

#include "chooseDefines.h"

#include "baseEntitySR.h"
#include "hostStructs.h"
#include "mathTypes.h"

#include "aggEntList.h"
#include "aggAggList.h"
#include "varDatumRec.h"
#include "objDesList.h"

// class DtAggregateStateRepository:
//
// Instances of DtAggregateStateRepository are used to maintain the
// state of an aggregate. It provides an interface which covers the
// various features which define the state of an aggregate.

class DT_DLL_VL DtAggregateStateRepository : public
DtBaseEntityStateRepository
{
public:

    // default constructor
    DtAggregateStateRepository();

    // destructor
    virtual ~DtAggregateStateRepository();

    // copy constructor
    DtAggregateStateRepository(const DtAggregateStateRepository&
orig);

    // assignment operator
    DtAggregateStateRepository& operator=(
        const DtAggregateStateRepository& orig);

    // return an empty object of the same type as this
    virtual DtStateRepository* clone() const;

    // Set/Get the aggregate's identifier. Calls down to base class.
    virtual void setAggregateID(const DtEntityIdentifier& id);
    virtual const DtEntityIdentifier& aggregateID() const;

    // Set/Get the force type.

```

```
virtual void setForceId(DtForceType force);
virtual DtForceType forceId() const;

// Set/Get the aggregate state.
virtual void setAggregateState(DtAggregateState value);
virtual DtAggregateState aggregateState() const;

// Set/Get the aggregate type. Calls down to base class.
virtual void setAggregateType(const DtEntityType& type);
virtual const DtEntityType& aggregateType() const;

// Set/Get the formation.
virtual void setFormation(DtAggregateFormation value);
virtual DtAggregateFormation formation() const;

// Set/Get marking character set.
virtual void setMarkingCharacterSet(DtCharacterSet value);
virtual DtCharacterSet markingCharacterSet() const;

// Set/Get marking text (copies 31 bytes).
virtual void setMarkingText(DtU8* copyFrom);
virtual void getMarkingText(DtU8* copyTo) const;
virtual const char* markingText() const;

// Set/Get the dimensions.
virtual void setDimensions(DtConstVector dims);
virtual DtConstVector dimensions() const;

// Return the list of published entities in the aggregate.
const DtGlobalObjectDesignatorList& entities() const;
DtGlobalObjectDesignatorList* entities();

// Return the list of published subaggregates in the aggregate.
DtGlobalObjectDesignatorList* subAggregates();
const DtGlobalObjectDesignatorList& subAggregates() const;

// Set/Get the number of DIS sub-aggregates. Set returns DtFalse
if the
// requested number is too large. Changing this number will
erase all
// previous data. Can be done through subAggregates() instead.
virtual DtBoolean setNumberOfDIS_Subaggregates(int newNumberOfAg-
gregates);
virtual int numberOfDIS_Subaggregates() const;

// Set/Get the number of DIS entities. Set returns DtFalse if the
// requested number is too large. Changing this number will
erase all
// previous data. Can be done through entities() instead.
```

```

    virtual DtBoolean setNumberOfDIS_Entities(int newNumberOfEntities);
    virtual int numberOfDIS_Entities() const;

    // Set/Get the number of silent aggregates. Set returns DtFalse if the
    // requested number is too large. Changing this number will
    // erase all
    // previous data.
    virtual DtBoolean setNumberOfSilentSubaggregates(
        int newNumberOfAggregates);
    virtual int numberOfSilentSubaggregates() const;

    // Set/Get the number of silent entities. Set returns DtFalse if the
    // requested number is too large. Changing this number will
    // erase all
    // previous data.
    virtual DtBoolean setNumberOfSilentEntities(int newNumberOfEntities);
    virtual int numberOfSilentEntities() const;

    // Set/Get a DIS sub-aggregate ID. Returns DtFalse if the index
    // is invalid. Can be done through subAggregates() instead.
    virtual DtBoolean setSubaggregateID(int index,
        const DtGlobalObjectDesignator& aggID);
    virtual DtBoolean getSubaggregateID(int index,
        DtGlobalObjectDesignator& aggID) const;

    // Set/Get a DIS entity ID. Returns DtFalse if the index is
    // invalid.
    // Can be done through entities() instead.
    virtual DtBoolean setEntityID(int index,
        const DtGlobalObjectDesignator& entID);
    virtual DtBoolean getEntityID(int index,
        DtGlobalObjectDesignator& entID) const;

    // Set/Get a silent aggregate list. Returns DtFalse if the index
    // is
    // invalid.
    virtual DtBoolean setSilentAggregateList(int index,
        const DtSilentAggregateList& aggList);
    virtual DtBoolean getSilentAggregateList(int index,
        DtSilentAggregateList& aggList) const;

    // Set/Get a silent entity list. Returns DtFalse if the index is
    // invalid.
    virtual DtBoolean setSilentEntityList(int index,
        const DtSilentEntityList& entList);
    virtual DtBoolean getSilentEntityList(int index,

```

```
        DtSilentEntityList& entList) const;

    // Set the number of variable datum records. Returns DtFalse if
    the number
    // of records is too large. This will erase the contents of
    existing
    // datum records.
    virtual DtBoolean setNumberOfDatumRecords(int newNumberOf-
Records);

    // Get the number of variable datum records
    virtual int numberOfDatumRecords() const;

    // Set/Get a variable datum record. Returns DtFalse if the index
    is
    // invalid.
    virtual DtBoolean setVariableDatumRecord(int index,
        const DtVariableDatumRecord& rec);
    virtual DtBoolean getVariableDatumRecord(int index,
        DtVariableDatumRecord& rec) const;

    // Get a reference to a variable datum record.
    virtual DtVariableDatumRecord& variableDatumRecordRef(int index);

    // Get a constant reference to a variable datum record.
    virtual const DtVariableDatumRecord& variableDatumRecordConstRef(
        int index) const;

public:

    // Print the state data.
    virtual void printData() const;

public:

    static DtAggregateStateRepository* create();

protected:

    // Cast off const-ness for trivial internal change.
    // Non-virtual since can't override on return type.
    DtAggregateStateRepository* self() const;

protected:

    DtForceType myForceID;
    DtAggregateState myAggregateState;
    DtAggregateFormation myFormation;
    DtCharacterSet myMarkingCharSet;
```

```
// Aggregate state PDU marking is 31 bytes long, RPR marking struct
is 11
#ifdef DtHLA
#define AGGREGATE_SR_MARKING_TEXT_LENGTH 11
#else
#define AGGREGATE_SR_MARKING_TEXT_LENGTH 31
#endif

    char myMarkingText[AGGREGATE_SR_MARKING_TEXT_LENGTH];
    DtVector myDimensions;

    DtGlobalObjectDesignatorList myEntities;
    DtGlobalObjectDesignatorList mySubAggs;

    int myNumberOfSilentAggregateTypes;
    DtSilentAggregateList* mySilentAggregates;

    int myNumberOfSilentEntityTypes;
    DtSilentEntityList* mySilentEntities;

    int myNumberOfVariableDatumRecords;
    DtVariableDatumRecord* myVariableDatumRecords;

    DtVariableDatumRecord myDumRec;

};
#endif
```

antPatRec.h

```
#include "MachineTypes.h"
#include "chooseDefines.h"

// class DtAntennaPatternRecord:
//
// Instances of DtAntennaPatternRecord are

class DT_DLL_VL DtAntennaPatternRecord
{
public:

    // default constructor
    DtAntennaPatternRecord();

    // copy constructor
    DtAntennaPatternRecord(const DtAntennaPatternRecord& orig);

    // destructor
    virtual ~DtAntennaPatternRecord();

    // assignment operator
    DtAntennaPatternRecord& operator=
        (const DtAntennaPatternRecord& orig);

    // equality operator
    DtBoolean operator==
        (const DtAntennaPatternRecord& orig) const;

    // inequality operator
    DtBoolean operator!=
        (const DtAntennaPatternRecord& orig) const;

    // Set from network representation
    virtual void setFromNet(const void* net, int size);

    // Returns network representation, fills size with size of
    netRep.
    // Return value is pointer to internal object with same lifetime
    as
    // this. Do not delete.
    virtual const void* netRep(int* size = NULL) const;

    // Returns size of network representation.
    virtual int netSize() const;

    // Returns a new'ed clone of the instance. Caller deletes.
    virtual DtAntennaPatternRecord* clone() const;
```

```
// prints the data
virtual void printData() const;

protected:

    char* myNetRep;
    int myNetSize;
};
```

approximator.h

```
#include "mathTypes.h"
#include "vlTime.h"
#include "chooseDefines.h"

// class DtApproximator:
//
// DtApproximator is an abstract base class. It is used to provide
// approximated position and orientation information for entities.
// Position and orientation information is approximated over time
// from the last set absolute (non-approximated) information.

class DT_DLL_VL DtApproximator
{
public:

    // destructor
    virtual ~DtApproximator();

    // assignment operator
    DtApproximator& operator=(const DtApproximator& orig);

    // Create a copy of this object. Memory is allocated and it is
    the
    // responsibility of the caller to free the memory via delete.
    virtual DtApproximator* clone() const = 0;

    // Set absolute location.
    virtual void setLocation(DtConstVector loc,
        DtTime t = DtGetSimTime()) = 0;
    // Get the approximated location at the specified time.
    virtual DtConstVector approxLocation(
        DtTime t = DtGetSimTime()) const = 0;

    // Set absolute velocity.
    virtual void setVelocity(DtConstVector vel,
        DtTime t = DtGetSimTime()) = 0;
    // Get the approximated velocity at the specified time.
    virtual DtConstVector approxVelocity(
        DtTime t = DtGetSimTime()) const = 0;

    // Set absolute orientation.
    virtual void setOrientation(const DtTaitBryan& ori,
        DtTime t = DtGetSimTime()) = 0;
    // Get approximated orientation (euler angles) at the specified
    time.
    virtual const DtTaitBryan& approxEuler(
        DtTime t = DtGetSimTime()) const = 0;
```



```
// Set/Get absolute acceleration.
virtual void setAcceleration(DtConstVector acc,
    DtTime t = DtGetSimTime()) = 0;
virtual DtConstVector acceleration(
    DtTime t = DtGetSimTime()) const = 0;

// Set/Get absolute rotational velocity.
virtual void setRotationalVelocity(DtConstVector vel,
    DtTime t = DtGetSimTime()) = 0;
virtual DtConstVector rotationalVelocity(
    DtTime t = DtGetSimTime()) const = 0;

// Get the approximated data body to geocentric coordinate
matrix.
virtual DtConstDcm approxBodyToGeoc(DtTime t = DtGetSimTime())
const = 0;

protected:

    // Constructors are protected since this is an abstract base
    class.

    // default constructor
    DtApproximator();

    // copy constructor
    DtApproximator(const DtApproximator& orig);
};
```

apSpec.h

```
#include <stdlib.h>
#include "chooseDefines.h"

const int DtMAX_PARAMS_PER_PART = 31;

typedef struct DtArtPartSpec
{
    int partType;
    int attachedType;
    int params[DtMAX_PARAMS_PER_PART];
} DtArtPartSpec;

const int DtAttachedToBase = 0;
#define DtNoArtParts ((DtArtPartSpec*) NULL)
#define DtNullArtPartSpec 0,0,{0}
```

arealObjStPdu.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "hostStructs.h"
#include "pdu.h"
#include "mathTypes.h"
#include "exerciseConn.h"

#include "packets/arealObjStP.h"

#include "objectType.h"

// class DtArealObjectStatePdu:
//
// Instances of DtArealObjectStatePdu are used to communicate infor-
// mation
// about the addition/modification of a synthetic object which is
// geometrically
// anchored to the terrain with multiple points.

// DtArealObjectStatePdu callback function signature.
typedef void (*DtArealObjectStatePduCb)(
    DtArealObjectStatePdu* pdu, void* usr);

class DT_DLL_VL DtArealObjectStatePdu : public DtPdu
{
public:
    // Constructors
    DtArealObjectStatePdu(DtBufferPtr buffer =
DtUSE_INTERNAL_BUFFER);
    DtArealObjectStatePdu(const DtNetArealObjectStatePdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtArealObjectStatePdu(const DtArealObjectStatePdu& orig);

    // destructor
    virtual ~DtArealObjectStatePdu();

    // Create a DtArealObjectStatePdu.
    // Caller is responsible for deletion.
    static DtPdu* create(const DtNetPacket* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // Add (register)/Remove (deregister) function to be called when
    // pdu is received.

```

```
static void addCallback(DtExerciseConn* conn,
    DtArealObjectStatePduCb cb, void* usr);
static void removeCallback(DtExerciseConn* conn,
    DtArealObjectStatePduCb cb, void* usr);

// Assignment operator
DtArealObjectStatePdu& operator=(const DtArealObjectStatePdu&
orig);

// Set/Get object ID
virtual void setObjectID(const DtEntityIdentifier& id);
virtual const DtEntityIdentifier& objectID() const;

// Set/Get referenced object ID
virtual void setReferencedObjectID(const DtEntityIdentifier& id);
virtual const DtEntityIdentifier& referencedObjectID() const;

// Set/Get update number
virtual void setUpdateNumber(DtU16 field);
virtual DtU16 updateNumber() const;

// Set/Get the force ID
virtual void setForceID(DtForceType force);
virtual DtForceType forceID() const;

// Set/Get the modifications (by 8-bit integer)
virtual void setModifications(DtU8 value);
virtual DtU8 modifications() const;

// Set/Get the modifications (by field)
virtual void setModificationsField(
    DtModificationField field, DtBoolean value);
virtual DtBoolean modificationsField(
    DtModificationField field) const;

// Set/Get the object type
virtual void setObjectType(const DtObjectType& type);
virtual const DtObjectType& objectType() const;

// Set/Get appearance, the lower 16 bits (as an integer)
virtual void setAppearance(DtU16 app);
virtual DtU16 appearance() const;

// Set/Get appearance by attribute for the lower 16 bits
virtual void setAppearanceAttribute(
    DtAppearAttribute attr, DtAppearAttrVal val);
virtual DtAppearAttrVal appearanceAttribute(
    DtAppearAttribute attr) const;
```

```

    // Set/Get specific appearance, the upper 32 bits (as a 32 bit
integer)
    virtual void setSpecificAppearance(DtU32 app);
    virtual DtU32 specificAppearance() const;

    // Set/Get specific appearance by attribute, the upper 32 bits
    virtual void setSpecificAppearanceAttribute(
        const DtAppearAttribute attr, const DtAppearAttrVal val);
    virtual DtAppearAttrVal specificAppearanceAttribute(
        const DtAppearAttribute attr) const;

    // Set/Get the number of points
    virtual DtBoolean setNumPoints(DtU16 num);
    virtual DtU16 numPoints() const;

    // Set/Get the requester ID
    virtual void setRequesterID(const DtSimulationAddress& type);
    virtual const DtSimulationAddress& requesterID() const;

    // Set/Get the receiving ID
    virtual void setReceivingID(const DtSimulationAddress& type);
    virtual const DtSimulationAddress& receivingID() const;

    // Set/Get a point location. Returns DtFalse if the index is out
of range.
    virtual DtBoolean setPointLocation(int index,
        DtConstVector init);
    virtual DtBoolean getPointLocation(int index,
        DtVectorRef record) const;

    // Print the data contained by this PDU
    virtual void printData() const;

protected:

    // Returns the internal PDU kind
    virtual DtPduKind internalGetPduKind() const;

    // Casts the netPdu element to our network struct
    virtual DtNetAreaObjectStatePdu* netPduStruct() const;

    // Casts the 'this' pointer for changing elements in a const
function
    // Non-virtual since you can't override based on return type
    DtAreaObjectStatePdu* self() const;

    // Returns a pointer to the first byte of variable length data
    virtual char* var() const;

protected:

```

```
// Size of PDU with zero parameter records.
static int theStaticSize;

protected:

    // Temporary objects for reference returns
    DtEntityIdentifier myEntityID;
    DtObjectType myObjectType;
    DtSimulationAddress mySimulationAddress;

};
#endif
#endif
```

artPart.h

```

#include "MachineTypes.h"
#include "mathTypes.h"
#include "apSpec.h"
#include "vlTime.h"

// Used only for internal representation of Art Param information

typedef struct DtInternalArtParam
{
    float myValue;
    int myParamNum;
    int myNext;
    DtTime myTimeSet;
} DtInternalArtParam;

// class DtArtPart:
//
// Instances of DtArtPart are used by DtArtPartList to handle indi-
// vidual
// articulated parts (i.e. this is a helper class to DtArtPartList).

// One Articulated Part
class DT_DLL_VL DtArtPart
{
    friend class DtArtPartList;

public:
    // Set/Get the value for the specified parameter type.
    // Returns DtFalse if the parameter type has not been added,
    // returns DtTrue otherwise.
    virtual DtBoolean setVal(int paramType, DtFloat32 value);
    virtual DtBoolean getVal(int paramType, DtFloat32& value) const;

    // Get the value for the specified parameter type.
    // Note this does not check existence of the parameter type and
    // should only be called when you know that it does exist (as from
    // nextValidParamType()).
    virtual DtFloat32 val(int paramType) const;

    // Get the part type.
    virtual int partType() const;

    // Get the part number.
    virtual int partNum() const;

    // Get the part this part is attached to.

```

```
virtual int partAttachedTo() const;

// Get the number of parameters.
virtual int numParams() const;

// Get the first valid (added) parameter type.
virtual int firstValidParamType() const;

// Get the next valid parameter type.
virtual int nextValidParamType(int paramType) const;

// Returns the index of the parameter in a multiple part list
// as is found in HLA messages and DIS PDUs.
virtual int paramNum(int paramType) const;

// Get the euler angles.
// Euler is filled with the angles phi, theta and psi of the part
// with respect to its neutral orientation.
virtual const DtTaitBryan& angles() const;
virtual void getAngles(DtTaitBryan& euler) const;

// Get the translations.
// Vec is filled in with a vector representing the current trans-
lation,
// in x, y and z of the part from its neutral position.
virtual DtConstVector translations() const;
virtual void getTranslations(DtVectorRef vec) const;

// Set/Get whether or not to do approximating.
virtual void setApproximating(DtBoolean onOff);
virtual DtBoolean approximating() const;

// Print the part data.
virtual void print();

public:

// Get whether the parameter type is actually a rate.
static int artParamIsRate(int paramType);

// Get the part type from the part and parameter value.
static int computePartType(int partPlusParam);

// Get the parameter type from the part and parameter value.
static int computeParameter(int partPlusParam);

protected:

// Constructors are protected since this class is managed by
// DtArtPartList.
```



```

// default constructor
DtArtPart();

// constructor
DtArtPart(const DtArtPartSpec& spec);

// destructor
virtual ~DtArtPart();

// copy constructor
DtArtPart(const DtArtPart& orig);

// assignment operator
DtArtPart& operator=(const DtArtPart& orig);

// Set the parameter number of the specified parameter type.
virtual void setParamNum(int paramType, int num);

// Set the part number.
virtual void setPartNum(int num);

// Add the specified parameter type to the part.
// Note that this assumes that parameters are always added
// in order from lowest to highest.
virtual void addParam(int paramType);

// Find the next valid parameter type.
virtual int findNextValidParamType(int paramType) const;

// Get the approximated value.
virtual DtFloat32 approxVal(int paramType) const;

// Get the last set (absolute) value.
virtual DtFloat32 lastSetVal(int paramType) const;

// Get the time at which the value was last set.
virtual DtTime timeSet(int paramType) const;

// Cast off const-ness for trivial internal change.
// Non-virtual since can't override on return type.
DtArtPart* self() const;

```

protected:

```

int myPartType;
int myPartNum;
int myPartAttachedTo;
int myNumParams;
// myVals is an array of DtMAX_PARAMS_PER_PART + 1 items

```

```
// which are themselves pointers.  
DtInternalArtParam** myVals;  
int myFirstParam;  
int myLastParam;  
DtBoolean myApproxFlag;  
  
DtTaitBryan myAngles;  
DtVector myVector;  
};
```

artPartList.h

```

#include "vList.h"
#include "artPart.h"
#include "NetStructs.h"

// class DtArtPartList:
//
// Instances of DtArtPartList are used to manage articulated parts.
// Articulated part spec lists are mainly used to generate these
// lists.

class DT_DLL_VL DtArtPartList
{
public:

    // default constructor
    DtArtPartList();

    // constructor
    DtArtPartList(const DtArtPartSpec* specList);

    // destructor
    virtual ~DtArtPartList();

    // copy constructor
    DtArtPartList(const DtArtPartList& orig);

    // assignment operator
    DtArtPartList& operator=(const DtArtPartList& orig);

    // The init functions don't set values for the parameters,
    // they merely set up the part and parameter types.
    // Non-virtual since called from ctor.
    void init(const DtArtPartSpec* partSpecs);

#if DtHLA

#include "chooseDefines.h"

    // The init functions don't set values for the parameters,
    // they merely set up the part and parameter types. numParams
    // is the total number of records in netParams, which might
include
    // attached parameter records that will be ignored.
    virtual void init(int numParams,
        const DtNetArtParamRecord* netParams);

    // sets values from net representation, but does not change the

```

```
// set of valid parts and parameters. numParams
// is the total number of records in netParams, which might
include
// attached parameter records that will be ignored.
virtual void setValsFromNet(int numParams,
    const DtNetArtParamRecord* netParams);

// initializes the part types and parameters, and sets values as
well.
// numParams is the total number of records in netParams, which
might
// include attached parameter records that will be ignored.
virtual void setFromNet(int numParams,
    const DtNetArtParamRecord* netParams);

// fills in netParams based on current values
virtual void fillNetParams(DtNetArtParamRecord* netParams) const;

#endif // DtHLA

#if !DtHLA

#include "chooseDefines.h"

// The init functions don't set values for the parameters,
// they merely set up the part and parameter types. numParams
// is the total number of records in netParams, which might
include
// attached parameter records that will be ignored.
virtual void init(int numParams,
    const DtNetArtParamRecord* netParams);

// sets values from net representation, but does not change the
// set of valid parts and parameters. numParams
// is the total number of records in netParams, which might
include
// attached parameter records that will be ignored.
virtual void setValsFromNet(int numParams,
    const DtNetArtParamRecord* netParams);

// initializes the part types and parameters, and sets values as
well.
// numParams is the total number of records in netParams, which
might
// include attached parameter records that will be ignored.
virtual void setFromNet(int numParams,
    const DtNetArtParamRecord* netParams);

// fills in netParams based on current values
virtual void fillNetParams(DtNetArtParamRecord* netParams) const;
```

```
#endif // !DtHLA

    // removes all parts and parameters from list
    // Non-virtual since called from ctor/dtor.
    void clear();

    // Set/Get the specified part type's specified parameter type
    value.
    // Returns DtFalse if the part or parameter are not found, other-
    wise
    // returns DtTrue.
    virtual DtBoolean setArtParamVal(int partType,
        int paramType, DtFloat32 val);
    virtual DtBoolean getArtParamVal(int partType,
        int paramType, DtFloat32& val) const;

    // Get the number of articulated parts.
    virtual int numArtParts() const;

    // Get the number of articulated parameters.
    virtual int numArtParams() const;

    // Get the articulated part from the part number.
    virtual DtArtPart* artPartFromPartNum(int partNum) const;

    // Get the first articulated part with the part type.
    virtual DtArtPart* artPartFromPartType(int partType) const;

    // returns 0 if there is no such part, otherwise the part number
    virtual int partNumFromPartType(int partType) const;

    // return the list of DtArtPartSpecs describing the parts in the
    list.
    virtual const DtArtPartSpec* artPartSpecs() const;

    // Print the articulated part list.
    virtual void print() const;

public:

    // Fill the articulated part spec list from the net parameter
    // record list. numParams is the total number of records in
    // netParams, which might include attached parameter records that
    // will be ignored.
    #if DtHLA

    #include "chooseDefines.h"

    static void fillSpecList(
```

```
        int numParams,
        const DtNetArtParamRecord* netParams,
        DtArtPartSpec* specList);

    // Returns true if param is an articulated parameter record,
    // false if its an attached part record.
    static DtBoolean artNotAttached(const DtNetArtParamRecord*
param);

#endif // DtHLA

#if    !DtHLA

#include "chooseDefines.h"

    static void fillSpecList(
        int numParams,
        const DtNetArtParamRecord* netParams,
        DtArtPartSpec* specList);

    // Returns true if param is an articulated parameter record,
    // false if its an attached part record.
    static DtBoolean artNotAttached(const DtNetArtParamRecord*
param);

#endif // !DtHLA

protected:

    // Returns whether the part number is within bounds.
    virtual DtBoolean partNumInBounds(int partNum) const;

    // Cast off const-ness for trivial internal change.
    // Non-virtual since can't override on return type.
    DtArtPartList* self() const;

protected:

    DtArtPart** myPartList;
    int myNumArtParts;
    int myNumArtParams;
    DtArtPartSpec* mySpecList;
};
```

artPartThresh.h

```

#include "artPartList.h"

// class DtArtPartThresholder:
//
// DtArtPartThresholder is used to store art-part-related thresh-
// olds,
// and to perform some art part thresholding.

class DT_DLL_VL DtArtPartThresholder
{
public:

    enum DtArtPartThresholdType
    {
        DtArtPartDfltThreshMode = -1,
        DtArtPartIndividualThresh = 0,
        DtArtPartAggregateThresh = 1
    };

public:

    // Constructor
    DtArtPartThresholder();

    // Destructor
    ~DtArtPartThresholder();

    // copy constructor
    DtArtPartThresholder(const DtArtPartThresholder& orig);

    // assignment operator
    DtArtPartThresholder& operator=(const DtArtPartThresholder&
orig);

    // Set/Get rotation threshold in radians.
    // Negative number means use default.
    virtual void setRotationThreshold(DtReal thresh);
    virtual double rotationThreshold() const;

    // Returns the "RotationTrace" threshold
    virtual double rotationTraceThreshold() const;

    // Set/Get translation threshold in meters.
    // Negative number means use default.
    virtual void setTranslationThreshold(DtReal thresh);
    virtual double translationThreshold() const;

```

```
// Set/Get fractional threshold (fraction between 0 and 1).
// Negative number means use default.
virtual void setFractionalThreshold(DtReal thresh);
virtual double fractionalThreshold() const;

// Set/Get the threshold mode. Use DtArtPartDfltThreshMode
// to indicate use of the default.
virtual void setThreshMode(DtArtPartThresholdType mode);
virtual DtArtPartThresholdType threshMode() const;

// Get whether or not an aggregate threshold (such as position,
// translation, rotation, etc.) has been exceeded.
virtual DtBoolean artPartsExceedAggThreshold(
    const DtArtPartList& localList,
    const DtArtPartList& remoteList) const;

// Get whether or not an individual parameter threshold has been
// exceeded.
virtual DtBoolean artPartsExceedIndivThreshold(
    const DtArtPartList& localList,
    const DtArtPartList& remoteList) const;

// Get the thresholding value (functional, translational, or
// rotational) for the parameter type.
virtual float artPartThreshold(int paramType) const;
```

public:

```
// Set/Get the threshold mode.
static void setDfltThreshMode(DtArtPartThresholdType mode);
static DtArtPartThresholdType dfltThreshMode();

// Set/Get fractional threshold value.
static void setDfltFractionalThreshold(DtFloat32 thresh);
static DtFloat32 dfltFractionalThreshold();

// Set/Get rotational threshold value.
static void setDfltRotationThreshold(DtReal r);
static double dfltRotationThreshold();

// Set/Get translational threshold value.
static void setDfltTranslationThreshold(DtReal t);
static double dfltTranslationThreshold();
```

protected:

```
DtArtPartThresholdType myThreshMode;
float myFracThresh;
double myTransThresh;
```



```
double myRotThresh;  
double myTraceRotThresh;  
  
DtBoolean myUsingDfltMode;  
DtBoolean myUsingDfltTrans;  
DtBoolean myUsingDfltRot;  
DtBoolean myUsingDfltFrac;  
  
protected:  
  
    static DtArtPartThresholdType theDfltThreshMode;  
    static float theDfltFracThresh;  
    static double theDfltTransThresh;  
    static double theDfltRotThresh;  
    static double theDfltTraceRotThresh;  
};
```

attPartList.h

```
#include "hostStructs.h"

// This constant can be passed to a VR-Link function that
// expects a list of attached part stations to indicate
// an empty list.
const int DtNoAttachedParts[1] = {0};

enum DtAttachedPartStatus
{
    DtAttachedPartSuccess = -2,
    DtNoSuchAttachStation = -1,
    DtPartNotAttached = 0,
    DtPartAttached = 1
};

// class DtAttachedPartList:
//
// Instances of DtAttachedPartList are lists of attached parts.

class DT_DLL_VL DtAttachedPartList
{
public:
    // default constructor
    DtAttachedPartList();

    // copy constructor
    DtAttachedPartList(const DtAttachedPartList& orig);

    // assignment operator
    DtAttachedPartList& operator=(const DtAttachedPartList& orig);

    // destructor
    virtual ~DtAttachedPartList();

    // Set/Get the number of attached parts.
    // Set may return DtFalse if the number of requested parts is
    // invalid.
    virtual DtBoolean setNumAttachedParts(int num);
    virtual int numAttachedParts() const;

    // Returns a zero-termination list of station numbers that are
    // valid in
    // this attached part list.
    virtual const int* stationList() const;
```

```

    // Stations is a zero-terminated array of stations.
    // Init initializes the attPartList with the stations in the
array.
    virtual void init(const int* stations);

    // Set/Get the station number for a part.
    // Returns DtFalse if the index is out of range.
    virtual DtBoolean setStationNumber(int partIndex, int station);
    virtual DtBoolean getStationNumber(int partIndex, int& station)
const;

    // Returns the station number for a part.
    // This version of the function requires that partIndex be valid
    // (between 0 and numAttachedParts()).
    virtual int stationNumber(int partIndex) const;

    // Set/Get the entity type for a part.
    // Returns DtFalse if the index is out of range.
    virtual DtBoolean setEntityType(int partIndex, const DtEntity-
Type& type);
    virtual DtBoolean getEntityType(int partIndex, DtEntityType&
type) const;

    // Returns the entityType for a part, by index.
    // This version of the function requires that partIndex be valid
    // (between 0 and numAttachedParts()).
    virtual const DtEntityType& entityType(int partIndex) const;

    // Set/Get the entity type for a part.
    // Returns DtFalse if the index is out of range.
    virtual DtAttachedPartStatus setEntityTypeAtStation(int station,
const DtEntityType& type);
    virtual DtAttachedPartStatus getEntityTypeAtStation(int station,
DtEntityType& type) const;

    // Returns the entityType at a particular station.
    // If the list does not contain an entry for the station,
    // DtEntityType::nullEntityType is returned;
    virtual const DtEntityType& entityTypeAtStation(int station)
const;

    // After this call, this attached part list will reflect the
information
    // contained in netParams. Both the set of valid stations, and
    // the entity types attached to those stations might be changed.
    // numParams is the total number of records in netParams, which
might
    // include articulated parameter records that will be ignored.
    virtual void unloadNetList(int numParams,
const DtNetAttachedPart* partList);

```

```
// Load a list from network data representation.
// numAttachedParts() network part records will be written/read.
virtual void loadNetList(DtNetAttachedPart* partList) const;

// Get the network data length for this list.
virtual int netRecordLength() const;

// Returns true if we contain the exact same information as
other.
DtBoolean equivalent(const DtAttachedPartList& other) const;

// Print the information contained in the record.
virtual void print() const;

protected:

    int myNumParts;
    // zero terminated list that should always be of length
    // myNumParts + 1
    int* myStationList;

    // NULL if myNumArtParts = 0, otherwise has length = myNumParts.
    DtEntityType* myTypeList;
};
```

baseEmbedDec.h

```

#if DtHLA

#ifndef DtBaseEmbeddedSystemDecoder_H_
#define DtBaseEmbeddedSystemDecoder_H_

#include "chooseDefines.h"
#include "hStateDecoder.h"

// This is used to declare attribute decoding functions
// which are members of the class.
#define DtDECLARE_EMBEDSYS_ATTR_DECODER(attrName) \
    DtDECLARE_ATTR_DECODER(DtEmbeddedSystemRepository, attrName)

// class DtBaseEmbeddedSystemDecoder:
//
// Instances of DtBaseEmbeddedSystemDecoder are

class DT_DLL_VL DtBaseEmbeddedSystemDecoder : public DtHlaStateDe-
coder
{
public:

    // default constructor
    DtBaseEmbeddedSystemDecoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtBaseEmbeddedSystemDecoder();

    // copy constructor
    DtBaseEmbeddedSystemDecoder(const DtBaseEmbeddedSystemDecoder&
orig);

    // assignment operator
    DtBaseEmbeddedSystemDecoder& operator=
        (const DtBaseEmbeddedSystemDecoder& orig);

public:

    // Create a DtBaseEntityDecoder. Caller is responsible for dele-
    tion.
    static DtHlaStateDecoder* create(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

protected:

```

```
// Add decoding functions for the attributes.
// Note: Non-virtual since meant to be called from constructors
to
//      add decoders.
void addMyDecoders();

// static decode<AttributeName> member functions.
DtDECLARE_EMBEDSYS_ATTR_DECODER(EntityID);
DtDECLARE_EMBEDSYS_ATTR_DECODER(HostObjectID);
DtDECLARE_EMBEDSYS_ATTR_DECODER(RelativePosition);
};
// These macros are used to define simple attribute decoder func-
tions
// (members of a derived DtHlaStateDecoder that decode
// individual attributes). The simple decoding is just to
// convert the value pulled out of the ahvps to a host type using
// the cast expression castExpr, and pass it to the mutator func-
tion.
#define DtDEFINE_SIMPLE_BASE_EMBEDSYS_ATTR_DECODER_WITH_CAST( \
    attrName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_ATTR_DECODER_WITH_CAST( \
        DtBaseEmbeddedSystemDecoder, DtEmbeddedSystemRepository, \
        attrName, netType, mutator, castExpr)

#define DtDEFINE_SIMPLE_BASE_EMBEDSYS_ATTR_DECODER( \
    attrName, netType, inspector) \
    DtDEFINE_SIMPLE_BASE_EMBEDSYS_ATTR_DECODER_WITH_CAST( \
        attrName, netType, inspector, (netType))

#endif //DtBaseEmbeddedSystemDecoder_H_
```

baseEmbedEnc.h

```

#if DtHLA

#ifndef DtBaseEmbeddedSystemEncoder_H_
#define DtBaseEmbeddedSystemEncoder_H_

#include "chooseDefines.h"
#include "hStateEncoder.h"

// This is used to declare static member functions for determining
// whether or not an attribute needs to be encoded.
#define DtDECLARE_BASE_EMBEDSYS_ATTR_CHECKER(attrName) \
    DtDECLARE_ATTR_CHECKER(DtEmbeddedSystemRepository, attrName)

// This is used to declare attribute encoding functions
// which are members of the class.
#define DtDECLARE_BASE_EMBEDSYS_ATTR_ENCODER(attrName) \
    DtDECLARE_ATTR_ENCODER(DtEmbeddedSystemRepository, attrName)

// class DtBaseEmbeddedSystemEncoder:
//
// Instances of DtBaseEmbeddedSystemEncoder are

class DT_DLL_VL DtBaseEmbeddedSystemEncoder : public DtHlaStateEncoder
{
public:

    // default constructor
    DtBaseEmbeddedSystemEncoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtBaseEmbeddedSystemEncoder();

    // copy constructor
    DtBaseEmbeddedSystemEncoder(const DtBaseEmbeddedSystemEncoder&
orig);

    // assignment operator
    DtBaseEmbeddedSystemEncoder& operator=(const DtBaseEmbeddedSystemEncoder& orig);

public:

    // Create a DtBaseEntityEncoder. Caller is responsible for deletion.

```

```
static DtHlaStateEncoder* create(
    DtExerciseConn* exConn,
    DtObjClassDesc* classDesc);

protected:

    DtDECLARE_BASE_EMBEDSYS_ATTR_CHECKER(EntityID);
    DtDECLARE_BASE_EMBEDSYS_ATTR_CHECKER(HostObjectID);
    DtDECLARE_BASE_EMBEDSYS_ATTR_CHECKER(RelativePosition);

    DtDECLARE_BASE_EMBEDSYS_ATTR_ENCODER(EntityID);
    DtDECLARE_BASE_EMBEDSYS_ATTR_ENCODER(HostObjectID);
    DtDECLARE_BASE_EMBEDSYS_ATTR_ENCODER(RelativePosition);
};
// This macro is used to define simple attribute encoder functions
// (members of a derived DtHlaStateEncoder that encode
// individual attributes). The simple encoding is just to implicitly
// convert the value returned by the inspector to a netType,
// and add it to the ahvps.
#define DtDEFINE_SIMPLE_BASE_EMBEDSYS_ATTR_ENCODER( \
    attrName, netType, inspector) \
    DtDEFINE_SIMPLE_ATTR_ENCODER(DtBaseEmbeddedSystemEncoder, \
        DtEmbeddedSystemRepository, \
        attrName, netType, inspector)

// This macro is used to define simple attribute checker functions
// (members of a derived DtHlaStateEncoder that check whether
// a particular attribute is out of date. The simple implementation
// is to just compare the values returned by the two StateReps'
// inspector functions.
#define DtDEFINE_SIMPLE_BASE_EMBEDSYS_ATTR_CHECKER( \
    attrName, inspector) \
    DtDEFINE_SIMPLE_ATTR_CHECKER(DtBaseEmbeddedSystemEncoder, \
        DtEmbeddedSystemRepository, \
        attrName, inspector)

#endif //DtBaseEmbeddedSystemEncoder_H_
```


baseEmitttrDec.h

```

#if DtHLA

#ifndef DtBaseEmitterBeamDecoder_H_
#define DtBaseEmitterBeamDecoder_H_

#include "chooseDefines.h"
#include "hStateDecoder.h"

// This is used to declare attribute decoding functions
// which are members of the class.
#define DtDECLARE_EMITTERBEAM_ATTR_DECODER(attrName) \
    DtDECLARE_ATTR_DECODER(DtEmitterBeamRepository, attrName)

// class DtBaseEmitterBeamDecoder:
//
// Instances of DtBaseEmitterBeamDecoder are

class DT_DLL_VL DtBaseEmitterBeamDecoder : public DtHlaStateDecoder
{
public:

    // default constructor
    DtBaseEmitterBeamDecoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtBaseEmitterBeamDecoder();

    // copy constructor
    DtBaseEmitterBeamDecoder(const DtBaseEmitterBeamDecoder& orig);

    // assignment operator
    DtBaseEmitterBeamDecoder& operator=
        (const DtBaseEmitterBeamDecoder& orig);

public:

    // Create a DtBaseEntityDecoder. Caller is responsible for dele-
    // tion.
    static DtHlaStateDecoder* create(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

protected:

    // Add decoding functions for the attributes.

```

```
// Note: Non-virtual since meant to be called from constructors
to
//      add decoders.
void addMyDecoders();

// static decode<AttributeName> member functions.
DtDECLARE_EMITTERBEAM_ATTR_DECODER(AzimuthCenter);
DtDECLARE_EMITTERBEAM_ATTR_DECODER(AzimuthSweep);
DtDECLARE_EMITTERBEAM_ATTR_DECODER(BeamFunction);
DtDECLARE_EMITTERBEAM_ATTR_DECODER(BeamID);
DtDECLARE_EMITTERBEAM_ATTR_DECODER(BeamParameterIndex);
DtDECLARE_EMITTERBEAM_ATTR_DECODER(ElevationCenter);
DtDECLARE_EMITTERBEAM_ATTR_DECODER(ElevationSweep);
DtDECLARE_EMITTERBEAM_ATTR_DECODER(EmittingSystemID);
DtDECLARE_EMITTERBEAM_ATTR_DECODER(ERP);
DtDECLARE_EMITTERBEAM_ATTR_DECODER(EntityID);
DtDECLARE_EMITTERBEAM_ATTR_DECODER(EventID);
DtDECLARE_EMITTERBEAM_ATTR_DECODER(Frequency);
DtDECLARE_EMITTERBEAM_ATTR_DECODER(FrequencyRange);
DtDECLARE_EMITTERBEAM_ATTR_DECODER(PRF);
DtDECLARE_EMITTERBEAM_ATTR_DECODER(PulseWidth);
DtDECLARE_EMITTERBEAM_ATTR_DECODER(SweepSynch);
};
// These macros are used to define simple attribute decoder func-
tions
// (members of a derived DtHlaStateDecoder that decode
// individual attributes). The simple decoding is just to
// convert the value pulled out of the ahvps to a host type using
// the cast expression castExpr, and pass it to the mutator func-
tion.
#define DtDEFINE_SIMPLE_BASE_EMITTERBEAM_ATTR_DECODER_WITH_CAST( \
    attrName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_ATTR_DECODER_WITH_CAST( \
        DtBaseEmitterBeamDecoder, DtEmitterBeamRepository, \
        attrName, netType, mutator, castExpr)

#define DtDEFINE_SIMPLE_BASE_EMITTERBEAM_ATTR_DECODER( \
    attrName, netType, inspector) \
    DtDEFINE_SIMPLE_BASE_EMITTERBEAM_ATTR_DECODER_WITH_CAST( \
        attrName, netType, inspector, (netType))

#endif //DtBaseEmitterBeamDecoder_H_
```

baseEmitttrEnc.h

```

#if DtHLA

#ifndef DtBaseEmitterBeamEncoder_H_
#define DtBaseEmitterBeamEncoder_H_

#include "chooseDefines.h"
#include "hStateEncoder.h"

// This is used to declare static member functions for determining
// whether or not an attribute needs to be encoded.
#define DtDECLARE_BASE_EMITTERBEAM_ATTR_CHECKER(attrName) \
    DtDECLARE_ATTR_CHECKER(DtEmitterBeamRepository, attrName)

// This is used to declare attribute encoding functions
// which are members of the class.
#define DtDECLARE_BASE_EMITTERBEAM_ATTR_ENCODER(attrName) \
    DtDECLARE_ATTR_ENCODER(DtEmitterBeamRepository, attrName)

// class DtBaseEmitterBeamEncoder:
//
// Instances of DtBaseEmitterBeamEncoder are

class DT_DLL_VL DtBaseEmitterBeamEncoder : public DtHlaStateEncoder
{
public:

    // default constructor
    DtBaseEmitterBeamEncoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtBaseEmitterBeamEncoder();

    // copy constructor
    DtBaseEmitterBeamEncoder(const DtBaseEmitterBeamEncoder& orig);

    // assignment operator
    DtBaseEmitterBeamEncoder& operator=(const DtBaseEmitterBeamEncoder& orig);

public:

    // Create a DtBaseEntityEncoder. Caller is responsible for deletion.
    static DtHlaStateEncoder* create(
        DtExerciseConn* exConn,

```

```
DtObjClassDesc* classDesc);

protected:

    DtDECLARE_BASE_EMITTERBEAM_ATTR_CHECKER(AzimuthCenter);
    DtDECLARE_BASE_EMITTERBEAM_ATTR_CHECKER(AzimuthSweep);
    DtDECLARE_BASE_EMITTERBEAM_ATTR_CHECKER(BeamFunction);
    DtDECLARE_BASE_EMITTERBEAM_ATTR_CHECKER(BeamID);
    DtDECLARE_BASE_EMITTERBEAM_ATTR_CHECKER(BeamParameterIndex);
    DtDECLARE_BASE_EMITTERBEAM_ATTR_CHECKER(ElevationCenter);
    DtDECLARE_BASE_EMITTERBEAM_ATTR_CHECKER(ElevationSweep);
    DtDECLARE_BASE_EMITTERBEAM_ATTR_CHECKER(EmittingSystemID);
    DtDECLARE_BASE_EMITTERBEAM_ATTR_CHECKER(ERP);
    DtDECLARE_BASE_EMITTERBEAM_ATTR_CHECKER(EntityID);
    DtDECLARE_BASE_EMITTERBEAM_ATTR_CHECKER(EventID);
    DtDECLARE_BASE_EMITTERBEAM_ATTR_CHECKER(Frequency);
    DtDECLARE_BASE_EMITTERBEAM_ATTR_CHECKER(FrequencyRange);
    DtDECLARE_BASE_EMITTERBEAM_ATTR_CHECKER(PRF);
    DtDECLARE_BASE_EMITTERBEAM_ATTR_CHECKER(PulseWidth);
    DtDECLARE_BASE_EMITTERBEAM_ATTR_CHECKER(SweepSynch);

    DtDECLARE_BASE_EMITTERBEAM_ATTR_ENCODER(AzimuthCenter);
    DtDECLARE_BASE_EMITTERBEAM_ATTR_ENCODER(AzimuthSweep);
    DtDECLARE_BASE_EMITTERBEAM_ATTR_ENCODER(BeamFunction);
    DtDECLARE_BASE_EMITTERBEAM_ATTR_ENCODER(BeamID);
    DtDECLARE_BASE_EMITTERBEAM_ATTR_ENCODER(BeamParameterIndex);
    DtDECLARE_BASE_EMITTERBEAM_ATTR_ENCODER(ElevationCenter);
    DtDECLARE_BASE_EMITTERBEAM_ATTR_ENCODER(ElevationSweep);
    DtDECLARE_BASE_EMITTERBEAM_ATTR_ENCODER(EmittingSystemID);
    DtDECLARE_BASE_EMITTERBEAM_ATTR_ENCODER(ERP);
    DtDECLARE_BASE_EMITTERBEAM_ATTR_ENCODER(EntityID);
    DtDECLARE_BASE_EMITTERBEAM_ATTR_ENCODER(EventID);
    DtDECLARE_BASE_EMITTERBEAM_ATTR_ENCODER(Frequency);
    DtDECLARE_BASE_EMITTERBEAM_ATTR_ENCODER(FrequencyRange);
    DtDECLARE_BASE_EMITTERBEAM_ATTR_ENCODER(PRF);
    DtDECLARE_BASE_EMITTERBEAM_ATTR_ENCODER(PulseWidth);
    DtDECLARE_BASE_EMITTERBEAM_ATTR_ENCODER(SweepSynch);
};
// This macro is used to define simple attribute encoder functions
// (members of a derived DtHlaStateEncoder that encode
// individual attributes). The simple encoding is just to implicitly
// convert the value returned by the inspector to a netType,
// and add it to the ahvps.
#define DtDEFINE_SIMPLE_BASE_EMITTERBEAM_ATTR_ENCODER( \
    attrName, netType, inspector) \
    DtDEFINE_SIMPLE_ATTR_ENCODER(DtBaseEmitterBeamEncoder, \
        DtEmitterBeamRepository, \
        attrName, netType, inspector)
```

```
// This macro is used to define simple attribute checker functions
// (members of a derived DtHlaStateEncoder that check whether
// a particular attribute is out of date. The simple implementation
// is to just compare the values returned by the two StateReps'
// inspector functions.
#define DtDEFINE_SIMPLE_BASE_EMITTERBEAM_ATTR_CHECKER( \
    attrName, inspector) \
    DtDEFINE_SIMPLE_ATTR_CHECKER(DtBaseEmitterBeamEncoder, \
        DtEmitterBeamRepository, \
        attrName, inspector)

#endif    //DtBaseEmitterBeamEncoder_H_
```

baseEntDec.h

```
#if DtHLA

#include "hStateDecoder.h"

// This is used to declare attribute decoding functions
// which are members of the class.
#define DtDECLARE_ENTITY_ATTR_DECODER(attrName) \
    DtDECLARE_ATTR_DECODER(DtEntityStateRepository, attrName)

// This is used to declare attribute decoding functions
// which are members of the class.
#define DtDECLAREAggregate_ATTR_DECODER(attrName) \
    DtDECLARE_ATTR_DECODER(DtAggregateStateRepository, attrName)

// This is used to declare attribute decoding functions
// which are members of the class.
#define DtDECLARE_BASE_ENTITY_ATTR_DECODER(attrName) \
    DtDECLARE_ATTR_DECODER(DtBaseEntityStateRepository, attrName)

// class DtBaseEntityDecoder:
//
// Instances of DtBaseEntityDecoder are used to decode a
// base entity state message into a state repository in an HLA
// (as opposed to DIS) environment.

class DT_DLL_VL DtBaseEntityDecoder : public DtHlaStateDecoder
{
public:

    // constructor
    DtBaseEntityDecoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtBaseEntityDecoder();

    // copy constructor
    DtBaseEntityDecoder(const DtBaseEntityDecoder& orig);

    // assignment operator
    DtBaseEntityDecoder& operator=(const DtBaseEntityDecoder& orig);

    // Virtual function override. Return a new instance of this
    class.
    virtual DtHlaStateDecoder* clone() const;
};

#endif
```

```

public:

    // Create a DtBaseEntityDecoder. Caller is responsible for deletion.
    static DtHlaStateDecoder* create(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

protected:

    // Add decoding functions for the attributes.
    // Note: Non-virtual since meant to be called from constructors to
    //       add decoders.
    void addMyDecoders();

    // virtual function override that sets some timestamping info,
    // then calls
    // down to the base class to perform decoding of attributes.
    virtual void decode(const DtStateMsg& msg,
        DtStateRepository* stateRep) const;

    // static decode<AttributeName> member functions.
    DtDECLARE_BASE_ENTITY_ATTR_DECODER(AccelerationVector);
    DtDECLARE_BASE_ENTITY_ATTR_DECODER(AngularVelocityVector);
    DtDECLARE_BASE_ENTITY_ATTR_DECODER(DRAAlgorithm);
    DtDECLARE_BASE_ENTITY_ATTR_DECODER(EntityType);
#ifdef !DtrPR016
    DtDECLARE_BASE_ENTITY_ATTR_DECODER(EntityID);
#else
    DtDECLARE_BASE_ENTITY_ATTR_DECODER(FederateID);
#endif
    DtDECLARE_BASE_ENTITY_ATTR_DECODER(IsFrozen);
    DtDECLARE_BASE_ENTITY_ATTR_DECODER(Orientation);
    DtDECLARE_BASE_ENTITY_ATTR_DECODER(Position);
    DtDECLARE_BASE_ENTITY_ATTR_DECODER(VelocityVector);
};

// These macros are used to define simple attribute decoder functions
// (members of a derived DtHlaStateDecoder that decode
// individual attributes). The simple decoding is just to
// convert the value pulled out of the ahvps to a host type using
// the cast expression castExpr, and pass it to the mutator function.
#define DtDEFINE_SIMPLE_BASE_ENTITY_ATTR_DECODER_WITH_CAST( \
    attrName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_ATTR_DECODER_WITH_CAST( \
        DtBaseEntityDecoder, DtBaseEntityStateRepository, \
        attrName, netType, mutator, castExpr)

```

```
#define DtDEFINE_SIMPLE_BASE_ENTITY_ATTR_DECODER( \
    attrName, netType, inspector) \
    DtDEFINE_SIMPLE_BASE_ENTITY_ATTR_DECODER_WITH_CAST( \
        attrName, netType, inspector, (netType))

#endif
```


baseEntEnc.h

```

#if DtHLA

#include "hStateEncoder.h"

// This is used to declare static member functions for determining
// whether or not an attribute needs to be encoded.
#define DtDECLARE_ENTITY_ATTR_CHECKER(attrName) \
    DtDECLARE_ATTR_CHECKER(DtEntityStateRepository, attrName)

// This is used to declare attribute encoding functions
// which are members of the class.
#define DtDECLARE_ENTITY_ATTR_ENCODER(attrName) \
    DtDECLARE_ATTR_ENCODER(DtEntityStateRepository, attrName)

// This is used to declare static member functions for determining
// whether or not an attribute needs to be encoded.
#define DtDECLAREAggregate_ATTR_CHECKER(attrName) \
    DtDECLARE_ATTR_CHECKER(DtAggregateStateRepository, attrName)

// This is used to declare attribute encoding functions
// which are members of the class.
#define DtDECLAREAggregate_ATTR_ENCODER(attrName) \
    DtDECLARE_ATTR_ENCODER(DtAggregateStateRepository, attrName)

// This is used to declare static member functions for determining
// whether or not an attribute needs to be encoded.
#define DtDECLARE_BASE_ENTITY_ATTR_CHECKER(attrName) \
    DtDECLARE_ATTR_CHECKER(DtBaseEntityStateRepository, attrName)

// This is used to declare attribute encoding functions
// which are members of the class.
#define DtDECLARE_BASE_ENTITY_ATTR_ENCODER(attrName) \
    DtDECLARE_ATTR_ENCODER(DtBaseEntityStateRepository, attrName)

// class DtBaseEntityEncoder:
//
// Instances of DtBaseEntityEncoder are used to encode a
// base entity state repository into a state message in an HLA
// (as opposed to DIS) environment.

class DT_DLL_VL DtBaseEntityEncoder : public DtHlaStateEncoder
{
public:

    // constructor
    DtBaseEntityEncoder(
        DtExerciseConn* exConn,

```

```
DtObjClassDesc* classDesc);

// destructor
virtual ~DtBaseEntityEncoder();

// copy constructor
DtBaseEntityEncoder(const DtBaseEntityEncoder& orig);

// assignment operator
DtBaseEntityEncoder& operator=(const DtBaseEntityEncoder& orig);

public:

    // Create a DtBaseEntityEncoder. Caller is responsible for deletion.
    static DtHlaStateEncoder* create(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // Get whether or not rotational or translational thresholds have
    // been exceeded.
    static DtBoolean tpsiChanged(
        const DtBaseEntityStateRepository& stateRep,
        const DtBaseEntityStateRepository& asSeenByRemote);

protected:

    // static need<Attribute> member functions.
    DtDECLARE_BASE_ENTITY_ATTR_CHECKER(AccelerationVector);
    DtDECLARE_BASE_ENTITY_ATTR_CHECKER(AngularVelocityVector);
    DtDECLARE_BASE_ENTITY_ATTR_CHECKER(DRAlgorithm);
    DtDECLARE_BASE_ENTITY_ATTR_CHECKER(EntityType);
    #if !DtrPR016
        DtDECLARE_BASE_ENTITY_ATTR_CHECKER(EntityID);
    #else
        DtDECLARE_BASE_ENTITY_ATTR_CHECKER(FederateID);
    #endif
    DtDECLARE_BASE_ENTITY_ATTR_CHECKER(IsFrozen);
    DtDECLARE_BASE_ENTITY_ATTR_CHECKER(Orientation);
    DtDECLARE_BASE_ENTITY_ATTR_CHECKER(Position);
    DtDECLARE_BASE_ENTITY_ATTR_CHECKER(VelocityVector);

    // static encode<Attribute> member functions.
    DtDECLARE_BASE_ENTITY_ATTR_ENCODER(AccelerationVector);
    DtDECLARE_BASE_ENTITY_ATTR_ENCODER(AngularVelocityVector);
    DtDECLARE_BASE_ENTITY_ATTR_ENCODER(DRAlgorithm);
    DtDECLARE_BASE_ENTITY_ATTR_ENCODER(EntityType);
    #if !DtrPR016
        DtDECLARE_BASE_ENTITY_ATTR_ENCODER(EntityID);
    #else
```

```

    DtDECLARE_BASE_ENTITY_ATTR_ENCODER(FederateID);
#endif
    DtDECLARE_BASE_ENTITY_ATTR_ENCODER(IsFrozen);
    DtDECLARE_BASE_ENTITY_ATTR_ENCODER(Orientation);
    DtDECLARE_BASE_ENTITY_ATTR_ENCODER(Position);
    DtDECLARE_BASE_ENTITY_ATTR_ENCODER(VelocityVector);
};
// This macro is used to define simple attribute encoder functions
// (members of a derived DtHlaStateEncoder that encode
// individual attributes). The simple encoding is just to implicitly
// convert the value returned by the inspector to a netType,
// and add it to the ahvps.
#define DtDEFINE_SIMPLE_BASE_ENTITY_ATTR_ENCODER( \
    attrName, netType, inspector) \
    DtDEFINE_SIMPLE_ATTR_ENCODER(DtBaseEntityEncoder, \
        DtBaseEntityStateRepository, \
        attrName, netType, inspector)

// This macro is used to define simple attribute checker functions
// (members of a derived DtHlaStateEncoder that check whether
// a particular attribute is out of date. The simple implementation
// is to just compare the values returned by the two StateRepD

```

baseEntitySR.h

```
#include "stateRep.h"
#include "hostStructs.h"
#include "mathTypes.h"
#include "tStampType.h"
#include "thresholder.h"

// class DtBaseEntityStateRepository:
//
// Instances of DtBaseEntityStateRepository are used to maintain
// basic state
// information needed by all kinds of entities.

class DT_DLL_VL DtBaseEntityStateRepository : public DtStateRepository
{
public:

    // default constructor
    DtBaseEntityStateRepository();

    // destructor
    virtual ~DtBaseEntityStateRepository();

    // copy constructor
    DtBaseEntityStateRepository(const DtBaseEntityStateRepository&
orig);

    // assignment operator
    DtBaseEntityStateRepository& operator=(
        const DtBaseEntityStateRepository& orig);

    // return an empty object of the same type as this
    virtual DtStateRepository* clone() const;

    // Set/Get flag indicating whether we're using absolute times-
    tamps for dead
    // reckoning. This flag will be set to DtAbsolute by the decoders
    only if
    // incoming state messages indicate that the sender is using
    absolute time
    // stamps, AND our local exercise connection indicates that we
    are using
    // absolute timestamping.
    void setTimeStampType(DtTimeStampType type);
    DtTimeStampType timeStampType() const;
```

```

    // Set/Get the location of the center of mass in world coordi-
nates.
    virtual void setLocation(DtConstVector location,
        DtTime timeNow = DtGetSimTime());
    virtual DtConstVector location(DtTime timeNow = DtGetSimTime())
const;
    // Return the value last passed to setLocation, regardless of
    // whether we have an approximator.
    virtual DtConstVector lastSetLocation() const;

    // Set/Get the velocity.
    virtual void setVelocity(DtConstVector Velocity,
        DtTime timeNow = DtGetSimTime());
    virtual DtConstVector velocity(
        DtTime timeNow = DtGetSimTime()) const;
    // Return the value last passed to setVelocity, regardless of
    // whether we have an approximator.
    virtual DtConstVector lastSetVelocity() const;

    // Set/Get the acceleration.
    virtual void setAcceleration(DtConstVector Acceleration,
        DtTime timeNow = DtGetSimTime());
    virtual DtConstVector acceleration(
        DtTime timeNow = DtGetSimTime()) const;
    // Return the value last passed to setAcceleration, regardless of
    // whether we have an approximator.
    virtual DtConstVector lastSetAcceleration() const;

    // Set/Get the orientation.
    virtual void setOrientation(const DtTaitBryan& orient,
        DtTime timeNow = DtGetSimTime());
    virtual const DtTaitBryan& orientation(
        DtTime timeNow = DtGetSimTime()) const;
    // Return the value last passed to setOrientation, regardless of
    // whether we have an approximator.
    virtual const DtTaitBryan& lastSetOrientation() const;

    // Set/Get the rotational velocity.
    virtual void setRotationalVelocity(DtConstVector BodyRates,
        DtTime timeNow = DtGetSimTime());
    virtual DtConstVector rotationalVelocity(
        DtTime timeNow = DtGetSimTime()) const;
    // Return the value last passed to setRotationalVelocity, regard-
less of
    // whether we have an approximator.
    virtual DtConstVector lastSetRotationalVelocity() const;

    // Get the body to geocentric coordinate matrix.
    virtual DtConstDcm bodyToGeoc(
        DtTime timeNow = DtGetSimTime()) const;

```

```
// Set/Get the dead reckoning algorithm.
virtual void setAlgorithm(DtDeadReckonTypes alg);
virtual DtDeadReckonTypes algorithm() const;

// Set/Get the entity's identifier.
virtual void setEntityId(const DtEntityIdentifier& id);
virtual const DtEntityIdentifier& entityId() const;

// Set/Get the entity type.
virtual void setEntityType(const DtEntityType& type);
virtual const DtEntityType& entityType() const;

#if DtHLA
// Set/Get the federate ID (part of entity ID)
virtual void setFederateId(const DtFederateId& id);
virtual const DtFederateId& federateId() const;
#endif

#if DtDISVERS != 3
// Set/Get whether frozen or not.
virtual void setFrozen(DtBoolean onOff);
virtual DtBoolean frozen() const;
#endif

// Get and set the approximator used to provide approximated
// position
// and orientation information for the entity. If your approxi-
// mator
// is a DtDeadReckoner, use useDeadReckoner instead of setApprox-
// imator.
// If it's a smoother, use useSmoother. Caller
// of setApproximator has the responsibility to delete approx
// after the StateRep no longer needs it.
virtual void setApproximator(DtApproximator* approx);
virtual DtApproximator* approximator() const;

// Tells the ESR to use dr as its approximator, and to notify it
// when DR algorithm changes. Caller is responsible for deleting
dr,
// but only after the StateRep no longer needs it.
virtual void useDeadReckoner(DtDeadReckoner* dr);

// Create and use a new (private) dead-reckoner as our approxi-
// mator
virtual void useDeadReckoner();

// If the ESR is using a dead reckoner, returns it. Else returns
NULL.
virtual DtDeadReckoner* usingDeadReckoner();
```

```
// Tells the ESR to use sm as its approximator, and to notify it
// when DR algorithm changes. Caller is responsible for deleting
sm,
// but only after the StateRep no longer needs it.
virtual void useSmoother(DtSmoother* sm);

// Create and use a new (private) smoother as our approximator
virtual void useSmoother();

// If the ESR is using a smoother, returns it. Else returns NULL.
virtual DtSmoother* usingSmoother();

// Returns the SR's threshold, which contains thresholds needed
// for encoding
virtual DtThreshold* threshold();

// Const version
virtual const DtThreshold& threshold() const;

// Print the state data.
virtual void printData() const;
```

protected:

```
// Cast off const-ness for trivial internal change.
// Non-virtual since can't override on return type.
DtBaseEntityStateRepository* self() const;
```

protected:

```
// holds what's currently being used as our approximator
DtApproximator* myApproximator;

// holds what was most recently passed to useDeadReckoner
DtDeadReckoner* myDeadReckoner;

// holds what was most recently passed to useSmoother
DtSmoother* mySmoother;

// Created and deleted by this ESR if needed
DtDeadReckoner* myDfltDeadReckoner;
DtSmoother* myDfltSmoother;

DtVector myLocation;
DtVector myVelocity;
DtVector myAcceleration;

DtTaitBryan myOrientation;
DtVector myRotationalVelocity;
```

```
DtDeadReckonTypes myAlgorithm;

DtDcm myBodyToGeoc;
DtBoolean myBodyToGeocDirty;

DtTimeStampType myTimeStampType;

DtEntityIdentifier myEntityId;
#if DtHLA
    DtFederateId myFederateId;
#endif
    DtEntityType myEntityType;
#if DtDISVERS != 3
    DtBoolean myFrozen;
#endif

DtThresholder myThresholder;

protected:

    // Used by DtBaseEntityDecoder to support absolute DR in HLA.
    friend class DtBaseEntityDecoder;
    virtual void setTimeForSets(DtTime time);
    virtual DtTime timeForSets() const;
    DtTime myTimeForSets;
};
#if DtHLA
#endif
```


baseExConn.h

```

#include "vlTime.h"
#include "vlList.h"
#include "chooseDefines.h"

// class DtBaseExerciseConn:
//
// DtBaseExerciseConn is an abstract base class. It provides an
// application's interface or connection to a network.

typedef void (*DtPostDrainCb)(void* usr);

class DT_DLL_VL DtBaseExerciseConn
{
public:

    // destructor
    virtual ~DtBaseExerciseConn();

    // assignment operator
    DtBaseExerciseConn& operator=(const DtBaseExerciseConn& orig);

    // Send a state message to the network.
    virtual void send(const DtStateMsg& msg) = 0;

    // Send an interaction to the network.
    virtual void send(const DtInteraction& msg) = 0;

    // Send a state message to the network which has been
    // time stamped with the current time.
    virtual void sendStamped(const DtStateMsg& msg) = 0;

    // Send an interaction to the network which has been
    // time stamped with the current time.
    virtual void sendStamped(const DtInteraction& msg) = 0;

    // Read and process messages from the network until either no
more
    // messages, or for timeout seconds, whichever happens first.
    // Will read nReadPerTimeCheck messages at a time before checking
    // for timeout.
    // Returns number of incoming events which have been processed.
    virtual int drainInput(DtTime timeout=-1,
        int nReadPerTimeCheck=100) = 0;

    // add/remove callbacks to be executed from within drainInput
    // right before it returns
    virtual void addPostDrainCallback(DtPostDrainCb cb, void *usr);

```

```
    virtual void removePostDrainCallback(DtPostDrainCb cb, void
*usr);
```

protected:

```
    // Constructors are protected since this is an abstract base
class.
```

```
    // default constructor
    DtBaseExerciseConn();
```

```
    // copy constructor
    DtBaseExerciseConn(const DtBaseExerciseConn& orig);
```

```
    // should be called by derived class' drainInputs.
    virtual void executePostDrainCbs();
```

protected:

```
    DtList myPostDrainCbs;
};
```

baseHashKey.h

```

#include "MachineTypes.h"
#include "str.h"

// class DtBaseHashKey:
//
// DtBaseHashKey is an abstract base class for hashkeys.

class DT_DLL_VLUTIL DtBaseHashKey
{
public:

    // default constructor
    DtBaseHashKey();

    // destructor
    virtual ~DtBaseHashKey();

    // copy constructor
    DtBaseHashKey(const DtBaseHashKey& orig);

    // assignment operator
    DtBaseHashKey& operator=(const DtBaseHashKey& orig);

    // equality operator
    virtual DtBoolean operator==(const DtBaseHashKey& other) const;

    // equality operator
    virtual DtBoolean operator!=(const DtBaseHashKey& other) const;

public:

    // get the hashed value for this key
    virtual DtU32 hashValue() const;

    // make a copy of this hash key
    virtual DtBaseHashKey* clone() const = 0;

    // get the string representation of this key
    virtual const DtString& stringRep() const = 0;

protected:

    // compute the hashed value for this key
    virtual DtU32 computeHashValue() = 0;

protected:

```

```
DtBoolean myHashValueIsComputed;  
DtU32 myHashValue;  
};
```

baseMsg.h

```

#include "stdlib.h"
#include "stdio.h"
#include "MachineTypes.h"

#define DtUSE_INTERNAL_BUFFER_REP NULL
#define DtUSE_INTERNAL_BUFFER (DtBufferPtr(NULL))

class DT_DLL_VLUTIL DtBaseMsg
{
public:

    // Destructor
    virtual ~DtBaseMsg();

    // Copy constructor
    DtBaseMsg(const DtBaseMsg& msg,
              DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // Assignment
    DtBaseMsg &operator=(const DtBaseMsg& msg);

    // Get network representation
    virtual const char* netRep() const;

    // Get size of message
    virtual int length() const;

protected:

    // Constructor
    // Object is not fully constructed until initMsg is called.
    DtBaseMsg();

    void initMsg(int size, DtBufferPtr bufferPtr);
    void initMsg(const void *initial,
                 int size, DtBufferPtr bufferPtr);

    // Changes the size of the message. If we are using an external
    // buffer, assumes that the full buffer size is big enough to
    // accommodate the new msg size. If using an internal buffer,
    // the call may result in allocating a new internal buffer, and
    copying
    // data from the old to the new buffer.
    virtual void changeSize(int size);

    // Inserts numBytes zeroed-out bytes at the indicated offset
    // into the net rep. Uses changeSize to increase the size of the

```

```
// buffer.
virtual void insertBytes(int offset, int numBytes);

// Inserts numBytes zeroed-out bytes at the indicated address.
// Assumes that address falls within the net rep.
// Uses changeSize to increase the size of the buffer.
virtual void insertBytes(void *address, int numBytes);

// Deletes numBytes bytes at the indicated offset in the net rep.
// Uses change size to decrease the size of the buffer.
virtual void deleteBytes(int offset, int numBytes);

// Deletes numBytes bytes starting at the indicated address.
// Assumes that address falls within the net rep.
// Uses changeSize to decrease the size of the buffer.
virtual void deleteBytes(void *address, int numBytes);

// zeros out all bytes in the network representation
virtual void clearAll();

// Sets myInternalBuff to point to a malloc'ed buffer of
// size bytes. Sets myIntBuffSize to size. Returns myInternal-
Buff.
// Size must be a positive number.
virtual void *newInternalBuffer(int size);

// Called by changeSize. Derived versions can provide
// definitions that set the size field in network rep.
virtual void sizeChanged(int size) = 0;

protected:

    void *myNetMessage;
    void *myInternalBuff;
    int myIntBuffSize;
    int myMsgSize;
};
```

baseReflAgg.h

```

#if DtDISVERS >= 5

#include "vList.h"
#include "mathTypes.h"
#include "hostStructs.h"
#include "stateMsg.h"

#include "chooseDefines.h"

// class DtBaseReflectedAggregate:
//
// DtBaseReflectedAggregate is an abstract base class. It is used
// to
// maintain the representation of a simulation aggregate reflected
// across a network (i.e. this is not a full simulation running
// on this machine).

class DT_DLL_VL DtBaseReflectedAggregate : public DtListedObj
{
public:

    // destructor
    virtual ~DtBaseReflectedAggregate();

    // assignment operator
    DtBaseReflectedAggregate& operator=(const DtBaseReflectedAggregate& orig);

    // Get the aggregate's state repository.
    virtual DtAggregateStateRepository* aggregateStateRep() const =
0;

    // Get the aggregate's identifier.
    virtual const DtObjectId& id() const;

protected:

    // Constructors are protected since this is an abstract base
    class.

    // constructor
    DtBaseReflectedAggregate(const DtObjectId& id);

    // copy constructor
    DtBaseReflectedAggregate(const DtBaseReflectedAggregate& orig);

```

protected:

 DtObjectId myObjectId;

};

#endif

baseRefEnt.h

```

#include "vList.h"
#include "mathTypes.h"
#include "hostStructs.h"
#include "stateMsg.h"

#include "vlprint.h"          // Andy - for a DtWarn() message

// class DtBaseReflectedEntity:
//
// DtBaseReflectedEntity is an abstract base class. It is used to
// maintain the representation of a simulation entity reflected
// across a network (i.e. this is not an full simulation running
// on this machine).

class DT_DLL_VL DtBaseReflectedEntity : public DtListedObj
{
public:

    // destructor
    virtual ~DtBaseReflectedEntity();

    // assignment operator
    DtBaseReflectedEntity& operator=(const DtBaseReflectedEntity&
orig);

    // Get the entity's state repository.
    virtual DtEntityStateRepository* entityStateRep() const = 0;

    // Get the entity's identifier.
    virtual const DtObjectId& id() const;

protected:

    // Constructors are protected since this is an abstract base
    class.

    // constructor
    DtBaseReflectedEntity(const DtObjectId& id);

    // copy constructor
    DtBaseReflectedEntity(const DtBaseReflectedEntity& orig);

protected:

    DtObjectId myObjectId;
};

```

beamAntRec.h

```
#include "disEnums.h"
#include "mathTypes.h"
#include "antPatRec.h"

// class DtBeamAntennaPatternRecord:
//
// Instances of DtBeamAntennaPatternRecord are beam antenna pattern
// records
// used in conjunction with DtRadioTransmitterStateRepository (TSR).
// A DtAntennaPatternRecord obtained using the TSR's antennaPat-
// tern()
// inspector can be used to initialize a DtBeamAntennaPatternRecord,
// whose
// inspectors you can then use to view the data (assuming the Anten-
// naPattern
// Record really contained BeamAntennaPattern data.
// Similarly, a DtBeamAntennaPatternRecord can be filled out and
// passed to
// TSR's setAntennaPattern.

class DT_DLL_VL DtBeamAntennaPatternRecord : public DtAntennaPat-
ternRecord
{
public:

    // default constructor
    DtBeamAntennaPatternRecord();

    // copy constructor
    DtBeamAntennaPatternRecord(const DtBeamAntennaPatternRecord&
orig);

    // destructor
    virtual ~DtBeamAntennaPatternRecord();

    // assignment operator
    DtBeamAntennaPatternRecord& operator=(
        const DtBeamAntennaPatternRecord& orig);

    // Initialize from DtAntennaPatternRecord
    DtBeamAntennaPatternRecord(const DtAntennaPatternRecord& orig);

    // Set/Get the beam direction
    virtual void setAngles(const DtTaitBryan &angles);
    virtual const DtTaitBryan &angles() const;

    // Set/Get the azimuth bandwidth and elevation
```

```

virtual void setBeamwidth(DtFloat32 azimuth,
    DtFloat32 elevation);

// Get the azimuth beam bandwidth
virtual DtFloat32 azimuthBeamwidth() const;

// Get the elevation beam width
virtual DtFloat32 elevationBeamwidth() const;

// Set/Get the beam reference system
virtual void setReferenceSystem(const DtReferenceSys sys);
virtual DtReferenceSys referenceSystem() const;

// Set electric field
virtual void setElectricField(DtFloat32 z, DtFloat32 x);

// Get electric field Z
virtual DtFloat32 electricFieldZ() const;

// Get electric field X
virtual DtFloat32 electricFieldX() const;

// Set/Get the phase
virtual void setPhase(DtFloat32 phase);
virtual DtFloat32 phase() const;

// print the class data
virtual void printData() const;

// Returns a clone of the instance
virtual DtAntennaPatternRecord* clone() const;

// Set from network representation
virtual void setFromNet(const void* net, int size);

// Returns network representation, fills size with size of
netRep.
virtual const void* netRep(int* size = NULL) const;

// Returns size of network representation.
virtual int netSize() const;

// Set from network representation
virtual void setFromNetBeam(const DtNetBeamAntennaPatternRecord&
net);

// Also sets from network representation. This function is here
for
// backwards compatibility with pre-3.3 releases.

```

```
virtual void setFromNet(const DtNetBeamAntennaPatternRecord&
net);

// Returns network representation
virtual const DtNetBeamAntennaPatternRecord& netBeam() const;

// Also returns network representation. This function is here
for
// backwards compatibility with pre-3.3 releases.
virtual const DtNetBeamAntennaPatternRecord& networkRepresenta-
tion() const;

protected:
    DtTaitBryan    myBeamDirection;
    DtFloat32      myAzimuthBandwidth;
    DtFloat32      myElevationBeamwidth;
    DtReferenceSys myReferenceSystem;
    DtFloat32      myEz;
    DtFloat32      myEx;
    DtFloat32      myPhase;
};
```

cartView.h

```

#include "LibMatrix.h"
#include "chooseDefines.h"
// class DtCartesianView:
//
// Instances of DtCartesianView are used to create a Cartesian
// coordinate view of the geocentric coordinates in an
// Entity State Repository.

class DT_DLL_VL DtCartesianView
{
public:

    // constructor
    DtCartesianView(DtBaseEntityStateRepository* ksr);

    DtCartesianView(DtBaseEntityStateRepository* ksr,
        DtConstVector originInGeoc,
        DtConstDcm geocToLocalRot);

    // destructor
    virtual ~DtCartesianView();

    // copy constructor
    DtCartesianView(const DtCartesianView& orig);

    // assignment operator
    DtCartesianView& operator=(const DtCartesianView& orig);

    // set the origin of the local system in geocentric coords
    virtual void setOffset(DtConstVector originInGeoc);

    // set the rotation matrix that converts from geocentric to local
    coords
    virtual void setRotation(DtConstDcm geocToLocalRot);

    // Set/Get the location in local coordinates.
    virtual void setLocation(DtConstVector Location);
    virtual DtConstVector location() const;

    // Set/Get the velocity.
    virtual void setVelocity(DtConstVector Velocity);
    virtual DtConstVector velocity() const;

    // Set/Get the orientation.
    virtual void setOrientation(const DtTaitBryan& Orient);
    virtual const DtTaitBryan& orientation() const;

```

```
// Get the body to local coordinate matrix. This is just
// a matrix representation of the euler angles returned by
// orientation().
virtual DtConstDcm bodyToLocal() const;

// Set/Get the acceleration.
virtual void setAcceleration(DtConstVector Acceleration);
virtual DtConstVector acceleration() const;

// Set/Get the rotational velocity.
virtual void setRotationalVelocity(DtConstVector BodyRates);
virtual DtConstVector rotationalVelocity() const;

protected:

    // Cast off const-ness for trivial internal change.
    // Non-virtual since can't override on return type.
    DtCartesianView* self() const;

protected:

    DtBaseEntityStateRepository* myESR;
    DtCoordTransform myGeocToLocal;
    DtCoordTransform myLocalToGeoc;

    // used as return values by inspectors. Do not necessarily
    // hold current state.
    DtVector myLocalLoc;
    DtVector myLocalVel;
    DtVector myLocalAccel;
    DtTaitBryan myLocalOrient;
    DtDcm myBodyToLocal;
};
```

ClientSocket.h

```

#include "NetTypes.h"
#include "Socket.h"
#include "NetSocket.h"
#include "cliServer.h"

/*
 * ClientSocket uses the vrlink packet server to read and write
 * packets
 * The destination address (incl. broadcast vs. internal-mode) are
 * controlled by the packet server.
 *
 * Constructor differences with respect to NetSocket:
 *   the destAddr is determined by the packet server (aka vrlink-
 * daemon)
 *   all ClientSockets are presently ReadWrite
 */

#if DT_HAVE_DAEMON
class DT_DLL_VLUTIL DtCsFilterTest : public DtFilterTest
{
public:
    DtCsFilterTest(int fd, DtU8 filterType, DtU32 clientId);
    virtual ~DtCsFilterTest();

    void setOperator(DtTestOperator op);
    void extendTest(int n, DtU32 *values);
    void reduceTest(int n, DtU32 *vals);
    // No-op. Real test done on server
    DtBoolean evalTest(char *, int , DtFilterType){return DtTrue;}

private:
    DtCsFilterMsg    filterMsg;
};
#endif

#define DtCsFlagsDefault (DtCsFlagsReadWrite | DtCsFlagsServer-
ChooseDropMode )

#define DtCsQDefaultSize DtCsUseDefaultSize

class DT_DLL_VLUTIL DtClientSocket : public DtSocket
{
public:

    DtClientSocket(int port, int recvQSize =
DtCsQDefaultSize, unsigned short csFlags = DtCsFlagsDefault);

```

```
DtClientSocket(int port, DtInetAddr destAddr, int recvQSize =
DtCsQDefaultSize, unsigned short csFlags = DtCsFlagsDefault);

~DtClientSocket();

int sendTo(caddr_t packet, size_t size, DtInetAddr addr);

int DtRecv(caddr_t, size_t); /* this is slower as it does a
memcpy() */
int DtRecv(caddr_t *buffptr, int *status=NULL);

int listenToMulticastGroup(DtInetAddr multicastAddr);
int dropMulticastGroup(DtInetAddr multicastAddr);
int setMcastTtl(int ttl) { return netSock->setMcastTtl(ttl); }

void setUnbundling(int onOff);
/* int isUnbundling(); is not currently implemented. The answer
   depends on the state of the Packet Server. */

DtInetAddr lastSourceInetAddr() { return sourceAddr; }
void DtBecomeNonBlocking(void);

int readFd()           { return recvFd;   }
int writeFd()          { return sendFd;   }
int isOk()             { return ok;       }
DtBoolean isInternal();

void setPsMsgTimeout(int seconds);

void notifyMe();
void disconnectFromServer();
void setPacketServerInternal();
void setPacketServerExternal();
void pingClients();
void testClients();
void terminatePacketServer();
void requestServerState();
void requestServerCStates();
void setPacketServerUnbundling(int onOff);

#if DT_HAVE_DAEMON
DtFilterTest *exerciseId()      { return exercise; }
DtFilterTest *protocolVersion() { return version; }
DtFilterTest *pduKind()        { return kind; }
DtFilterTest *ipAddr()         { return addr; }
#endif

static DtTime psMsgTimeout;

protected:
```



```

    void init(int port, DtInetAddr destAddr, int recvQSize,
              unsigned short csFlags);

    int timedRecv(caddr_t buf, size_t size, char *psKind);
    void drainNotifySocket();
    void checkGoAway();

#ifdef DT_HAVE_DAEMON
    DtShmMsgQueueReader *reader;
#endif

    DtNetSocket *netSock;
    DtBoolean notifyPending;

    DtBoolean ok;
    int recvFd;
    int sendFd;
    long clientId;

private:

    void processControlMsg(DtCsMsg *msg);
    DtCsMsg *getPsMsg (DtCsMsg *buf, char desiredKind);
    void cleanup(void);

#ifdef DT_HAVE_DAEMON
    DtCsFilterTest *exercise;
    DtCsFilterTest *version;
    DtCsFilterTest *kind;
    DtCsFilterTest *addr;
#endif

    DtInetAddr destAddr;
    char clientSocketPath[DtCsClientPathSize];
    int serverPort;
    DtBoolean internal;
    DtInetAddr sourceAddr;
    DtNetFloat64 pktBuff8[(DtMaxPacketSize + sizeof(DtCsServer-
Packet) + 7)
                        / sizeof(DtNetFloat64) ];
    char *shmAddr;
#ifdef DT_HAVE_DAEMON
    DtCsControlReqMsg *controlMsg;
#endif
};

```

collidePdu.h

```
#if !DtHLA

#include "hostStructs.h"
#include "interaction.h"
#include "packets/collisionP.h"
#include "mathTypes.h"
#include "exerciseConn.h"
#include "disEnums.h"

#include "chooseDefines.h"

const int DtCollisionStaticSize = 60;
typedef DtCollisionPdu DtCollisionInteraction;
typedef void (*DtCollisionInteractionCb)
    (DtCollisionInteraction *inter, void *usr);

class DT_DLL_VL DtCollisionPdu : public DtInteraction
{
public:

//***** STANDARD CONSTRUCTORS *****

    DtCollisionPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtCollisionPdu(const DtNetCollisionPdu *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    static DtPdu *create(const DtNetPacket *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    static void addCallback(DtExerciseConn *conn,
        DtCollisionInteractionCb cb, void *usr);
    static void removeCallback(DtExerciseConn *conn,
        DtCollisionInteractionCb cb, void *usr);

//***** INSPECTOR FUNCTIONS *****

    const DtEntityIdentifier &issueId() const;
    const DtEntityIdentifier &collideId() const;
    const DtEventID &eventId() const;
    #if DtDISVERS >= 5
        DtCollisionType collisionType() const;
    #endif
    DtConstVector velocity() const;
    DtFloat32 mass() const;
    DtConstVector entityCoords() const;

//***** MUTATOR FUNCTIONS *****
```

```
    void setIssueId(const DtEntityIdentifier &id);
    void setCollideId(const DtEntityIdentifier &id);
    void setEventId(const DtEventID &id);
#if DtDISVERS >= 5
    void setCollisionType(DtCollisionType t);
#endif
    void setVelocity(DtConstVector veloc);
    void setMass(DtFloat32 mass);
    void setEntityCoords(DtConstVector coords);

    void printData() const;

protected:
    DtPduKind internalGetPduKind() const {return DtCollisionPdu-
Kind;}
    DtEntityIdentifier iss;
    DtEntityIdentifier col;
    DtEventID event;
    DtVector vel, ent;
};
#endif
```

collisElasPdu.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "hostStructs.h"
#include "pdu.h"
#include "mathTypes.h"
#include "exerciseConn.h"

#include "packets/collisElasP.h"

// class DtCollisionElasticPdu:
//
// Instances of DtCollisionElasticPdu are used to send simulated
// elastic
// collision information.

// DtCollisionElasticPdu callback function signature.
typedef void (*DtCollisionElasticPduCb)(DtCollisionElasticPdu* pdu,
void* usr);

class DT_DLL_VL DtCollisionElasticPdu : public DtPdu
{
public:
    // Constructors
    DtCollisionElasticPdu(DtBufferPtr buffer =
DtUSE_INTERNAL_BUFFER);
    DtCollisionElasticPdu(const DtNetCollisionElasticPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtCollisionElasticPdu(const DtCollisionElasticPdu& orig);

    // destructor
    virtual ~DtCollisionElasticPdu();

    // Create a DtCollisionElasticPdu.
    // Caller is responsible for deletion.
    static DtPdu* create(const DtNetPacket* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // Add (register)/Remove (deregister) function to be called when
    // pdu is received.
    static void addCallback(DtExerciseConn* conn,
        DtCollisionElasticPduCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtCollisionElasticPduCb cb, void* usr);
```

```
// Assignment operator
DtCollisionElasticPdu& operator=(const DtCollisionElasticPdu&
orig);

// Set/Get issuing entity identifier
virtual void setIssuingEntityID(const DtEntityIdentifier&
entity);
virtual const DtEntityIdentifier& issuingEntityID() const;

// Set/Get colliding entity identifier
virtual void setCollidingEntityID(const DtEntityIdentifier&
entity);
virtual const DtEntityIdentifier& collidingEntityID() const;

// Set/Get event ID
virtual void setEventID(const DtEventID& event);
virtual const DtEventID& eventID() const;

// Set/Get contact velocity
virtual void setContactVelocity(DtConstVector velocity);
virtual DtConstVector contactVelocity() const;

// Set/Get mass
virtual void setMass(DtFloat32 value);
virtual DtFloat32 mass() const;

// Set/Get location of impact
virtual void setLocationOfImpact(DtConstVector coords);
virtual DtConstVector locationOfImpact() const;

// Set/Get collision intermediate result XX
virtual void setCollisionInterResultXX(DtFloat32 result);
virtual DtFloat32 collisionInterResultXX() const;

// Set/Get collision intermediate result XY
virtual void setCollisionInterResultXY(DtFloat32 result);
virtual DtFloat32 collisionInterResultXY() const;

// Set/Get collision intermediate result XZ
virtual void setCollisionInterResultXZ(DtFloat32 result);
virtual DtFloat32 collisionInterResultXZ() const;

// Set/Get collision intermediate result YY
virtual void setCollisionInterResultYY(DtFloat32 result);
virtual DtFloat32 collisionInterResultYY() const;

// Set/Get collision intermediate result YZ
virtual void setCollisionInterResultYZ(DtFloat32 result);
virtual DtFloat32 collisionInterResultYZ() const;
```

```
// Set/Get collision intermediate result ZZ
virtual void setCollisionInterResultZZ(DtFloat32 result);
virtual DtFloat32 collisionInterResultZZ() const;

// Set/Get unit surface normal
virtual void setUnitSurfaceNormal(DtConstVector coords);
virtual DtConstVector unitSurfaceNormal() const;

// Set/Get coefficient of restitution
virtual void setCoefficientOfRestitution(DtFloat32 coeff);
virtual DtFloat32 coefficientOfRestitution() const;

// Print the data contained by this PDU
virtual void printData() const;
```

protected:

```
// Returns the internal PDU kind
virtual DtPduKind internalGetPduKind() const;

// Casts the netPdu element to our network struct
// Non-virtual since you can't override based on return type
virtual DtNetCollisionElasticPdu* netPduStruct() const;

// Casts the 'this' pointer for changing elements in a const
function
// Non virtual since you can't override based on return type
DtCollisionElasticPdu* self() const;
```

protected:

```
// Size of PDU with zero data bytes.
static int theStaticSize;
```

protected:

```
// Temporary objects for reference returns
DtEntityIdentifier myEntityID;
DtEventID myEventID;
DtVector myVector;
```

```
};
#endif
#endif
```

commentPdu.h

```

#if !DtHLA

#include "hostStructs.h"
#include "pduWithData.h"
#include "datumIdType.h"

#if DtDISVERS >= 5

#define DtCommentStaticSize 32

class DT_DLL_VL DtCommentPdu;
typedef DtCommentPdu DtCommentInteraction;
typedef void (*DtCommentPduCb)(DtCommentPdu *pdu, void *usr);

class DT_DLL_VL DtCommentPdu : public DtPduWithData
{
public:

/***** STANDARD CONSTRUCTORS *****/

    DtCommentPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtCommentPdu(const DtNetCommentPdu *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtCommentPdu(const DtEntityIdentifier &sender,
        const DtEntityIdentifier &receiver,
        const char *message,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    static DtPdu *create(const DtNetPacket *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    static void addCallback(DtExerciseConn *conn,
        DtCommentPduCb cb, void *usr);
    static void removeCallback(DtExerciseConn *conn,
        DtCommentPduCb cb, void *usr);

/***** INSPECTOR FUNCTIONS *****/

    /* none besides those inherited from DtPduWithData */

/***** MUTATOR FUNCTIONS *****/

    void setComment(const char *message, int size);

protected:

    DtPduKind internalGetPduKind() const {return DtCommentPduKind;}
    DtNetDataPduTail *dataSection() const;

```

```
};  
typedef DtCommentPdu DtMessagePdu;  
  
#endif // DtDISVERS */  
#endif // !DtHLA
```


commentRPdu.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "hostStructs.h"
#include "pduWithData.h"
#include "datumIdType.h"
#include "packets/commentRP.h"
#include "exerciseConn.h"

// class DtCommentRPdu:
//
// Instances of DtCommentRPdu are used to send/receive
// reliable comment PDUs.

// DtCommentRPdu callback function signature.
typedef void (*DtCommentRPduCb)(DtCommentRPdu* pdu, void* usr);

class DT_DLL_VL DtCommentRPdu : public DtPduWithData
{
public:

    // Constructors
    DtCommentRPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtCommentRPdu(const DtNetCommentRPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtCommentRPdu(const DtEntityIdentifier& sender,
        const DtEntityIdentifier& receiver,
        const char* message,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtCommentRPdu(const DtCommentRPdu& orig);

    // destructor
    virtual ~DtCommentRPdu();

    // Assignment operator
    DtCommentRPdu& operator=(const DtCommentRPdu& orig);

    // Set the comment.
    virtual void setComment(const char* message, int size);

public:

    // Create a DtCommentRPdu.
    // Caller is responsible for deletion.
    static DtPdu* create(const DtNetPacket* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

```

```
// Add (register)/Remove (deregister) function to be called when
// pdu is received.
static void addCallback(DtExerciseConn* conn,
    DtCommentRPduCb cb, void* usr);
static void removeCallback(DtExerciseConn* conn,
    DtCommentRPduCb cb, void* usr);
```

protected:

```
// Set the comment.
// Non-virtual since called from ctor.
void setCommentInternal(const char* message, int size);

// Returns the internal PDU kind
virtual DtPduKind internalGetPduKind() const;

// Returns the data portion of the pdu.
virtual DtNetDataPduTail* dataSection() const;
```

protected:

```
static int theStaticSize;
```

```
};
#endif // DtDISVERS
#endif // !DtHLA
```

createEntRPdu.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "hostStructs.h"
#include "simanPdu.h"
#include "packets/createEntRP.h"
#include "exerciseConn.h"

// class DtCreateEntityRPdu:
//
// Instances of DtCreateEntityRPdu are used to send/receive
// reliable create entity PDUs.

// DtCreateEntityRPdu callback function signature.
typedef void (*DtCreateEntityRPduCb)(DtCreateEntityRPdu* pdu, void*
usr);

class DT_DLL_VL DtCreateEntityRPdu : public DtSimanPdu
{
public:

    // Constructors
    DtCreateEntityRPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtCreateEntityRPdu(const DtNetCreateEntityRPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtCreateEntityRPdu(const DtEntityIdentifier& to,
        const DtEntityIdentifier& from,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtCreateEntityRPdu(const DtCreateEntityRPdu& orig);

    // destructor
    virtual ~DtCreateEntityRPdu();

    // Assignment operator
    DtCreateEntityRPdu& operator=(const DtCreateEntityRPdu& orig);

    // Set/Get the request id.
    virtual void setRequestId(int id);
    virtual int requestId() const;

    // Set/Get the required reliability service.
    virtual void setReqRelService(DtRequiredReliabilityService
value);
    virtual DtRequiredReliabilityService reqRelService() const;

    // Print the data contained by this PDU
    virtual void printData() const;

```

public:

```
// Create a DtCreateEntityRPdu.  
// Caller is responsible for deletion.  
static DtPdu* create(const DtNetPacket* initial,  
    DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);  
  
// Add (register)/Remove (deregister) function to be called when  
// pdu is received.  
static void addCallback(DtExerciseConn* conn,  
    DtCreateEntityRPduCb cb, void* usr);  
static void removeCallback(DtExerciseConn* conn,  
    DtCreateEntityRPduCb cb, void* usr);
```

protected:

```
// Returns the internal PDU kind  
virtual DtPduKind internalGetPduKind() const;
```

protected:

```
static int theStaticSize;
```

```
};  
#endif  
#endif
```

createPdu.h

```

#if !DtHLA

#include "hostStructs.h"
#include "simanPdu.h"

#define DtCreateStaticSize 28

class DT_DLL_VL DtCreateEntityPdu;
typedef DtCreateEntityPdu DtCreateEntityInteraction;
typedef void (*DtCreateEntityPduCb)(DtCreateEntityPdu *pdu, void
*usr);

class DT_DLL_VL DtCreateEntityPdu : public DtSimanPdu
{
public:

/***** STANDARD CONSTRUCTORS *****/

    DtCreateEntityPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtCreateEntityPdu(const DtNetCreateEntityPdu *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

/***** ADDITIONAL CONSTRUCTORS *****/

    DtCreateEntityPdu(const DtEntityIdentifier &to,
        const DtEntityIdentifier &from,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    static DtPdu *create(const DtNetPacket *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    static void addCallback(DtExerciseConn *conn,
        DtCreateEntityPduCb cb, void *usr);
    static void removeCallback(DtExerciseConn *conn,
        DtCreateEntityPduCb cb, void *usr);

/***** INSPECTOR FUNCTIONS *****/

#if DtDISVERS != 3
    int requestId() const;
#endif

/***** MUTATOR FUNCTIONS *****/

#if DtDISVERS != 3
    void setRequestId(int id);
    void printData() const;
#endif

```

```
protected:
    DtPduKind internalGetPduKind() const {return DtCreateEntityPdu-
Kind;}
};
#endif
```

dAggDec.h

```

#include "MachineTypes.h"
#include "hostStructs.h"
#include "stateMsg.h"
#include "exerciseConn.h"
#include "aggEntList.h"
#include "aggAggList.h"
#include "varDatumRec.h"

#if !DtHLA
#if DtDISVERS >= 5

#include "aggStatePdu.h"
#include "aggregateSR.h"

// class DtAggregateStateDecoder:
//
// Instances of DtAggregateStateDecoder are used to take state
// information
// from an aggregate state PDU and place it into an aggregate state
// repository.

class DT_DLL_VL DtAggregateStateDecoder
{
public:
    // default constructor
    DtAggregateStateDecoder();

    // destructor
    virtual ~DtAggregateStateDecoder();

    // copy constructor
    DtAggregateStateDecoder(const DtAggregateStateDecoder& orig);

    // assignment operator
    DtAggregateStateDecoder& operator=(const DtAggregateStateDecoder&
orig);
public:
    // Decode an aggregate state PDU and update a state repository
    virtual void decode(const DtAggregateStatePdu& pdu,
        DtAggregateStateRepository* stateRep);
};
#endif
#endif

```

dAggEnc.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "aggStatePdu.h"
#include "aggregateSR.h"

// class DtAggregateStateEncoder:
//
// Instances of DtAggregateStateEncoder are used to take state
// information
// from an aggregate state repository and place it into an aggregate
// state
// PDU.

class DT_DLL_VL DtAggregateStateEncoder
{
public:

    // default constructor
    DtAggregateStateEncoder();

    // destructor
    virtual ~DtAggregateStateEncoder();

    // copy constructor
    DtAggregateStateEncoder(const DtAggregateStateEncoder& orig);

    // assignment operator
    DtAggregateStateEncoder& operator=(const DtAggregateStateEncoder&
orig);

    // create a state message from a local and remote state reposi-
    // tory
    virtual DtAggregateStatePdu* stateMsg(
        const DtAggregateStateRepository& stateRep,
        const DtAggregateStateRepository& asSeenByRemote);

    // force an state message to be generated
    virtual void setSendNeeded();

protected:

    // determine if a state message needs to be sent
    virtual DtBoolean sendNeeded(
        const DtAggregateStateRepository& stateRep,
        const DtAggregateStateRepository& asSeenByRemote);
```



```
// fill out a state message which reflects a state repository
virtual void fillOut(
    const DtAggregateStateRepository& stateRep,
    const DtAggregateStateRepository& asSeenByRemote,
    DtAggregateStatePdu* pdu);

protected:

    DtAggregateStatePdu myASPdu;
    int forceSendNeeded;
};
#endif
#endif
```

dAggPub.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "exerciseConn.h"
#include "aggregateSR.h"
#include "dAggEnc.h"
#include "dAggDec.h"

#include "chooseDefines.h"

#ifdef __GNUC__ // g++
// Note: Here due to bug in g++ when using class static member func-
tion
// as a default argument. Don't call these otherwise.
#ifdef __GNUC__ // g++
#else
#undef gFindOwnId
#endif
#endif
#endif
```

dataPdu.h

```

#if !DtHLA

#include "hostStructs.h"
#include "pduWithData.h"
#include "datumIdType.h"

#define DtDataStaticSize 40

class DT_DLL_VL DtDataPdu;
typedef DtDataPdu DtDataInteraction;
typedef void (*DtDataPduCb)(DtDataPdu *pdu, void *usr);

class DT_DLL_VL DtDataPdu : public DtPduWithData
{
public:

/***** STANDARD CONSTRUCTORS *****/

    DtDataPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtDataPdu(const DtNetDataPdu *initial,
               DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    static DtPdu *create(const DtNetPacket *initial,
                        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    static void addCallback(DtExerciseConn *conn,
                           DtDataPduCb cb, void *usr);
    static void removeCallback(DtExerciseConn *conn,
                               DtDataPduCb cb, void *usr);

/***** INSPECTOR FUNCTIONS *****/

    unsigned long requestId() const;

/***** MUTATOR FUNCTIONS *****/

    void setRequestId(unsigned long id);

    void printData() const;

protected:

    DtPduKind internalGetPduKind() const {return DtDataPduKind;}
    DtNetDataPduTail *dataSection() const;
};
#endif

```

dataQueryRPdu.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "hostStructs.h"
#include "simanPdu.h"
#include "packets/dataQueryRP.h"
#include "exerciseConn.h"

// class DtDataQueryRPdu
//
// Instances of DtDataQueryRPdu are used to send/receive
// reliable data query PDUs.

// DtDataQueryRPdu callback function signature.
typedef void (*DtDataQueryRPduCb)(DtDataQueryRPdu* pdu, void* usr);

class DT_DLL_VL DtDataQueryRPdu : public DtSimanPdu
{
public:

    // Constructors
    DtDataQueryRPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtDataQueryRPdu(const DtNetDataQueryRPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtDataQueryRPdu(const DtDataQueryRPdu& orig);

    // destructor
    virtual ~DtDataQueryRPdu();

    // Assignment operator
    DtDataQueryRPdu& operator=(const DtDataQueryRPdu& orig);

    // Set/Get the request id.
    virtual void setRequestId(unsigned long id);
    virtual unsigned long requestId() const;

    // Set/Get the time interval.
    virtual void setTimeInterval(DtTime seconds);
    virtual DtTime timeInterval() const;

    // Get the number of fixed fields.
    virtual unsigned long numFixedFields() const;

    // Get the number of variable fields.
    virtual unsigned long numVarFields() const;

    // Set/Get the fixed datum id at the specified index.
```

```

virtual int setFixedDatumId(int index, DtDatumParam id);
virtual DtDatumParam fixedDatumId(int index) const;

// Set/Get the variable datum id at the specified index.
virtual int setVarDatumId(int index, DtDatumParam id);
virtual DtDatumParam varDatumId(int index) const;

// Set/Get the required reliability service.
virtual void setReqRelService(DtRequiredReliabilityService
value);
virtual DtRequiredReliabilityService reqRelService() const;

// Initialize the fixed and variable fields.
virtual void initDatumIds(unsigned long numFixedFields,
    unsigned long numVarFields);

// Print the data contained by this PDU
virtual void printData() const;

public:

    // Create a DtDataQueryRPdu.
    // Caller is responsible for deletion.
    static DtPdu* create(const DtNetPacket* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // Add (register)/Remove (deregister) function to be called when
    // pdu is received.
    static void addCallback(DtExerciseConn* conn,
        DtDataQueryRPduCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtDataQueryRPduCb cb, void* usr);

protected:

    // Returns the internal PDU kind
    virtual DtPduKind internalGetPduKind() const;

protected:

    static int theStaticSize;

};
#endif
#endif

```

dataRPdu.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "hostStructs.h"
#include "pduWithData.h"
#include "datumIdType.h"
#include "packets/dataRP.h"
#include "exerciseConn.h"

// class DtDataRPdu
//
// Instances of DtDataRPdu are used to send/receive
// reliable data PDUs.

// DtDataRPdu callback function signature.
typedef void (*DtDataRPduCb)(DtDataRPdu* pdu, void* usr);

class DT_DLL_VL DtDataRPdu : public DtPduWithData
{
public:

    // Constructors
    DtDataRPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtDataRPdu(const DtNetDataRPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtDataRPdu(const DtDataRPdu& orig);

    // destructor
    virtual ~DtDataRPdu();

    // Assignment operator
    DtDataRPdu& operator=(const DtDataRPdu& orig);

    // Set/Get the request id.
    virtual void setRequestId(unsigned long id);
    virtual unsigned long requestId() const;

    // Set/Get the required reliability service.
    virtual void setReqRelService(DtRequiredReliabilityService
value);
    virtual DtRequiredReliabilityService reqRelService() const;

    // Print the data contained by this PDU
    virtual void printData() const;

public:
```

```
// Create a DtDataRPdu.
// Caller is responsible for deletion.
static DtPdu* create(const DtNetPacket* initial,
    DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

// Add (register)/Remove (deregister) function to be called when
// pdu is received.
static void addCallback(DtExerciseConn* conn,
    DtDataRPduCb cb, void* usr);
static void removeCallback(DtExerciseConn* conn,
    DtDataRPduCb cb, void* usr);

protected:

    // Returns the internal PDU kind
    virtual DtPduKind internalGetPduKind() const;

    // Returns the data portion of the net structure.
    virtual DtNetDataPduTail* dataSection() const;

protected:

    static int theStaticSize;

};
#endif
#endif
```

datumIdType.h

```
#include "MachineTypes.h"
#include "disEnums.h"

#define DtTYPE_NOT_DEFINED 0
#define DtTYPE_BYTE_ARRAY 1
#define DtTYPE_FLOAT32 2
#define DtTYPE_FLOAT64 3
#define DtTYPE_INT32 4
#define DtTYPE_UNSIGNED32 5
#define DtTYPE_STRING 6

DT_DLL_VL int DtDatumId2Type(DtDatumParam id);
DT_DLL_VL int DtAddDatumId2TypeMapping(DtDatumParam id, int
dataType,
    int warn = 0);
```


datumMap.h

```

#if !DtHLA

#include "vlconfig.h"
#include <stdio.h>

#if DT_HAVE_STD_H
#include <std.h>
#endif

#include "hashlist.h"
#include "disEnums.h"
#include "setDataPdu.h"
#include "queryPdu.h"
#include "dataPdu.h"

class DT_DLL_VL DtDatumMapEntry
{
public:
    DtDatumMapEntry() {}
    virtual void setdata(const DtSetDataPdu &setDataPdu,
        int fixedOrVar, int index, void *usr) = 0;
    virtual void getdata(DtDataPdu *dataPdu, int fixedOrVar, int
index,
        void *usr) = 0;
};
typedef void (*DtGlobalSetFcn)(const DtSetDataPdu &pdu, int fixe-
dOrVar,
        int index, void *usr);
typedef void (*DtGlobalGetFcn)(DtDataPdu *pdu, int fixedOrVar,
        int index, void *usr);

class DT_DLL_VL DtDmeGlobalFcns : public DtDatumMapEntry
{
public:
    DtDmeGlobalFcns(DtGlobalSetFcn setF, DtGlobalGetFcn getF)
    : DtDatumMapEntry(), setFcn(setF), getFcn(getF) {}

    void setdata(const DtSetDataPdu &setDataPdu,
        int fixedOrVar, int index, void *usr)
    {
    if (setFcn)
        (*setFcn)(setDataPdu, fixedOrVar, index, usr);
    }

    void getdata(DtDataPdu *dataPdu, int fixedOrVar, int index,
        void *usr)
    {

```

```
    if (getFcn)
        (*getFcn)(dataPdu, fixedOrVar, index, usr);
    }
protected:
    DtGlobalGetFcn getFcn;
    DtGlobalSetFcn setFcn;
};

class DT_DLL_VL DtDatumMap {
public:
    DtDatumMap(void) : list(101) { }

    void setData(const DtSetDataPdu &data, void *usr);

    void fillDataPdu(DtDataPdu *dataPdu, void *usr);
    void initDataPdu(const DtSetDataPdu &setDataPdu, DtDataPdu
*dataPdu);
    void initDataPdu(const DtDataQueryPdu &queryPdu, DtDataPdu
*dataPdu);
    void makeResponseDataPdu(const DtSetDataPdu &setDataPdu,
        DtDataPdu *dataPdu, void *usr)
    {
        initDataPdu(setDataPdu, dataPdu);
        fillDataPdu(dataPdu, usr);
    }
    void makeResponseDataPdu(const DtDataQueryPdu &queryPdu,
        DtDataPdu *dataPdu, void *usr)
    {
        initDataPdu(queryPdu, dataPdu);
        fillDataPdu(dataPdu, usr);
    }

    DtDatumMapEntry *lookup(DtDatumParam id, int verbose=1);
    void install(DtDatumMapEntry *entry, DtDatumParam id);
    void remove(DtDatumParam id, int verbose=1);
protected:
    DtHashlist list;
};

#endif
```

datumRecSet.h

```

#include "hostStructs.h"
#include "datumIdType.h"
#include "baseMsg.h"
#include "packets/dataP.h"

enum DtDatumRetCode
{
    DtDatumValid,
    DtDatumBadLength,
    DtDatumBadId,
    DtDatumBadIndex
};

enum DtDatumRecordType
{
    DtFixed,
    DtVar
};

// This class has public data members for backwards
// compatibility with pre-VR-Link3.3 versions.
class DT_DLL_VL DtVarDatumSpec
{
public:
    DtDatumParam param;
    int length;
};
#define DtNullDatumParam DtDatumParam(0)
#define DtNullVarDatumSpec DtNullDatumParam,0

class DT_DLL_VL DtDatumRecordSet : public DtBaseMsg
{
public:
    // Constructor
    DtDatumRecordSet(DtBufferPtr bufferPtr = DtUSE_INTERNAL_BUFFER);

    // Constructor
    DtDatumRecordSet(const void* netRep, int size,
        DtBufferPtr bufferPtr = DtUSE_INTERNAL_BUFFER);

    // Destructor
    virtual ~DtDatumRecordSet();

    // Copy ctor
    DtDatumRecordSet(const DtDatumRecordSet& orig);

```

```
// Assignment operator
DtDatumRecordSet& operator=(const DtDatumRecordSet& orig);

// returns number of fixed fields
virtual int numFixedFields() const;

// returns number of variable length fields
virtual int numVarFields() const;

// Returns the datum param found in the index'th variable
// datum record. Index is between 1 and numRecords. Returns
// DtDatumParam(0) if index is out of bounds.
virtual DtDatumParam varDatumId(int index) const;

// Returns the datum param found in the index'th fixed
// datum record. Index is between 1 and numRecords. Returns
// DtDatumParam(0) if index is out of bounds.
virtual DtDatumParam fixedDatumId(int index) const;

// Returns the datum param found in the index'th fixed or vari-
// able
// datum record, depending on whether fixedOrVar is DtFixed or
// DtVar.
// Index is between 1 and numRecords. Returns
// DtDatumParam(0) if index is out of bounds.
virtual DtDatumParam datumId(DtDatumRecordType fixedOrVar, int
index) const;

// Returns the number of bits of data contained in the value
// fields of the index'th fixed or variable datum record,
// depending on whether fixedOrVar is DtFixed or DtVar. If
// DtFixed,
// the answer is always 32.
virtual int datumSizeBits(DtDatumRecordType fixedOrVar, int
index) const;

// Returns the number of bytes of data contained in the value
// fields of the index'th fixed or variable datum record,
// depending on whether fixedOrVar is DtFixed or DtVar. If
// DtFixed,
// the answer is always 4.
virtual int datumSizeBytes(DtDatumRecordType fixedOrVar, int
index) const;

// Looks through all fixed and variable records for one
// containing
// param as the parameter ID. If it finds one, fills supplied
// fixedOrVar
// pointer with DtFixed or DtVar, and index with the index to
// indicate
```

```

    // which record was found, and fills retcode with DtDatumValid.
    If param
        // was not found, retcode is filled with DtDatumBadId, and index
    and
        // fixedOrVar are invalid.
        virtual void getIndex(DtDatumParam param, DtDatumRecordType*
fixedOrVar,
        int* index, DtDatumRetCode* retcode = NULL) const;

// ***** INSPECTING DATUM VALUES BY INDEX *****
// If you know the data type of the value contained in a particular
// record, use one of the following to return it. fixedOrVar
// (either DtFixed
// or DtVar) and index (between 1 and numRecords) indicate which
// record
// you are inspecting. datumValByteArray is used when the data
// type is
// a character string, or something other than a long, unsigned
// long, float
// or double. It returns a pointer to a character string containing
// the
// exact bytes in the network representation that represents the
// value
// being inspected. The supplied retcode pointer will be filled
// with
// DtDatumValid if the value was successfully inspected, and DtDa-
// tumBadLength
// if either the index was invalid, or the size of the data indi-
// cated that
// it could not be interpreted as the chosen data type.

    virtual DtFloat32 datumValFloat32(DtDatumRecordType fixedOrVar,
int index,
        DtDatumRetCode* retcode = NULL) const;
    virtual DtFloat64 datumValFloat64(DtDatumRecordType fixedOrVar,
int index,
        DtDatumRetCode* retcode = NULL) const;
    virtual unsigned long datumValUnsigned32(DtDatumRecordType fixe-
dOrVar,
        int index, DtDatumRetCode* retcode = NULL) const;
    virtual long datumValInt32(DtDatumRecordType fixedOrVar, int
index,
        DtDatumRetCode* retcode = NULL) const;
    virtual const char* datumValByteArray(DtDatumRecordType fixe-
dOrVar,
        int index) const;

// ***** INSPECTING DATUM VALUES BY PARAM ID *****
// If you know the data type of the value contained in a particular

```

```
// record, use one of the following to return it. The particular
// record
// inspected will be the first one containing param as the datum ID.
// datumValByteArray is used when the data type is
// a character string, or something other than a long, unsigned
// long, float
// or double. It returns a pointer to a character string containing
// the
// exact bytes in the network representation that represents the
// value
// being inspected. The supplied retcode pointer will be filled
// with
// DtDatumValid if the value was successfully inspected, and DtDa-
// tumBadLength
// if either param was not found, or the size of the data indicated
// that
// it could not be interpreted as the chosen data type.

virtual DtFloat32 datumValFloat32(DtDatumParam param,
    DtDatumRetCode* retcode = NULL) const;
virtual DtFloat64 datumValFloat64(DtDatumParam param,
    DtDatumRetCode* retcode = NULL) const;
virtual unsigned long datumValUnsigned32(DtDatumParam param,
    DtDatumRetCode* retcode = NULL) const;
virtual long datumValInt32(DtDatumParam param,
    DtDatumRetCode* retcode = NULL) const;
virtual const char* datumValByteArray(DtDatumParam param,
    DtDatumRetCode* retcode = NULL) const;

// ***** MUTATOR FUNCTIONS *****

// Initialize to reflect the fixed parameters
// and variable parameters and sizes indicated. fixedParams is a
// NULL-
// terminated array of DtDatumParams. specList is an array of
// DtVarDatumSpecs
// terminated with the DtNullVarDatumSpec constant. This call
// can be made
// instead of calls to setNumVarDatumRecords, setNumFixedDatum-
// Record,
// and repeated calls to setDatumParam and setVarDataBytes.
virtual void init(const DtDatumParam* fixedParams,
    const DtVarDatumSpec* specList);

// Sets the number of fixed or variable fields to newNum. Previ-
// ously
// existing fixed fields with indices greater than newNum are
// deleted.
// If newNum exceeds the number of existing fields, trailing new
// fields
```

```

    // are created with zeros as datum params, sizes (for var only)
    and values.
    virtual void setNumFixedFields(int newNum);
    virtual void setNumVarFields(int newNum);

    // Set the datum ID in the index'th fixed or variable datum record
    to
    // param, depending on whether fixedOrVar is DtFixed or DtVar.
    Supplied
    // retcode pointer is filled with DtDatumValid if successful, or
    DtDatumBadIndex
    // if the index is not valid.
    virtual void setDatumParam(DtDatumRecordType fixedOrVar, int
    index, DtDatumParam param,
        DtDatumRetCode* retcode = NULL);

    // Allocates length bytes for the index'th variable datum record.
    // Should be called before using one of the setDatumVal functions
    // to actually set the data. Supplied retcode pointer is set to
    // DtDatumValid if successful, or DtDatumBadIndex if the index
    // is not valid.
    virtual void setVarDataBytes(int index, int length, DtDatumRet-
    Code* retcode = NULL);

    // ***** SETTING DATA VALUES BY INDEX *****
    // Use one of the following to set the data value in a fixed or vari-
    able datum
    // record, depending on the data type of your data. fixedOrVar
    (either DtFixed
    // or DtVar) and index (between 1 and numRecords) indicate which
    record
    // you are modifying. setDatumValByteArray is used when the data
    type is
    // a character string, or something other than a long, unsigned
    long, float
    // or double. It takes a pointer to a character string containing
    the
    // exact bytes that should be used in the network representation.
    // The supplied retcode pointer will be filled with
    // DtDatumValid if the value was successfully set, DtDatumBadIndex
    if the
    // index is invalid, and DtDatumBadLength if the size of the data
    indicated that
    // it could not be set from as the chosen data type.

    virtual void setDatumValFloat64(DtDatumRecordType fixedOrVar, int
    index, DtFloat64 val,
        DtDatumRetCode* retcode = NULL);
    virtual void setDatumValFloat32(DtDatumRecordType fixedOrVar, int
    index, DtFloat32 val,

```

```
DtDatumRetCode* retcode = NULL);
virtual void setDatumValInt32(DtDatumRecordType fixedOrVar, int
index, long val,
    DtDatumRetCode* retcode = NULL);
virtual void setDatumValUnsigned32(DtDatumRecordType fixedOrVar,
int index,
    unsigned long val, DtDatumRetCode* retcode = NULL);
virtual void setDatumValByteArray(DtDatumRecordType fixedOrVar,
int index,
    const char* val, DtDatumRetCode* retcode = NULL);

// ***** SETTING DATA VALUES BY PARAM ID *****
// Use one of the following to set the data value in a fixed or vari-
// able datum
// record, depending on the data type of your data. The particular
// record modified
// will be the first one containing param as its parameter ID.
// setDatumValBytesArray is used when the data type is
// a character string, or something other than a long, unsigned
// long, float
// or double. It takes a pointer to a character string containing
// the
// exact bytes that should be used in the network representation.
// The supplied retcode pointer will be filled with
// DtDatumValid if the value was successfully set, DtDatumBadId if
// param
// is invalid, and DtDatumBadLength if the size of the data indi-
// cated that
// it could not be set from the chosen data type.

virtual void setDatumValFloat64(DtDatumParam param, DtFloat64
val,
    DtDatumRetCode* retcode = NULL);
virtual void setDatumValFloat32(DtDatumParam param, DtFloat32
val,
    DtDatumRetCode* retcode = NULL);
virtual void setDatumValInt32(DtDatumParam param, long val,
    DtDatumRetCode* retcode = NULL);
virtual void setDatumValUnsigned32(DtDatumParam param, unsigned
long val,
    DtDatumRetCode* retcode = NULL);
virtual void setDatumValByteArray(DtDatumParam param, const char*
val,
    DtDatumRetCode* retcode = NULL);

// prints the data
virtual void printData() const;

// Returns a pointer to the network representation, as might
// appear
```



```

    // in a data PDU. Consists of numberOfFixedFields, numberOf-
    VarFields,
    // then the fixed and variable fields.
    virtual DtNetDataPduTail* netDataRep() const;

    // Return a pointer to the index'th fixed data record in the
    // network representation. Index is between 1 and numRecords.
    virtual DtNetFixDatum* nthFixedDataRecord(int index) const;

    // Return a pointer to the index'th variable data record in the
    // network representation. Index is between 1 and numRecords.
    virtual DtNetVarDatum* nthVarDataRecord(int index) const;

    // Sets the fixed data from a network representation of a
    // fixed datum record array. Also pass the number of fixed
    fields.
    virtual void setFromNetFixed(const char* fixedFields, int
    numFixedFields);

    // Sets the variable data from a network representation
    consisting of
    // the variable length records concatenated together. Also pass
    // the number of fields, and the total length of the variable data
    records.
    virtual void setFromNetVar(const char* varFields, int
    numVarFields,
        int totalVarFieldsLength);

protected:

    // Really, should not be pure virtual in DtBaseMsg, but since
    // it is, we need a null definition here.
    virtual void sizeChanged(int size);

    // Returns a pointer to the index'th fixed or variable record in
    the
    // network representation, depending on whether fixedOrVar is
    DtFixed
    // or DtVar.
    virtual void* datumValPtr(DtDatumRecordType fixedOrVar, int
    index) const;
};

// For backwards compatibility:
#define DtFIXED DtFixed
#define DtVAR DtVar
#define DtDATUM_VALID DtDatumValid
#define DtDATUM_BAD_LENGTH DtDatumBadLength
#define DtDATUM_BAD_ID DtDatumBadId
#define DtDATUM_BAD_INDEX DtDatumBadIndex

```

dcm.h

```
#include "vlVector.h"

typedef const DtDcm &DtConstDcm;
typedef DtDcm &DtDcmRef;

class DT_DLL_MATRIX DtDcm
{
public:
    DtDcm();
    DtDcm(double a, double b, double c,
          double d, double e, double f,
          double g, double h, double i);

    DtDcm(DtConstVector a, DtConstVector b, DtConstVector c);
    DtDcm(const DtDcm &mat);
    DtDcm &operator=(const DtDcm &mat);

    operator DtVector* ();
    operator const DtVector* () const;

    DtVector &operator[](int i);
    const DtVector &operator[](int i) const;

    // Prints the data in the matrix
    void print() const;

public:
    static DtConstDcm zero();
    static DtConstDcm identity();

protected:
    DtVector rep[3];
};
// Inline functions
```

deadReckoner.h

```

#include "approximator.h"
#include "disEnums.h"

// class DtDeadReckoner:
//
// Instances of DtDeadReckoner are used to provide approximated
// position and orientation information for entities using various
// dead reckoning algorithms.
// Position and orientation information is approximated over time
// from the last set absolute (non-approximated) information.
// Functions that start with deadReckoned return dead-reckoned
// values.
// Functions that start with approx return approximated values -
// which
// default to the dead-reckoned values, but this might be overridden
// in
// classes derived from DtDeadReckoner.

class DT_DLL_VL DtDeadReckoner : public DtApproximator
{
public:

    // default constructor
    DtDeadReckoner();

    // destructor
    virtual ~DtDeadReckoner();

    // copy constructor
    DtDeadReckoner(const DtDeadReckoner& orig);

    // assignment operator
    DtDeadReckoner& operator=(const DtDeadReckoner& orig);

    // Create a copy of this object. Memory is allocated and it is
    // the
    // responsibility of the caller to free the memory via delete.
    virtual DtApproximator* clone() const;

    // Set absolute location.
    virtual void setLocation(DtConstVector loc,
        DtTime curTime = DtGetSimTime());
    // Get the approximated (i.e. dead-reckoned for base DtDeadReck-
    // oner)
    // location at the specified time.
    virtual DtConstVector approxLocation(
        DtTime curTime = DtGetSimTime()) const;

```

```
// Get the dead-reckoned location at the specified time.
virtual DtConstVector deadReckonedLocation(
    DtTime curTime = DtGetSimTime()) const;

// Set absolute velocity.
virtual void setVelocity(DtConstVector vel,
    DtTime curTime = DtGetSimTime());
// Get the approximated (i.e. dead-reckoned for base DtDeadReck-
oner)
// velocity at the specified time.
virtual DtConstVector approxVelocity(
    DtTime curTime = DtGetSimTime()) const;
// Get the dead-reckoned velocity at the specified time.
virtual DtConstVector deadReckonedVelocity(
    DtTime curTime = DtGetSimTime()) const;

// Set absolute orientation.
virtual void setOrientation(const DtTaitBryan& ori,
    DtTime curTime = DtGetSimTime());
// Get the approximated (i.e. dead-reckoned for base DtDeadReck-
oner)
// orientation at the specified time.
virtual const DtTaitBryan& approxEuler(
    DtTime curTime = DtGetSimTime()) const;
// Get dead-reckoned orientation (euler angles) at the specified
time.
virtual const DtTaitBryan& deadReckonedEuler(
    DtTime curTime = DtGetSimTime()) const;

// Set/Get absolute acceleration.
virtual void setAcceleration(DtConstVector acc,
    DtTime curTime = DtGetSimTime());
virtual DtConstVector acceleration(
    DtTime curTime = DtGetSimTime()) const;

// Set/Get absolute rotational velocity.
virtual void setRotationalVelocity(DtConstVector vel,
    DtTime curTime = DtGetSimTime());
virtual DtConstVector rotationalVelocity(
    DtTime curTime = DtGetSimTime()) const;

// Get the approximated (i.e. dead-reckoned for base DtDeadReck-
oner)
// body to geocentric coordinate matrix.
virtual DtConstDcm approxBodyToGeoc(
    DtTime curTime = DtGetSimTime()) const;
// Get the dead-reckoned body to geocentric coordinate matrix.
virtual DtConstDcm deadReckonedBodyToGeoc(
    DtTime curTime = DtGetSimTime()) const;
```

```

    // Set/Get the dead reckoning algorithm used for generating the
    // approximated values from the absolute values.
    virtual void setAlgorithm(DtDeadReckonTypes alg);
    virtual DtDeadReckonTypes algorithm() const;

public:
    // Calculate the dead reckon position information.
    static void deadReckonPosition(DtTime dt, DtDeadReckonTypes algo-
rithm,
        DtConstVector oldPos, DtConstVector oldVel,
        DtConstVector oldAccel,
        DtVectorRef newVel, DtVectorRef newPos);

    // Calculate the orientation matrix.
    static void deadReckonOrientation(DtTime dt, DtDeadReckonTypes
algorithm,
        DtConstDcm old_b2w,
        const DtDirectedRotRate& drr,
        DtDcmRef new_b2w);

protected:
    // If the last dead reckoned position is not valid for the
    // specified time, recalculate it and same the validity time.
    virtual void deadReckonLoc(DtTime curTime) const;

    // If the last dead reckoned body to geocentri coordinate matrix
    // is not valid for the specified time, recalculate it and same
    // the validity time.
    virtual void deadReckonBodyToGeoc(DtTime curTime) const;

    // If the last dead reckoned orientation (euler angles) is not
    valid
    // for the specified time, recalculate it and same the validity
    time.
    virtual void deadReckonEuler(DtTime curTime) const;

    // Get rid of constness on the "this" pointer in order to access
    // non-const functions from const member functions (such as
    simple
    // accessors). This should be used sparingly since it bypasses
    // the const protection.
    // Note: This is essentially like an operator and is
    //         non-virtual since can't override based on return type.
    //         Allows derived classes to also define this member.
    DtDeadReckoner* self() const;

protected:

```

```
// Returns 1 if the dead reckoning algorithm is a rotational one,
// 0 otherwise.
static int isRotational(DtDeadReckonTypes algorithm);

protected:

    DtVector myLocAtTimeOfValidity;
    DtVector myVelAtTimeOfValidity;
    DtTime mySimTimePosDataValid;
    DtDcm myBodyToGeocAtTimeOfValidity;
    DtTime mySimTimeOrientDataValid;

    DtVector myDRLoc;
    DtVector myDRVel;
    DtTime myDRPostime;
    DtDcm myDRBodyToGeoc;
    DtTime myDRBodyToGeocTime;
    DtTaitBryan myDREuler;
    DtTime myDREulerTime;
    DtDirectedRotRate myDirectedRotRate;
    DtDeadReckonTypes myAlgorithm;
    DtVector myAccel;
    DtBodyRates myRotVel;
};
```

decFactory.h

```

#if DtHLA

#include "hStateDecoder.h"
#include "MachineTypes.h"

// class DtStateDecoderFactory:
//
// Instances of DtStateDecoderFactory are used to create decoders
// which
// know how to decode an HLA state message (or network representa-
// tion)
// into a state repository (or host representation).

class DT_DLL_VL DtStateDecoderFactory
{
public:

    // constructor
    DtStateDecoderFactory(DtExerciseConn* exConn);

    // destructor
    virtual ~DtStateDecoderFactory();

    // copy constructor
    // Copies across added decoder creation functions.
    DtStateDecoderFactory(const DtStateDecoderFactory& orig);

    // assignment operator
    // Copies across added decoder creation functions.
    DtStateDecoderFactory& operator=(const DtStateDecoderFactory&
orig);

    // Create a decoder for the specified object class.
    // Note: Caller is responsible for deleting.
    // Returns NULL if decoder creation function is not in the
factory.
    virtual DtHlaStateDecoder* createDecoder(RTI::ObjectClassHandle
hand);

    // Add a decoder prototype to the factory. The decoder prototype
    // will be cloned by createDecoder when called for the appro-
priate
    // handle. Caller deletes the decoder, but AFTER the decoder
    // factory no longer needs it. Returns any previous decoder
which was
    // replaced by the add.
    virtual DtHlaStateDecoder* addDecoder(

```

```
    RTI::ObjectClassHandle hand,
    DtHlaStateDecoder* decoder);

// This function is used to add an attribute decoder function
// to several prototype decoders all at once - namely, those
// associated with the object class denoted by hand, as well
// as all of its subclasses. If applyToSubclasses is DtFalse,
// then the function is only registered with the one decoder
// prototype, making it equivalent to
//   decoder(hand)->addDecoder(hand, decoder));
virtual void addAttributeDecoder(
    RTI::ObjectClassHandle classHand,
    RTI::AttributeHandle attrHand,
    DtHlaStateDecoder::DtAttributeDecoder decoder,
    DtBoolean applyToSubclasses = DtTrue);

// Same as above, but identify class by name
void addAttributeDecoder(
    char* className,
    RTI::AttributeHandle attrHand,
    DtHlaStateDecoder::DtAttributeDecoder func,
    DtBoolean applyToSubclasses = DtTrue);

// Same as above, but identify attr by name
void addAttributeDecoder(
    RTI::ObjectClassHandle classHand,
    char* attrName,
    DtHlaStateDecoder::DtAttributeDecoder func,
    DtBoolean applyToSubclasses = DtTrue);

// Same as above, but identify class and attr by name
void addAttributeDecoder(
    char* className,
    char* attrName,
    DtHlaStateDecoder::DtAttributeDecoder func,
    DtBoolean applyToSubclasses = DtTrue);

// Returns the decoder prototype associated with handle by a
previous
// call to addDecoder. If defaultToBase is DtTrue, then if there
is
// no decoder registered with hand, rather than returning NULL,
// we search up the class hierarchy, and return the decoder
registered
// with the most specific superclass for which a decoder has
// been registered.
virtual DtHlaStateDecoder* decoder(RTI::ObjectClassHandle hand,
    DtBoolean defaultToBase = DtFalse);

protected:
```



```
    // myProtos is an array of DtHlaObject::maxObjectClasses() items.  
    DtHlaStateDecoder** myProtos;  
    DtExerciseConn* myExConn;  
};  
#endif
```

dEntityDec.h

```
#if !DtHLA

#include "esPdu.h"
#include "entitySR.h"

// class DtEntityStateDecoder:
//
// Instances of DtEntityStateDecoder are objects capable of decoding
// EntityState PDUs into a DtEntityStateRepository

class DT_DLL_VL DtEntityStateDecoder
{
public:

    // constructor
    DtEntityStateDecoder(DtExerciseConn* exConn);

    // destructor
    virtual ~DtEntityStateDecoder();

    // copy constructor
    DtEntityStateDecoder(const DtEntityStateDecoder& orig);

    // assignment operator
    DtEntityStateDecoder& operator=(const DtEntityStateDecoder&
orig);

    // decode a PDU into an EntityStateRepository
    virtual void decode(const DtEntityStatePdu& pdu,
        DtEntityStateRepository* stateRep);

protected:

    // decode the articulated parts portion of a
    // PDU into an EntityStateRepository
    virtual void decodeArtParts(
        const DtEntityStatePdu& pdu,
        DtEntityStateRepository* stateRep);

    // decode the attached parts portion of a
    // PDU into an EntityStateRepository
    virtual void decodeAttachedParts(
        const DtEntityStatePdu& pdu,
        DtEntityStateRepository* stateRep);

protected:
```

```
    DtExerciseConn* myExConn;  
};  
#endif
```

dEntityEnc.h

```
#if !DtHLA

#include "esPdu.h"
#include "entitySR.h"

class DT_DLL_VL DtEntityStateEncoder
{
public:

    // Constructor
    DtEntityStateEncoder();

    virtual DtEntityStatePdu*
        stateMsg(const DtEntityStateRepository& stateRep,
                const DtEntityStateRepository& asSeenByRemote);

    // Force the generation of an entity state PDU next time
    // stateMsg is called even if one would not otherwise be needed.
    virtual void setSendNeeded();

protected:

    // Returns true if a PDU needs to be sent, (based on thresholds,
    etc)
    // false otherwise.
    virtual DtBoolean sendNeeded(const DtEntityStateRepository&
stateRep,
        const DtEntityStateRepository& asSeenByRemote);

    // Copy current ESR data to the pdu.
    virtual void fillOut(const DtEntityStateRepository& stateRep,
        const DtEntityStateRepository& asSeenByRemote,
        DtEntityStatePdu* pdu);

    // Returns true if art part info is out of date, false otherwise.
    virtual DtBoolean artPartsNeedUpdating(
        const DtEntityStateRepository& stateRep,
        const DtEntityStateRepository& asSeenByRemote);

    // Returns true if att part info is out of date, false otherwise.
    virtual DtBoolean attPartsNeedUpdating(
        const DtEntityStateRepository& stateRep,
        const DtEntityStateRepository& asSeenByRemote);

    // Copies current articulated and attached part info
    // from the ESR to the PDU.
    virtual void fillOutArtAndAttParts(
```

```
    const DtEntityStateRepository& stateRep,  
    const DtEntityStateRepository& asSeenByRemote,  
    DtEntityStatePdu* pdu);
```

```
protected:
```

```
    DtEntityStatePdu myESPdu;  
    int myForceSendNeeded;  
};  
#endif
```

dEntityPub.h

```
#if !DtHLA

#include "exerciseConn.h"
#include "entitySR.h"
#include "vlprint.h"
#include "dEntityEnc.h"
#include "dEntityDec.h"

#include "chooseDefines.h"

#ifdef __GNUC__ // g++
// Note: Here due to bug in g++ when using class static member func-
tion
// as a default argument. Don't call these otherwise.
#ifdef __GNUC__ // g++
#else
#undef    gFindOwnId
#undef    gGuiseSameAsType
#endif
#endif

#endif // !DtHLA
```

designatePdu.h

```

#if !DtHLA

#if DtDISVERS != 3

#include "hostStructs.h"
#include "pdu.h"
#include "mathTypes.h"
#include "packets/designateP.h"

#define DtDesignatorStaticSizeVERS5 88
#define DtDesignatorStaticSizeVERS4 72
#if DtDISVERS >= 5
#define DtDesignatorStaticSize DtDesignatorStaticSizeVERS5
#else
#define DtDesignatorStaticSize DtDesignatorStaticSizeVERS4
#endif

class DT_DLL_VL DtDesignatorPdu;
typedef void (*DtDesignatorPduCb)(DtDesignatorPdu *pdu, void *usr);

class DT_DLL_VL DtDesignatorPdu : public DtPdu
{
public:

/***** STANDARD CONSTRUCTORS *****/

    DtDesignatorPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtDesignatorPdu(const DtNetDesignatorPdu *initial,
                    DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    static DtPdu *create(const DtNetPacket *initial,
                        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    static void addCallback(DtExerciseConn *conn,
                           DtDesignatorPduCb cb, void *usr);
    static void removeCallback(DtExerciseConn *conn,
                               DtDesignatorPduCb cb, void *usr);

/***** INSPECTOR FUNCTIONS *****/

    const DtEntityIdentifier &designatingEntityId() const;
    DtDesignatorCodeName codeName() const;
    const DtEntityIdentifier &designatedEntityId() const;
    DtDesignatorCode designatorCode() const;
    float power() const;
    float wavelength() const;
    DtConstVector designatorSpotRelative() const;
    DtConstVector designatorSpotWorld() const;

```

```
#if DtDISVERS >= 5
    DtDeadReckonTypes deadReckoningAlgorithm() const;
    DtConstVector linearAcceleration() const;
#endif

/***** MUTATOR FUNCTIONS *****/

    void setDesignatingEntityId(const DtEntityIdentifier &id);
    void setCodeName(DtDesignatorCodeName name);
    void setDesignatedEntityId(const DtEntityIdentifier &id);
    void setDesignatorCode(DtDesignatorCode code);
    void setPower(float power);
    void setWavelength(float wavelength);
    void setDesignatorSpotRelative(DtConstVector loc);
    void setDesignatorSpotWorld(DtConstVector loc);
#if DtDISVERS >= 5
    void setDeadReckoningAlgorithm(DtDeadReckonTypes alg);
    void setLinearAcceleration(DtConstVector acc);
#endif
    void printData() const;
protected:
    DtPduKind internalGetPduKind() const {return DtDesignatorPdu-
Kind;}
    DtEntityIdentifier theId;
    DtVector loc;
};
#endif // DtDISVERS
#endif // DtHLA
```


designatorDec.h

```

#if DtHLA

#ifndef DtDesignatorDecoder_H_
#define DtDesignatorDecoder_H_

#include "chooseDefines.h"
#include "baseEmbedDec.h"

// This is used to declare attribute decoding functions
// which are members of the class.
#define DtDECLARE_DESIGNATOR_ATTR_DECODER(attrName) \
    DtDECLARE_ATTR_DECODER(DtDesignatorRepository, attrName)

// class DtDesignatorDecoder:
//
// Instances of DtDesignatorDecoder are

class DT_DLL_VL DtDesignatorDecoder : public DtBaseEmbeddedSystemDe-
coder
{
public:

    // default constructor
    DtDesignatorDecoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtDesignatorDecoder();

    // copy constructor
    DtDesignatorDecoder(const DtDesignatorDecoder& orig);

    // assignment operator
    DtDesignatorDecoder& operator=(const DtDesignatorDecoder& orig);

public:

    // Create a DtBaseEntityDecoder. Caller is responsible for dele-
    tion.
    static DtHlaStateDecoder* create(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

protected:

    // Add decoding functions for the attributes.

```

```
// Note: Non-virtual since meant to be called from constructors
to
//      add decoders.
void addMyDecoders();

// static decode<AttributeName> member functions.
DtDECLARE_DESIGNATOR_ATTR_DECODER(CodeName);
DtDECLARE_DESIGNATOR_ATTR_DECODER(DesignatedObject);
DtDECLARE_DESIGNATOR_ATTR_DECODER(DesignatorCode);
DtDECLARE_DESIGNATOR_ATTR_DECODER(DesignatorID);
DtDECLARE_DESIGNATOR_ATTR_DECODER(DesignatorPower);
DtDECLARE_DESIGNATOR_ATTR_DECODER(DesignatorSpotLocation);
DtDECLARE_DESIGNATOR_ATTR_DECODER(DesignatorWavelength);
DtDECLARE_DESIGNATOR_ATTR_DECODER(DRAgorithm);
DtDECLARE_DESIGNATOR_ATTR_DECODER(RelativeSpotLocation);
DtDECLARE_DESIGNATOR_ATTR_DECODER(SpotLinearAcceleration);
};
// These macros are used to define simple attribute decoder func-
tions
// (members of a derived DtHlaStateDecoder that decode
// individual attributes). The simple decoding is just to
// convert the value pulled out of the ahvps to a host type using
// the cast expression castExpr, and pass it to the mutator func-
tion.
#define DtDEFINE_SIMPLE_DESIGNATOR_ATTR_DECODER_WITH_CAST( \
    attrName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_ATTR_DECODER_WITH_CAST( \
        DtDesignatorDecoder, DtDesignatorRepository, \
        attrName, netType, mutator, castExpr)

#define DtDEFINE_SIMPLE_DESIGNATOR_ATTR_DECODER( \
    attrName, netType, inspector) \
    DtDEFINE_SIMPLE_DESIGNATOR_ATTR_DECODER_WITH_CAST( \
        attrName, netType, inspector, (netType))

#endif //DtDesignatorDecoder_H_
```

designatorEnc.h

```

#if DtHLA

#ifndef DtDesignatorEncoder_H_
#define DtDesignatorEncoder_H_

#include "chooseDefines.h"
#include "baseEmbedEnc.h"

// This is used to declare static member functions for determining
// whether or not an attribute needs to be encoded.
#define DtDECLARE_DESIGNATOR_ATTR_CHECKER(attrName) \
    DtDECLARE_ATTR_CHECKER(DtDesignatorRepository, attrName)

// This is used to declare attribute encoding functions
// which are members of the class.
#define DtDECLARE_DESIGNATOR_ATTR_ENCODER(attrName) \
    DtDECLARE_ATTR_ENCODER(DtDesignatorRepository, attrName)

// class DtDesignatorEncoder:
//
// Instances of DtDesignatorEncoder are

class DT_DLL_VL DtDesignatorEncoder : public DtBaseEmbeddedSystemEncoder
{
public:

    // default constructor
    DtDesignatorEncoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtDesignatorEncoder();

    // copy constructor
    DtDesignatorEncoder(const DtDesignatorEncoder& orig);

    // assignment operator
    DtDesignatorEncoder& operator=(const DtDesignatorEncoder& orig);

public:

    // Create a DtBaseEntityEncoder. Caller is responsible for deletion.
    static DtHlaStateEncoder* create(
        DtExerciseConn* exConn,

```

```
        DtObjClassDesc* classDesc);

protected:

    DtDECLARE_DESIGNATOR_ATTR_CHECKER(CodeName);
    DtDECLARE_DESIGNATOR_ATTR_CHECKER(DesignatedObject);
    DtDECLARE_DESIGNATOR_ATTR_CHECKER(DesignatorCode);
    DtDECLARE_DESIGNATOR_ATTR_CHECKER(DesignatorID);
    DtDECLARE_DESIGNATOR_ATTR_CHECKER(DesignatorPower);
    DtDECLARE_DESIGNATOR_ATTR_CHECKER(DesignatorSpotLocation);
    DtDECLARE_DESIGNATOR_ATTR_CHECKER(DesignatorWavelength);
    DtDECLARE_DESIGNATOR_ATTR_CHECKER(DRAlgorithm);
    DtDECLARE_DESIGNATOR_ATTR_CHECKER(RelativeSpotLocation);
    DtDECLARE_DESIGNATOR_ATTR_CHECKER(SpotLinearAcceleration);

    DtDECLARE_DESIGNATOR_ATTR_ENCODER(CodeName);
    DtDECLARE_DESIGNATOR_ATTR_ENCODER(DesignatedObject);
    DtDECLARE_DESIGNATOR_ATTR_ENCODER(DesignatorCode);
    DtDECLARE_DESIGNATOR_ATTR_ENCODER(DesignatorID);
    DtDECLARE_DESIGNATOR_ATTR_ENCODER(DesignatorPower);
    DtDECLARE_DESIGNATOR_ATTR_ENCODER(DesignatorSpotLocation);
    DtDECLARE_DESIGNATOR_ATTR_ENCODER(DesignatorWavelength);
    DtDECLARE_DESIGNATOR_ATTR_ENCODER(DRAlgorithm);
    DtDECLARE_DESIGNATOR_ATTR_ENCODER(RelativeSpotLocation);
    DtDECLARE_DESIGNATOR_ATTR_ENCODER(SpotLinearAcceleration);
};
// This macro is used to define simple attribute encoder functions
// (members of a derived DtHlaStateEncoder that encode
// individual attributes). The simple encoding is just to implicitly
// convert the value returned by the inspector to a netType,
// and add it to the ahvps.
#define DtDEFINE_SIMPLE_DESIGNATOR_ATTR_ENCODER( \
    attrName, netType, inspector) \
    DtDEFINE_SIMPLE_ATTR_ENCODER(DtDesignatorEncoder, \
        DtDesignatorRepository, \
        attrName, netType, inspector)

// This macro is used to define simple attribute checker functions
// (members of a derived DtHlaStateEncoder that check whether
// a particular attribute is out of date. The simple implementation
// is to just compare the values returned by the two StateReps'
// inspector functions.
#define DtDEFINE_SIMPLE_DESIGNATOR_ATTR_CHECKER( \
    attrName, inspector) \
    DtDEFINE_SIMPLE_ATTR_CHECKER(DtDesignatorEncoder, \
        DtDesignatorRepository, \
        attrName, inspector)

#endif //DtDesignatorEncoder_H_
```

designatorSR.h

```

#include "embedSystemSR.h"

// class DtDesignatorRepository:
//
// Instances of DtDesignatorRepository are

class DT_DLL_VL DtDesignatorRepository : public DtEmbeddedSystemRe-
pository
{
public:

    // default constructor
    DtDesignatorRepository();

    // destructor
    virtual ~DtDesignatorRepository();

    // copy constructor
    DtDesignatorRepository(const DtDesignatorRepository& orig);

    // assignment operator
    DtDesignatorRepository& operator=(const DtDesignatorRepository&
orig);

    // return an empty object of the same type as this
    virtual DtStateRepository* clone() const;

    static DtDesignatorRepository* create();

    virtual void printData() const;

    // Set/Get the code name.
    virtual void setCodeName(const DtDesignatorCodeName& name);
    virtual const DtDesignatorCodeName& codeName() const;

    // Set/Get the designated object name.
    virtual void setDesignatedObject(const DtGlobalObjectDesignator&
name);
    virtual const DtGlobalObjectDesignator& designatedObject() const;

    // Set/Get the designator code.
    virtual void setDesignatorCode(const DtDesignatorCode& code);
    virtual const DtDesignatorCode& designatorCode() const;

    // Set/Get the designator object id.
    // Not communicated in RPR FOM 0.7.
    virtual void setDesignatorId(const DtGlobalObjectDesignator& id);

```

```
virtual const DtGlobalObjectDesignator& designatorId() const;

// Set/Get the designator power.
virtual void setDesignatorPower(DtFloat32 power);
virtual DtFloat32 designatorPower() const;

// Set/Get the designator spot location.
virtual void setDesignatorSpotLocation(DtConstVector location);
virtual DtConstVector designatorSpotLocation() const;

// Set/Get the designator wavelength.
virtual void setDesignatorWavelength(DtFloat32 wavelength);
virtual DtFloat32 designatorWavelength() const;

// Set/Get the dead reckoning algorithm.
virtual void setDRAlgorithm(DtDeadReckonTypes algorithm);
virtual DtDeadReckonTypes dRAlgorithm() const;

// Set/Get the relative spot location.
virtual void setRelativeSpotLocation(DtConstVector location);
virtual DtConstVector relativeSpotLocation() const;

// Set/Get the spot linear acceleration.
virtual void setSpotLinearAcceleration(DtConstVector accelera-
tion);
virtual DtConstVector spotLinearAcceleration() const;

protected:
    DtDesignatorCodeName      myCodeName;
    DtGlobalObjectDesignator  myDesignatedObject;
    DtGlobalObjectDesignator  myDesignatorId;
    DtDesignatorCode          myDesignatorCode;
    DtFloat32                  myDesignatorPower;
    DtVector                   myDesignatorSpotLocation;
    DtFloat32                  myDesignatorWavelength;
    DtDeadReckonTypes          myDRAlgorithm;
    DtVector                   myRelativeSpotLocation;
    DtVector                   mySpotLinearAcceleration;
};
```

detonatePdu.h

```

#if !DtHLA

#include "interaction.h"
#include "packets/detonationP.h"
#include "hostStructs.h"
#include "mathTypes.h"
#include "apSpec.h"
#include "fireInter.h"

#include "chooseDefines.h"

#define DtDetonateStaticSize 104

class DT_DLL_VL DtDetonationPdu;
typedef DtDetonationPdu DtDetonationInteraction;
typedef void (*DtDetonationInteractionCb)(DtDetonationInteraction
*inter,
        void *usr);

class DT_DLL_VL DtDetonationPdu : public DtInteraction
{
public:

    /***** STANDARD CONSTRUCTORS *****/

    DtDetonationPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtDetonationPdu(const DtNetDetonationPdu *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // Fills in common fields with data from Fire Interaction
    DtDetonationPdu(const DtFireInteraction &fire,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    static DtPdu *create(const DtNetPacket *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    static void addCallback(DtExerciseConn *conn,
        DtDetonationInteractionCb cb, void *usr);
    static void removeCallback(DtExerciseConn *conn,
        DtDetonationInteractionCb cb, void *usr);

    /***** ADDITIONAL CONSTRUCTORS *****/
    DtDetonationPdu(const DtEntityIdentifier &attacker,
        const DtEntityIdentifier &target,
        const DtEntityIdentifier &munition,
        const DtEventID &eventId,
        DtConstVector linVelocity,

```

```
        DtConstVector location,
        const DtBurstDescriptor &burst,
        DtConstVector entLocation,
        DtDetonationResult result,
        const DtArtPartSpec *specList,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

/***** INSPECTOR FUDCTIONS *****/

const DtObjectId &attackerId() const;
const DtObjectId &targetId() const;
const DtObjectId &munitionId() const;
const DtEventID &eventId() const;
DtConstVector velocity() const;
DtConstVector worldLocation() const;
DtConstVector entityLocation() const;
DtDetonationResult result() const;
DtDetonatorFuze fuseType() const;
const DtEntityType &munitionType() const;
int quantity() const;
int rate() const;
DtWarheadType warheadType() const;

// NOT USED IN HLA
const DtBurstDescriptor &burst() const;

int numArtParams() const;
const DtArtPartSpec *artPartSpecs() const;
int getArtParamVal(int paramIndex, DtFloat32 *value) const;
int getArtParamVal(int partType, int param, DtFloat32 *value)
const;

/***** MUTATOR FUDCTIONS *****/

void setAttackerId(const DtObjectId &id);
void setTargetId(const DtObjectId &id);
void setMunitionId(const DtObjectId &id);
void setEventId(const DtEventID &id);
void setVelocity(DtConstVector velocity);
void setWorldLocation(DtConstVector location);
void setEntityLocation(DtConstVector location);
void setResult(DtDetonationResult result);
void setFuseType(DtDetonatorFuze fuse);
void setMunitionType(const DtEntityType &munType);
void setQuantity(int quant);
void setRate(int rate);
void setWarheadType(DtWarheadType type);

// NOT USED IN HLA
```



```
void setBurst(const DtBurstDescriptor &burst);

void initArtParts(const DtArtPartSpec *specList);
int  setArtParamVal(int partType, int param, DtFloat32 value);
int  setArtParamVal(int paramIndex, DtFloat32 value);

void printData() const;

protected:
    DtPduKind internalGetPduKind() const {return DtDetonationPdu-
Kind;}
    void setNumArtParams(int numParams);
    DtNetArtPartRecord *artPartRecord(int partType, int param) const;
    void setArtParamVal(DtNetArtPartRecord *part, DtFloat32 value);
    float getArtParamVal(DtNetArtPartRecord *part) const;

    DtObjectId att;
    DtObjectId tar;
    DtObjectId mun;
    DtEventID event;
    DtVector vel, world, ent;
    DtBurstDescriptor theBurst;
    DtEntityType myMunType;
};
#endif
```

dExerciseConn.h

```
#if !DtHLA
#include "MachineTypes.h"

#include "Socket.h"
#include "pduCbMgr.h"
#include "disEnums.h"
#include "packets/pduHeader.h"
#include "hostStructs.h"
#include "mathTypes.h"
#include "hashlist.h"
#include "baseExConn.h"
#include "pduFactory.h"
#include "tStampType.h"

#include "chooseDefines.h"

#define DtMaxExerciseId 255
const DtExerciseId DtAnyExercise = 0;

typedef int (*DtPredicate)(DtPdu *pdu, void *arg);

#define DtNET_READ_SUCCESS 0
#define DtNET_READ_NO_PACKETS 1
#define DtNET_READ_BAD_PACKET 2
#define DtNET_READ_SERVER_MSG 3
#define DtNET_READ_SIZE_MISMATCH4
#define DtPROCESS_PDU_SUCCESS DtNET_READ_SUCCESS
#define DtPROCESS_PDU_WRONG_EXID 5
#define DtPROCESS_PDU_NO_PDU 6
#define DtNET_READ_FILTERED_PACKET7

class DT_DLL_VL DtExerciseConn : public DtBaseExerciseConn
{
public:

    // Constructors
    DtExerciseConn(int port, int exercise_id,
        int site_id, int application_num);
    DtExerciseConn(int port, DtInetAddr destAddr, int exercise_id,
        int site_id, int application_num);
    DtExerciseConn(DtSocket *skt, int exercise_id,
        int site_id, int application_num);

    // Destructor
    virtual ~DtExerciseConn();

    // ----- READING PDUs -----
```

```

// Register/deregister callbacks by PDU kind. Called by each
// derived DtPdu's addCallback function, which should be called
// instead of these by user code.
virtual void addPduCallback(
    DtPduKind pduKind, DtPduCallbackFcn fcn, void *usr);
virtual void removePduCallback(
    DtPduKind pduKind, DtPduCallbackFcn fcn, void *usr);

// Normal way of processing input from the network. Repeatedly
calls
// readAndProcess to read PDUs and execute user callbacks. If
timeout
// is -1, return only when there are no PDUs left to read. Other-
wise,
// read for a maximum of timeout seconds. nReadPerTimeCheck is
the
// number of PDUs to read in between checks to see if timeout has
expired.
virtual int drainInput(DtTime timeout=-1, int nReadPerTi-
meCheck=100);

// Called by drainInput, so application code rarely needs to use
this.
// Reads and processes (via netRead and processPdu) a single PDU.
virtual int readAndProcess(void);

// Called by readAndProcess (which is called by drainInput). The
// default behavior is to execute any user callbacks that have
been
// registered on the pdu's kind.
virtual int processPdu(DtPdu *pdu);

// Note: netRead() creates a new DtPdu on the heap. Caller,
// (usually readAndProcess()), is responsible for deleting
// it when through.
virtual DtPdu *netRead(int *retCode = NULL);

// Adds/subtracts interest in a PDU kind. PDUs in which no
interest
// has been registered are filtered out below the level of
netRead
// (possibly by the packet server). All PDU kinds for which user
// callbacks have been registered automatically are "inter-
esting",
// so there is rarely a need to application code to call these
functions.
virtual void addInterestInPduKind(DtPduKind kind);
virtual void subtractInterestInPduKind(DtPduKind kind);

```

```
// Repeatedly call readAndProcess until predicate holds on the
PDU
// that is being read, or until timeout seconds have elapsed.
sleepTime
// indicates the amount of time to sleep in between polling the
network
// when there are no packets to read.
virtual DtPduKind readUntil(DtPredicate predicate, void *arg,
    DtTime timeout, DtTime sleepTime=0.1);

// Add/Subtract interest in a multicast address. Valid multicast
// addresses are between 224.0.0.0 and 239.255.255.255.
virtual void addInterestInMcastAddr(DtInetAddr addr);
virtual void subtractInterestInMcastAddr(DtInetAddr addr);

// ----- SENDING PDUs -----

// Send a message - call down to send(DtPdu &pdu....)
virtual void send(const DtStateMsg &msg);
virtual void send(const DtInteraction &msg);

// Send a stamped message - call down to sendStamped(DtPdu
&pdu....)
virtual void sendStamped(const DtStateMsg &msg);
virtual void sendStamped(const DtInteraction &msg);

// Send a PDU to the specified address, or the DtExerciseConn's
default
// destination address if addr is DtUSE_DEFAULT_IP_ADDR. PDU
must be
// completely filled out, as it is not touched by this call.
virtual int send(
    const DtPdu &pdu, DtInetAddr addr = DtUSE_DEFAULT_IP_ADDR);

// Send a PDU to the specified address, or the DtExerciseConn's
default
// destination address if addr is DtUSE_DEFAULT_IP_ADDR. The
outgoing
// PDU is stamped with the current timeStamp (obtained using
// currentTimeForStamping()), and the current exerciseId.
virtual int sendStamped(
    const DtPdu &pdu, DtInetAddr addr = DtUSE_DEFAULT_IP_ADDR);

// Returns the current destination address for the PDU. Defaults
to
// DtUSE_DEFAULT_IP_ADDR.
virtual DtInetAddr destinationAddr(const DtPdu &pdu);

// ----- ID MANAGEMENT -----
```

```

// Set/Get the current exercise ID
virtual void setExerciseId(int exercise_id);
virtual DtExerciseId exerciseId();

// Returns the current application ID (site & host)
virtual DtSimulationAddress applicationId();

// Returns the next valid entity ID formed by calling makeEntityId
// with the current value of a entity number counter.
virtual const DtEntityIdentifier& nextId();

// Returns an entity ID consisting of the DtExerciseConn's site
// and host
// numbers, and entityNum as the entity number.
virtual DtEntityIdentifier makeEntityId(int entityNum);

// Return an event ID consisting of the DtExerciseConn's site and
// host
// numbers, and the current value of an event number counter.
virtual const DtEventID& nextEventId();

// ----- MISC -----

// Returns the current DtSocket that we are using for communica-
// tion.
virtual DtSocket *socket();

// Get the PDU factory used by netRead to create the right kind of
// derived DtPDU. You can use DtPduFactory member function,
// including
// its assignment operator to configure it to "know" about
// various
// PDU classes. For example, you can use
// *(exConn.pduFactory()) = DtDisPduFactory();
// to configure the PDU factory to know about all DIS PDUs and the
// VR-Link classes that represent them.
virtual DtPduFactory *pduFactory();

// Rather than just modifying the ExerciseConn's current PDU
// factory,
// you can tell the ExerciseConn to use a different PDU factory
// object.
// The object passed belongs to the ExerciseConn, and should not
// be deleted
// until after a subsequent call to setPduFactory, or the Exerci-
// seConn
// is deleted.
virtual void setPduFactory(DtPduFactory* fact);

```

```
// Enable/disable PDU filtering.
virtual void enableFiltering();
virtual void disableFiltering();

// Set/Get time stamp type (DtTimeStampAbsolute or DtTimeStamp-
Relative)
virtual void setTimeStampType(DtTimeStampType relOrAbs);
virtual DtTimeStampType timeStampType() const;

// This function has been obsoleted by the version above
// that use the DtTimeStampType enumeration, but is included
// for backwards compatibility.
virtual void setTimeStampType(int relOrAbs);

public:

// Returns a string indicating the VR-Link version number
static const char* vrlinkVersion();

static int nSizeMismatchComplain;

protected:

// copy constructor - not really implemented
DtExerciseConn(const DtExerciseConn& orig);

// assignment operator - not really implemented
DtExerciseConn& operator=(const DtExerciseConn& orig);

// Return time to send with outgoing messages
// (default is sim time if absolute timestamping, real time
// otherwise.
virtual DtTime currentTimeForStamping();

// Called by constructor
void init0(int port, int exerId, int siteId, int applNum, int v);
void init0(DtSocket *s, int exerId, int siteId, int applNum, int
v);
void init(DtSocket *s, int deleteSkt,
int exerId, int siteId, int applNum, int v);

protected:

DtExerciseId exid;
DtSimulationAddress appid;
int availEventNum;
int deleteSocket;
DtSocket* mysock;
DtPduCallbackManager myPduCbMgr;
int pduInterest[DtMAX_PDU_KINDS + 1];
```

```
void zeroPduInterestList();
int nSizeMismatch;
DtBoolean filtering;
DtHashList mcastAddrs;
DtPduFactory* myPduFact;
DtPduFactory* myDfltPduFact;
DtEntityIdentifier myCurrentId;
DtEventID myCurrentEventId;

DtTimeStampType myRelOrAbs;
};
// ----- INLINES -----

#endif // !DtHLA
```

dInteraction.h

```
#if !DtHLA

#include "pdu.h"

#include "chooseDefines.h"

// class DtInteraction:
//
// Instances of DtInteraction are used for representing DIS
// events.

class DT_DLL_VL DtInteraction : public DtPdu
{
public:

    // default constructor
    DtInteraction();

    // destructor
    virtual ~DtInteraction();

    // copy constructor
    DtInteraction(const DtInteraction& orig);

    // assignment operator
    DtInteraction& operator=(const DtInteraction& orig);

};
#endif
```


directRotRate.h

```
// rotation rate given by single rate and axis of rotation
class DT_DLL_MATRIX DtDirectedRotRate
{
public:
    DtDirectedRotRate();
    DtDirectedRotRate(DtConstVector pqr);

    void setFromVec(DtConstVector pqr);

    double thetaDot; /* radians per sec */
    double n[3];      /* axis unit vector */
};
// Inline functions
```

dirRtiAmb.h

```
#if DtHLA

#include "MachineTypes.h"
#ifndef DtDirectRtiAmbassador_H_
#define DtDirectRtiAmbassador_H_

#include "RTI.hh"
#include "vrlRtiAmb.h"

// class DtDirectRtiAmbassador:
//
// Instances of DtDirectRtiAmbassador are for wrapping the calls to
// The RTI ambassador so that we may override them.

class DT_DLL_VL DtDirectRtiAmbassador : public DtVrlRtiAmbassador
{
public:

    // default constructor
    DtDirectRtiAmbassador();

    // destructor
    virtual ~DtDirectRtiAmbassador();

    // Get the real RTI ambassador.
    virtual RTI::RTIambassador* rtiAmb();

    // Set/Get whether to reflect back local objects and interac-
    tions, if
    // supported by the derived class. The set call's return value
    indicates
    // the value of the flag as a result of the call.
    virtual DtBoolean setReflecting(DtBoolean onOff);
    virtual DtBoolean reflecting() const;

    // Calls down to the RTI function of the same name.
    virtual void createFederationExecution (
        const char *executionName,
        const char *FED);

    // Calls down to the RTI function of the same name.
    virtual void destroyFederationExecution (
        const char *executionName);

    // Calls down to the RTI function of the same name.
    virtual RTI::FederateHandle joinFederationExecution (
        const char* yourName,
```

```
    const char* executionName,  
    RTI::FederateAmbassadorPtr federateAmbassadorReference);  
  
    // Calls down to the RTI function of the same name.  
    virtual void resignFederationExecution(  
        RTI::ResignAction theAction);  
  
    // Calls down to the RTI function of the same name.  
    virtual void registerFederationSynchronizationPoint(  
        const char* label,  
        const char* theTag);  
  
    // Calls down to the RTI function of the same name.  
    virtual void registerFederationSynchronizationPoint(  
        const char* label,  
        const char* theTag,  
        const RTI::FederateHandleSet& syncSet);  
  
    // Calls down to the RTI function of the same name.  
    virtual void synchronizationPointAchieved(  
        const char* label);  
  
    // Calls down to the RTI function of the same name.  
    virtual void requestFederationSave (  
        const char* label,  
        const RTI::FedTime& theTime);  
  
    // Calls down to the RTI function of the same name.  
    virtual void requestFederationSave (  
        const char *label);  
  
    // Calls down to the RTI function of the same name.  
    virtual void federateSaveBegun();  
  
    // Calls down to the RTI function of the same name.  
    virtual void federateSaveComplete();  
  
    // Calls down to the RTI function of the same name.  
    virtual void federateSaveNotComplete();  
  
    // Calls down to the RTI function of the same name.  
    virtual void requestFederationRestore(const char* label);  
  
    // Calls down to the RTI function of the same name.  
    virtual void federateRestoreComplete();  
  
    // Calls down to the RTI function of the same name.  
    virtual void federateRestoreNotComplete();  
  
    // Calls down to the RTI function of the same name.
```

```
virtual void publishObjectClass(
    RTI::ObjectClassHandle theClass,
    const RTI::AttributeHandleSet& attributeList);

// Calls down to the RTI function of the same name.
virtual void unpublishObjectClass(
    RTI::ObjectClassHandle theClass);

// Calls down to the RTI function of the same name.
virtual void publishInteractionClass(
    RTI::InteractionClassHandle theInteraction);

// Calls down to the RTI function of the same name.
virtual void unpublishInteractionClass(
    RTI::InteractionClassHandle theInteraction);

// Calls down to the RTI function of the same name.
virtual void subscribeObjectClassAttributes(
    RTI::ObjectClassHandle theClass,
    const RTI::AttributeHandleSet& attributeList,
    RTI::Boolean active = RTI::RTI_TRUE);

// Calls down to the RTI function of the same name.
virtual void unsubscribeObjectClass(
    RTI::ObjectClassHandle theClass);

// Calls down to the RTI function of the same name.
virtual void subscribeInteractionClass(
    RTI::InteractionClassHandle theClass,
    RTI::Boolean active = RTI::RTI_TRUE);

// Calls down to the RTI function of the same name.
virtual void unsubscribeInteractionClass(
    RTI::InteractionClassHandle theClass);

// Calls down to the RTI function of the same name.
virtual RTI::ObjectHandle registerObjectInstance(
    RTI::ObjectClassHandle theClass, const char* theObject);

// Calls down to the RTI function of the same name.
virtual RTI::ObjectHandle registerObjectInstance(
    RTI::ObjectClassHandle theClass);

// Calls down to the RTI function of the same name.
virtual RTI::EventRetractionHandle updateAttributeValues(
    RTI::ObjectHandle theObject,
    const RTI::AttributeHandleValuePairSet& theAttributes,
    const RTI::FedTime& theTime,
    const char* theTag);
```

```
// Calls down to the RTI function of the same name.
virtual void updateAttributeValues(
    RTI::ObjectHandle theObject,
    const RTI::AttributeHandleValuePairSet& theAttributes,
    const char* theTag);

// Calls down to the RTI function of the same name.
virtual RTI::EventRetractionHandle sendInteraction(
    RTI::InteractionClassHandle theInteraction,
    const RTI::ParameterHandleValuePairSet& theParameters,
    const RTI::FedTime& theTime,
    const char* theTag);

// Calls down to the RTI function of the same name.
virtual void sendInteraction(
    RTI::InteractionClassHandle theInteraction,
    const RTI::ParameterHandleValuePairSet& theParameters,
    const char* theTag);

// Calls down to the RTI function of the same name.
virtual RTI::EventRetractionHandle deleteObjectInstance(
    RTI::ObjectHandle theObject,
    const RTI::FedTime& theTime,
    const char* theTag);

// Calls down to the RTI function of the same name.
virtual void deleteObjectInstance(
    RTI::ObjectHandle theObject,
    const char* theTag);

// Calls down to the RTI function of the same name.
virtual void localDeleteObjectInstance(
    RTI::ObjectHandle theObject);

// Calls down to the RTI function of the same name.
virtual void changeAttributeTransportationType (
    RTI::ObjectHandle theObject,
    const RTI::AttributeHandleSet& theAttributes,
    RTI::TransportationHandle theType);

// Calls down to the RTI function of the same name.
virtual void changeInteractionTransportationType (
    RTI::InteractionClassHandle theClass,
    RTI::TransportationHandle theType);

// Calls down to the RTI function of the same name.
virtual void requestObjectAttributeValueUpdate (
    RTI::ObjectHandle theObject,
    const RTI::AttributeHandleSet& theAttributes);
```

```
// Calls down to the RTI function of the same name.
virtual void requestClassAttributeValueUpdate (
    RTI::ObjectClassHandle theClass,
    const RTI::AttributeHandleSet& theAttributes);

// Calls down to the RTI function of the same name.
virtual void unconditionalAttributeOwnershipDivestiture(
    RTI::ObjectHandle theObject,
    const RTI::AttributeHandleSet& theAttributes);

// Calls down to the RTI function of the same name.
virtual void negotiatedAttributeOwnershipDivestiture(
    RTI::ObjectHandle theObject,
    const RTI::AttributeHandleSet& theAttributes,
    const char* theTag);

// Calls down to the RTI function of the same name.
virtual void attributeOwnershipAcquisition(
    RTI::ObjectHandle theObject,
    const RTI::AttributeHandleSet& desiredAttributes,
    const char* theTag);

// Calls down to the RTI function of the same name.
virtual void attributeOwnershipAcquisitionIfAvailable(
    RTI::ObjectHandle theObject,
    const RTI::AttributeHandleSet& desiredAttributes);

// Calls down to the RTI function of the same name.
virtual RTI::AttributeHandleSet* attributeOwnershipReleaseRe-
sponse(
    RTI::ObjectHandle theObject,
    const RTI::AttributeHandleSet& theAttributes);

// Calls down to the RTI function of the same name.
virtual void cancelNegotiatedAttributeOwnershipDivestiture(
    RTI::ObjectHandle theObject,
    const RTI::AttributeHandleSet& theAttributes);

// Calls down to the RTI function of the same name.
virtual void cancelAttributeOwnershipAcquisition(
    RTI::ObjectHandle theObject,
    const RTI::AttributeHandleSet& theAttributes);

// Calls down to the RTI function of the same name.
virtual void queryAttributeOwnership(
    RTI::ObjectHandle theObject,
    RTI::AttributeHandle theAttribute);

// Calls down to the RTI function of the same name.
virtual RTI::Boolean isAttributeOwnedByFederate(
```

```
RTI::ObjectHandle theObject,  
RTI::AttributeHandle theAttribute);  
  
// Calls down to the RTI function of the same name.  
virtual void enableTimeRegulation (  
    const RTI::FedTime& theFederateTime,  
    const RTI::FedTime& theLookahead);  
  
// Calls down to the RTI function of the same name.  
virtual void disableTimeRegulation();  
  
// Calls down to the RTI function of the same name.  
virtual void enableTimeConstrained();  
  
// Calls down to the RTI function of the same name.  
virtual void disableTimeConstrained();  
  
// Calls down to the RTI function of the same name.  
virtual void timeAdvanceRequest(  
    const RTI::FedTime& theTime);  
  
// Calls down to the RTI function of the same name.  
virtual void timeAdvanceRequestAvailable (  
    const RTI::FedTime& theTime);  
  
// Calls down to the RTI function of the same name.  
virtual void nextEventRequest(  
    const RTI::FedTime& theTime);  
  
// Calls down to the RTI function of the same name.  
virtual void nextEventRequestAvailable (  
    const RTI::FedTime& theTime);  
  
// Calls down to the RTI function of the same name.  
virtual void flushQueueRequest (  
    const RTI::FedTime& theTime);  
  
// Calls down to the RTI function of the same name.  
virtual void enableAsynchronousDelivery();  
  
// Calls down to the RTI function of the same name.  
virtual void disableAsynchronousDelivery();  
  
// Calls down to the RTI function of the same name.  
virtual void queryLBTS(RTI::FedTime& theTime);  
  
// Calls down to the RTI function of the same name.  
virtual void queryFederateTime(RTI::FedTime& theTime);  
  
// Calls down to the RTI function of the same name.
```

```
virtual void queryMinNextEventTime (
    RTI::FedTime& theTime);

// Calls down to the RTI function of the same name.
virtual void modifyLookahead (
    const RTI::FedTime& theLookahead);

// Calls down to the RTI function of the same name.
virtual void queryLookahead (
    RTI::FedTime& theTime);

// Calls down to the RTI function of the same name.
virtual void retract(
    RTI::EventRetractionHandle theHandle);

// Calls down to the RTI function of the same name.
virtual void changeAttributeOrderType(
    RTI::ObjectHandle theObject,
    const RTI::AttributeHandleSet& theAttributes,
    RTI::OrderingHandle theType);

// Calls down to the RTI function of the same name.
virtual void changeInteractionOrderType(
    RTI::InteractionClassHandle theClass,
    RTI::OrderingHandle theType);

// Calls down to the RTI function of the same name.
virtual void createRegion (
    RTI::SpaceHandle theSpace,
    RTI::ULong numberOfExtents);

// Calls down to the RTI function of the same name.
virtual void notifyAboutRegionModification (
    RTI::Region &theRegion);

// Calls down to the RTI function of the same name.
virtual void deleteRegion(RTI::Region *theRegion);

// Calls down to the RTI function of the same name.
virtual RTI::ObjectHandle registerObjectInstanceWithRegion(
    RTI::ObjectClassHandle theClass,
    const char* theObject,
    RTI::AttributeHandle theAttributes[],
    RTI::Region* theRegions[],
    RTI::ULong theNumberOfHandles);

// Calls down to the RTI function of the same name.
virtual RTI::ObjectHandle registerObjectInstanceWithRegion(
    RTI::ObjectClassHandle theClass,
    RTI::AttributeHandle theAttributes[],
```



```
RTI::Region* theRegions[],
RTI::ULong theNumberOfHandles);

// Calls down to the RTI function of the same name.
virtual void associateRegionForUpdates (
    RTI::Region& theRegion,
    RTI::ObjectHandle theObject,
    const RTI::AttributeHandleSet& theAttributes);

// Calls down to the RTI function of the same name.
virtual void unassociateRegionForUpdates(
    RTI::Region& theRegion,
    RTI::ObjectHandle theObject);

// Calls down to the RTI function of the same name.
virtual void subscribeObjectClassAttributesWithRegion(
    RTI::ObjectClassHandle theClass,
    RTI::Region& theRegion,
    const RTI::AttributeHandleSet &attributeList,
    RTI::Boolean active = RTI::RTI_TRUE);

// Calls down to the RTI function of the same name.
virtual void unsubscribeObjectClassWithRegion(
    RTI::ObjectClassHandle theClass,
    RTI::Region& theRegion);

// Calls down to the RTI function of the same name.
virtual void subscribeInteractionClassWithRegion (
    RTI::InteractionClassHandle theClass,
    RTI::Region& theRegion,
    RTI::Boolean active = RTI::RTI_TRUE);

// Calls down to the RTI function of the same name.
virtual void unsubscribeInteractionClassWithRegion (
    RTI::InteractionClassHandle theClass,
    RTI::Region& theRegion);

// Calls down to the RTI function of the same name.
virtual RTI::EventRetractionHandle sendInteractionWithRegion(
    RTI::InteractionClassHandle theInteraction,
    const RTI::ParameterHandleValuePairSet &theParameters,
    const RTI::FedTime& theTime,
    const char* theTag,
    const RTI::Region& theRegion);

// Calls down to the RTI function of the same name.
virtual void sendInteractionWithRegion(
    RTI::InteractionClassHandle theInteraction,
    const RTI::ParameterHandleValuePairSet &theParameters,
    const char* theTag,
```

```
    const RTI::Region& theRegion);

// Calls down to the RTI function of the same name.
virtual void requestClassAttributeValueUpdateWithRegion(
    RTI::ObjectClassHandle theClass,
    const RTI::AttributeHandleSet& theAttributes,
    const RTI::Region& theRegion);

// Calls down to the RTI function of the same name.
virtual RTI::ObjectClassHandle getObjectClassHandle (
    const char* theName);

// Calls down to the RTI function of the same name.
virtual char* getObjectClassName (
    RTI::ObjectClassHandle theHandle);

// Calls down to the RTI function of the same name.
virtual RTI::AttributeHandle getAttributeHandle (
    const char* theName,
    RTI::ObjectClassHandle whichClass);

// Calls down to the RTI function of the same name.
virtual char* getAttributeName (
    RTI::AttributeHandle theHandle,
    RTI::ObjectClassHandle whichClass);

// Calls down to the RTI function of the same name.
virtual RTI::InteractionClassHandle getInteractionClassHandle (
    const char* theName);

// Calls down to the RTI function of the same name.
virtual char* getInteractionClassName (
    RTI::InteractionClassHandle theHandle);

// Calls down to the RTI function of the same name.
virtual RTI::ParameterHandle getParameterHandle (
    const char* theName,
    RTI::InteractionClassHandle whichClass);

// Calls down to the RTI function of the same name.
virtual char* getParameterName (
    RTI::ParameterHandle theHandle,
    RTI::InteractionClassHandle whichClass);

// Calls down to the RTI function of the same name.
virtual RTI::ObjectHandle getObjectInstanceHandle(
    const char* theName);

// Calls down to the RTI function of the same name.
virtual char* getObjectInstanceName(
```

```
RTI::ObjectHandle theHandle);

// Calls down to the RTI function of the same name.
virtual RTI::SpaceHandle getRoutingSpaceHandle (
    const char* theName);

// Calls down to the RTI function of the same name.
virtual const char* getRoutingSpaceName (
    const RTI::SpaceHandle theHandle);

// Calls down to the RTI function of the same name.
virtual RTI::DimensionHandle getDimensionHandle(
    const char* theName,
    RTI::SpaceHandle whichSpace);

// Calls down to the RTI function of the same name.
virtual char* getDimensionName(
    RTI::DimensionHandle theHandle,
    RTI::SpaceHandle whichSpace);

// Calls down to the RTI function of the same name.
virtual RTI::SpaceHandle
    getAttributeRoutingSpaceHandle(
        RTI::AttributeHandle theHandle,
        RTI::ObjectClassHandle whichClass);

// Calls down to the RTI function of the same name.
virtual RTI::ObjectClassHandle getObjectClass (
    RTI::ObjectHandle theObject);

// Calls down to the RTI function of the same name.
virtual RTI::SpaceHandle getInteractionRoutingSpaceHandle(
    RTI::InteractionClassHandle theHandle);

// Calls down to the RTI function of the same name.
virtual RTI::TransportationHandle getTransportationHandle(
    const char *theName);

// Calls down to the RTI function of the same name.
virtual char* getTransportationName(
    RTI::TransportationHandle theHandle);

// Calls down to the RTI function of the same name.
virtual RTI::OrderingHandle getOrderingHandle(
    const char *theName);

// Calls down to the RTI function of the same name.
virtual char* getOrderingName(
    RTI::OrderingHandle theHandle);
```

```
// Calls down to the RTI function of the same name.
virtual void enableClassRelevanceAdvisorySwitch();

// Calls down to the RTI function of the same name.
virtual void disableClassRelevanceAdvisorySwitch();

// Calls down to the RTI function of the same name.
virtual void enableAttributeRelevanceAdvisorySwitch();

// Calls down to the RTI function of the same name.
virtual void disableAttributeRelevanceAdvisorySwitch();

// Calls down to the RTI function of the same name.
virtual void enableAttributeScopeAdvisorySwitch();

// Calls down to the RTI function of the same name.
virtual void disableAttributeScopeAdvisorySwitch();

// Calls down to the RTI function of the same name.
virtual void enableInteractionRelevanceAdvisorySwitch();

// Calls down to the RTI function of the same name.
virtual void disableInteractionRelevanceAdvisorySwitch();

// Calls down to the RTI function of the same name.
virtual RTI::Boolean tick();

// Calls down to the RTI function of the same name.
virtual RTI::Boolean tick(
    RTI::TickTime minimum,
    RTI::TickTime maximum);

// Calls down to the RTI function of the same name.
virtual RTI::RegionToken getRegionToken(RTI::Region*);

// Calls down to the RTI function of the same name.
virtual RTI::Region* getRegion(RTI::RegionToken);

public:

    // returns a new DtDirectRtiAmbassador
    static DtVrlRtiAmbassador* create();

protected:

    // copy constructor - not implemented
    DtDirectRtiAmbassador(const DtDirectRtiAmbassador& orig);

    // assignment operator - not implemented
```

```
    DtDirectRtiAmbassador& operator=(const DtDirectRtiAmbassador&  
orig);
```

```
protected:
```

```
    RTI::RTIambassador* myRtiAmb;  
    RTI::FederateAmbassadorPtr myFedAmb;  
    DtBoolean myReflecting;  
};  
#endif
```

disEnums.h

```
#include "packets/disVersion.h"

/***** IMPORTANT *****/
/* This file should contain only enumeration
   definitions. This file is processed by a program
   that generates the files parseEnum.cc and
   parseEnum.h. Therefore all enums must conform to the
   format of the ones already in this file. No comments
   on the same line as a member of the enum, and all
   enumeration names and member names must start with Dt.
                                   -Len                               */

#define DtEXTEND_ENUM(enumType, name, val) \
    const enumType name = enumType(val)

/* SECTION 1 */

enum DtDetonatorFuze
{
    DtFuzeOther= 0,
    DtFuzeContact  = 1000,
    DtFuzeContactInstant = 1100,
    DtFuzeContactDelayed = 1200,
    DtFuzeTimed = 2000,
    DtFuzeProximity = 3000,
    DtFuzeCommand = 4000,
    DtFuzeAltitude = 5000,
    DtFuzeDepth = 6000,
    DtFuzeAcoustic = 7000
};

enum DtWarheadType
{
    DtWarheadOther= 0,
    DtWarheadHighExplosive= 1000,
    DtWarheadHighExplosivePlastic= 1100,
    DtWarheadHighExplosiveIncendiary= 1200,
    DtWarheadHighExplosiveFragmentation= 1300,
    DtWarheadHighExplosiveAntiTank= 1400,
    DtWarheadHighExplosiveBomblets= 1500,
    DtWarheadHighExplosiveShapedCharge= 1600,
    DtWarheadSmoke= 2000,
    DtWarheadIllumination= 3000,
    DtWarheadPractice= 4000,
    DtWarheadKinetic= 5000,
    DtWarheadUnused= 6000,
    DtWarheadNuclear= 7000,
```

```

    DtWarheadChemicalGeneral= 8000,
    DtWarheadChemicalBlisterAgent= 8100,
    DtWarheadChemicalBloodAgent= 8200,
    DtWarheadChemicalNerveAgent= 8300,
    DtWarheadBiologicalGeneral= 9000
};

/* Section 2 */

enum DtRepairType
{
    DtNoRepairsPerformed,
    DtRepairEverything,
#ifdef DtDISVERS != 3
    DtDriveTrainMotor = 10,
    DtDriveTrainStarter = 20,
    DtDriveTrainAlternator = 30,
    DtDriveTrainGenerator = 40,
    DtDriveTrainBattery = 50,
    DtDriveTrainEngineCoolantLeak = 60,
    DtDriveTrainFuelFilter = 70,
    DtDriveTrainTransmissionOilLeak = 80,
    DtDriveTrainEngineOilLeak = 90,
    DtDriveTrainPumps = 100,
    DtDriveTrainFilters = 110,
#else
    DtPropulsionMotor = 10,
    DtPropulsionStarter = 20,
    DtPropulsionAlternator = 30,
    DtPropulsionGenerator = 40,
    DtPropulsionBattery = 50,
    DtPropulsionEngineCoolantLeak = 60,
    DtPropulsionFuelFilter = 70,
    DtPropulsionTransmissionOilLeak = 80,
    DtPropulsionEngineOilLeak = 90,
    DtPropulsionPumps = 100,
    DtPropulsionFilters = 110,
#endif
    DtBodyHull = 1000,
    DtBodyAirframe = 1010,
    DtInterfacePropeller = 1500,
#ifdef DtDISVERS != 3
    DtDriveTrainTransmission = 120,
    DtDriveTrainBrakes = 130,
#else
    DtInterfaceTransmission = 1510,
    DtInterfaceBrakes = 1530,
#endif
    DtInterfaceFilters = 1520,
    DtWeaponGunElevationDrive = 2000,

```

```
DtWeaponGunStabilizationSys = 2010,
DtWeaponGunneryPrimarySight = 2020,
DtWeaponCommandersGPSExtension = 2030,
DtWeaponLoadingMechanism = 2040,
DtFuelTransferPump = 4000,
DtFuelLines = 4010,
DtFuelGauges = 4020,
#if DtDISVERS != 3
DtElectronicWarfareSys = 4500,
DtElectronicDetectionSys = 4600,
DtElectronicRangeFinder = 4700,
DtElectronicSys = 4800,
DtElectronicRadios = 5000,
DtElectronicIntercoms = 5100,
DtElectronicLasers = 4840,
DtElectronicCoders = 5200,
DtElectronicDecoders = 5300,
DtElectronicComputers = 5500,
#else
DtElectronicWarfareSys = 4600,
DtElectronicDetectionSys = 5600,
DtElectronicLaserRangeFinder = 4700,
DtElectronicCounterMeasureSys = 4500,
DtElectronicRadios = 4800,
DtElectronicIntercoms = 4900,
DtElectronicLasers = 5200,
DtElectronicCoders = 5000,
DtElectronicDecoders = 5100,
DtElectronicEmitters = 5400,
DtElectronicComputers = 5300,
#endif
DtLifeSupportAirSupply = 8000,
DtLifeSupportFilters = 8010,
DtLifeSupportWaterSupply = 8020,
DtLifeSupportRefrigerationSys = 8030,
DtLifeSupportCBRProtection = 8040,
DtLifeSupportWaterWashSys = 8050,
DtLifeSupportDecontaminationSys = 8060,
DtActuatorWaterSupply = 9000,
DtActuatorCoolingSys = 9010,
DtActuatorWinches = 9020,
DtActuatorCatapults = 9030,
DtActuatorCranes = 9040,
DtActuatorLaunchers = 9050,
DtAuxCraftLifeBoat = 10000,
DtAuxCraftLandingCraft = 10010,
DtAuxCraftEjectionSeat = 10020
};

#if DtDISVERS >= 5
```



```

enum DtCollisionType
{
    // Elastic and Inelastic were incorrect prior to 3.1

    DtInelastic= 0,
    DtElastic= 1,

    // Other collision type is no longer a valid DIS enumeration.

    DtCollisionTypeOther= 2

};
#endif

/* Section 4 */

enum DtForceType
{
    DtForceOther,
    DtForceFriendly,
    DtForceOpposing,
    DtForceNeutral
};

enum DtDetonationResult
{
    DtDetResOther          = 0,
    DtDetResEntityImpact   = 1,
    DtDetResEntityProximate= 2,
    DtDetResGroundImpact   = 3,
    DtDetResGroundProximate= 4,
    DtDetResDetonation= 5,
    #if DtDISVERS == 3
        DtDetResNone          = 6
    #else
        DtDetResNone          = 6,
        DtDetResHE_HitSmall    = 7,
        DtDetResHE_HitMedium   = 8,
        DtDetResHE_HitLarge    = 9,
        DtDetResArmorPiercingHit = 10,
        DtDetResDirtBlastSmall  = 11,
        DtDetResDirtBlastMedium = 12,
        DtDetResDirtBlastLarge  = 13,
        DtDetResWaterBlastSmall = 14,
        DtDetResWaterBlastMedium = 15,
        DtDetResWaterBlastLarge = 16,
        DtDetResAirHit          = 17,
        DtDetResBuildingHitSmall = 18,
        DtDetResBuildingHitMedium = 19,
    #endif
};

```

```
DtDetResBuildingHitLarge      = 20,
DtDetResMineClearingLineCharge = 21,
DtDetResEnvironmentObjectImpact = 22,
DtDetResEnvironmentObjectProximate = 23,
DtDetResWaterImpact          = 24,
DtDetResAirBurst              = 25
#endif
};

enum DtPduKind
{
    DtOtherPduKind = 0,
    DtEntityStatePduKind = 1,
    DtFirePduKind = 2,
    DtDetonationPduKind = 3,
    DtCollisionPduKind = 4,
    DtServiceRequestPduKind = 5,
    DtResupplyOfferPduKind = 6,
    DtResupplyReceivedPduKind = 7,
    DtResupplyCancelPduKind = 8,
    DtRepairCompletePduKind = 9,
    DtRepairResponsePduKind = 10,
    DtCreateEntityPduKind = 11,
    DtRemoveEntityPduKind = 12,
    DtStartResumePduKind = 13,
    DtStopFreezePduKind = 14,
    DtAcknowledgePduKind = 15,
    DtActionRequestPduKind = 16,
    DtActionResponsePduKind = 17,
    DtDataQueryPduKind = 18,
    DtSetDataPduKind = 19,
    DtDataPduKind = 20,
    DtEventReportPduKind = 21,
    #if DtDISVERS >= 5
        DtCommentPduKind = 22,
    #endif
    DtMessagePduKind = 22,
    #if DtDISVERS != 3
        DtElectromagneticEmissionPduKind = 23,
        DtDesignatorPduKind = 24,
    #else
        DtEmissionPduKind = 23,
        DtLaserPduKind = 24,
    #endif
    DtTransmitterPduKind = 25,
    DtSignalPduKind = 26,
    DtReceiverPduKind = 27,

    // DMO enums used by 2.1.4 PDUs
    DtIffAtcNav aidsPduKind = 28,
```

```

    DtUnderwaterAcousticPduKind =      29,
    DtSuppEmisEntStatePduKind =      30,
    DtIntercomSignalPduKind =      31,
    DtIntercomControlPduKind =      32,
    DtAggregateStatePduKind =      33,
    DtIsGroupOfPduKind =      34,
    DtTransferControlRequestPduKind = 35,
    DtIsPartOfPduKind =      36,
    DtMinefieldStatePduKind =      37,
    DtMinefieldQueryPduKind =      38,
    DtMinefieldDataPduKind =      39,
    DtMinefieldResponseNackPduKind = 40,
    DtEnvironmentalProcessPduKind = 41,
    DtGriddedDataPduKind =      42,
    DtPointObjectStatePduKind =      43,
    DtLinearObjectStatePduKind =      44,
    DtAreaObjectStatePduKind =      45,
    DtLeTspiPduKind =      46,
    DtLeAppearancePduKind =      47,
    DtLeArticulatedPartsPduKind = 48,
    DtLeFirePduKind =      49,
    DtLeDetonationPduKind =      50,
    DtCreateEntityRPduKind =      51,
    DtRemoveEntityRPduKind =      52,
    DtStartResumeRPduKind =      53,
    DtStopFreezerRPduKind =      54,
    DtAcknowledgeRPduKind =      55,
    DtActionRequestRPduKind =      56,
    DtActionResponseRPduKind =      57,
    DtDataQueryRPduKind =      58,
    DtSetDataRPduKind =      59,
    DtDataRPduKind =      60,
    DtEventReptRPduKind =      61,
    DtCommentRPduKind =      62,
    DtRecordQueryPduKind =      63,
    DtSetRecordPduKind =      64,
    DtRecordPduKind =      65,
    DtCollisionElasticPduKind =      66,
    DtEntityStateUpdatePduKind =      67
};

const DtPduKind DtNPCollisionPduKind = DtPduKind(200);
const DtPduKind DtNPFieldPduKind = DtPduKind(201);
const DtPduKind DtViewControlPduKind = DtPduKind(202);
const DtPduKind DtMOPMigratorPduKind = DtPduKind(203);
const DtPduKind DtNPSurfaceContactPduKind = DtPduKind(204);
const DtPduKind DtGenericPduKind = DtPduKind(254);
const DtPduKind DtLgrControlPduKind = DtPduKind(205);

/* backwards compatible */

```

```
#define DtViewControlKind DtViewControlPduKind

#if DtDISVERS != 3
enum DtProtocolFamily
{
    DtFamilyOther,
    DtFamilyEntityInteraction,
    DtFamilyWarfare,
    DtFamilyLogistics,
    DtFamilyRadioCommunication,
    DtFamilySimulationManagement,
    DtFamilyDistributedEmissionRegeneration,
    DtFamilyEntityManagement,
    DtFamilyMinefield,
    DtFamilySyntheticEnvironment,
    DtFamilySimulationManagementWithReliability,
    DtFamilyLiveEntity,
    DtFamilyNonRealTime
};
#endif

enum DtRepairResult
{
    DtRepairResultOther,
    DtRepairResultEnded,
    DtRepairResultInvalid,
    DtRepairResultInterrupted,
    DtRepairResultCancelled
};

enum DtServiceType
{
    DtServiceTypeOther,
    DtServiceTypeResupply,
    DtServiceTypeRepair
};

enum DtStopReason
{
    DtStopReasonOther,
    DtStopReasonRecess,
    DtStopReasonTermination,
    DtStopReasonSystemFailure,
    DtStopReasonSecurityViolation,
    DtStopReasonEntityReconstitution
};

enum DtAcknowledgeFlag
{
    DtAckCreate = 1,
```

```
DtAckRemove,  
DtAckStart,  
DtAckStop  
};  
  
enum DtResponseFlag  
{  
    DtNoWay,  
    DtSureThing  
};  
  
enum DtActionId  
{  
    DtActionOther,  
    DtActionLocalStorage,  
    DtActionOutOfAmmo,  
    DtActionKilledInAction,  
    DtActionDamage,  
    DtActionMobilityDisabled,  
    DtActionFireDisabled,  
    DtActionOutOfFuel,  
    DtActionRecallCheckpointData,  
    DtActionRecallInitialParameters,  
    DtActionInitiateTetherLead,  
    DtActionInitiateTetherFollow,  
    DtActionUntether,  
    DtActionInitiateServiceStationResupply,  
    DtActionInitiateTailgateResupply,  
    DtActionInitiateHitchLead,  
    DtActionInitiateHitchFollow,  
    DtActionUnhitch,  
    DtActionMount,  
    DtActionDismount,  
    DtActionStartDRC,  
    DtActionStopDRC,  
    DtActionDataQuery,  
    DtActionStatusRequest,  
    DtActionSendObjectStateData,  
    DtActionReconstitute,  
    DtActionLockSiteConfiguration,  
    DtActionUnlockSiteConfiguration,  
    DtActionUpdateSiteConfiguration,  
    DtActionQuerySiteConfiguration,  
    DtActionTetheringInformation,  
    DtActionMountIntent,  
    DtActionOther2,  
    DtActionAcceptSubscription,  
    DtActionUnsubscribe,  
    DtActionTeleportEntity,  
    DtActionChangeAggregateState,
```

```
DtActionRequestStartPDU,  
DtActionWakeUpGetReadyForInitialization,  
DtActionInitializeInternalParameters,  
DtActionSendPlanData,  
DtActionSynchronizeInternalClocks,  
DtActionRun,  
DtActionSaveInternalParameters,  
DtActionSimulateMalfunction,  
DtActionJoinExercise,  
DtActionResignExercise,  
DtActionTimeAdvance  
};
```

```
enum DtRequestStatus  
{  
    DtRequestStatusOther,  
    DtRequestStatusPending,  
    DtRequestStatusExecuting,  
    DtRequestStatusPartial,  
    DtRequestStatusComplete,  
    DtRequestStatusRejected,  
    DtRequestStatusRetransmitNow,  
    DtRequestStatusRetransmitLater,  
    DtRequestStatusInvalidTimeParameters,  
    DtRequestStatusSimulationTimeExceeded,  
    DtRequestStatusRequestDone  
};
```

```
enum DtEventType  
{  
    DtEventOther,  
    DtEventOutOfFuel,  
    DtEventOutOfAmmo,  
    DtEventKilledInAction,  
    DtEventDamage  
};
```

```
#if DtDISVERS != 3  
enum DtDatumParam  
{  
    DtDatumMass= 26000,  
    DtDatumForceId= 11200,  
  
    DtDatumKind= 11110,  
    DtDatumDomain= 11120,  
    DtDatumCountry= 11130,  
    /* missing from spec */  
    DtDatumCategory= 11140,  
    DtDatumSubCategory= 11150,  
    DtDatumSpecific= 11160,
```

```
DtDatumExtra= 11170,

DtDatumAltKind= 12110,
DtDatumAltDomain= 12120,
DtDatumAltCountry= 12130,
/* missing from spec */
DtDatumAltCategory= 12140,
DtDatumAltSubCategory = 12150,
DtDatumAltSpecific= 12160,
DtDatumAltExtra= 12170,

DtDatumXpos= 31210,
DtDatumYpos= 31220,
DtDatumZpos= 31230,
#if !PCFTP
DtDatumXvel= 36100,
DtDatumYvel= 36200,
DtDatumZvel= 36300,
#endif
DtDatumPhi= 32500,
DtDatumTheta= 32600,
DtDatumPsi= 32700,

#if !PCFTP
DtDatumDrAlgorithm= 71000,

DtDatumXaccel= 37100,
DtDatumYaccel= 37200,
DtDatumZaccel= 37300,
DtDatumXangvel= 100001,
DtDatumYangvel= 100002,
DtDatumZangvel= 100003,

DtDatumAppearance= 33000,
#endif

/* character set */
DtDatumMarkingCS= 13100,
/* 11 bytes */
DtDatumMarking= 13000,

#if !PCFTP
DtDatumNumArtParam= 90000,
DtDatumArtParamId= 100014,
DtDatumArtParamType= 100047,
DtDatumArtParamValue = 100048,
DtDatumAmountAmmoSupplies= 27000,
DtDatumAmmoQuantity= 24000,
DtDatumFuelQuantity= 23100,
DtDatumRadarSysStatus= 61400,
```

```
DtDatumRadioCommSysStatus= 62100,
DtDatumStationaryRadioTime= 62200,
DtDatumMovingRadioTime= 62300,
DtDatumBodyPartDamagedRatio= 35000,
DtDatumTerrainDatabase= 41000,
DtDatumLoggerFile= 43000,
#endif
    DtDatumCapability= 34

};

#else

enum DtDatumParam
{
    DtDatumMass= 1,
    DtDatumForceId= 2,

    DtDatumKind= 3,
    DtDatumDomain= 4,
    DtDatumCountry= 5,
    /* missing from spec */
    DtDatumCategory= 128,
    DtDatumSubCategory= 6,
    DtDatumSpecific= 7,
    DtDatumExtra= 8,

    DtDatumAltKind= 9,
    DtDatumAltDomain= 10,
    DtDatumAltCountry= 11,
    /* missing from spec */
    DtDatumAltCategory= 129,
    DtDatumAltSubCategory = 12,
    DtDatumAltSpecific= 13,
    DtDatumAltExtra= 14,

    DtDatumXpos= 15,
    DtDatumYpos= 16,
    DtDatumZpos= 17,
    DtDatumXvel= 18,
    DtDatumYvel= 19,
    DtDatumZvel= 20,
    DtDatumPhi= 21,
    DtDatumTheta= 22,
    DtDatumPsi= 23,

    DtDatumDrAlgorithm= 24,

    DtDatumXaccel= 25,
    DtDatumYaccel= 26,
```



```

    DtDatumZaccel= 27,
    DtDatumXangvel= 28,
    DtDatumYangvel= 29,
    DtDatumZangvel= 30,

    DtDatumAppearance= 31,
    /* character set */
    DtDatumMarkingCS= 32,
    /* 11 bytes */
    DtDatumMarking= 33,
    DtDatumCapability= 34,
    DtDatumNumArtParam= 35,
    DtDatumArtParamId= 36,
    DtDatumArtParamType= 37,
    DtDatumArtParamValue = 38,
    DtDatumAmountAmmoSupplies= 39,
    DtDatumAmmoQuantity= 40,
    DtDatumFuelQuantity= 41,
    DtDatumRadarSysStatus= 42,
    DtDatumRadioCommSysStatus= 43,
    DtDatumStationaryRadioTime= 44,
    DtDatumMovingRadioTime= 45,
    DtDatumBodyPartDamagedRatio= 46,
    DtDatumTerrainDatabase= 47,
    DtDatumLoggerFile= 48
};

#endif

enum DtEmissionPduType
{
    DtStateUpdate,
    DtChangeDataUpdate
};

enum DtEmitterSysName
{
    DtEmitterSysOther
};

enum DtEmissionFunc
{
    DtEmissionFuncOther = 0,
    DtEmissionFuncMultiFunction = 1,
    DtEmissionFuncEarlyWarningSurveillance = 2,
    DtEmissionFuncHeightFinding = 3,
    DtEmissionFuncFireControl = 4,
    DtEmissionFuncAcquisitionDetection = 5,
    DtEmissionFuncTracking = 6,
    DtEmissionFuncGuidanceIllumination = 7,

```

```
DtEmissionFuncFiringPointLaunchPointLocation = 8,  
DtEmissionFuncRanging = 9,  
DtEmissionFuncRadarAltimeter = 10,  
DtEmissionFuncImaging = 11,  
DtEmissionFuncMotionDetection = 12,  
DtEmissionFuncNavigation = 13,  
DtEmissionFuncWeather = 14,  
DtEmissionFuncInstrumentation = 15,  
DtEmissionFuncIdentificationClassification = 16,  
DtEmissionFuncJammingNoise = 64,  
DtEmissionFuncJammingDeception = 65,  
DtEmissionFuncDecoy = 66,  
DtEmissionFuncWeaponNonLethal = 96,  
DtEmissionFuncWeaponLethal = 97  
};
```

```
enum DtBeamFunc  
{  
    DtBeamFuncOther = 0,  
    DtBeamFuncSearch = 1,  
    DtBeamFuncHeightFinder = 2,  
    DtBeamFuncAcquisition = 3,  
    DtBeamFuncTracking = 4,  
    DtBeamFuncAcquisitionAndTracking = 5,  
    DtBeamFuncCommandGuidance = 6,  
    DtBeamFuncIllumination = 7,  
    DtBeamFuncRangeOnlyRadar = 8,  
    DtBeamFuncMissileBeacon = 9,  
    DtBeamFuncMissileFuze = 10,  
    DtBeamFuncActiveRadarMissileSeeker = 11,  
    BeamFuncJammer = 12  
};
```

```
enum DtTrackingJamming  
{  
    DtNotTrackingAllTargets,  
    DtTrackingAllTargets  
};
```

```
enum DtReferenceSys  
{  
    DtReferenceSysWorldCoordinates = 1,  
    DtReferenceSysEntityCoordinates = 2  
};
```

```
enum DtDesignatorCodeName  
{  
    DtDesignatorNameOther,  
    DtDesignatorNameTBD  
};
```

```
enum DtDesignatorCode
{
    DtDesignatorCodeOther
};

#if DtDISVERS == 3
enum DtLaserCodeName
{
    DtLaserNameOther
};

enum DtLaserCode
{
    DtLaserCodeOther
};
#endif /* DtDISVERS */

#define DtRadioOther 0
enum DtRadioTransmitState
{
    DtOff,
    DtRadioOnNotTrasmitting,
    DtOn
};

enum DtRadioInputSource
{
    DtRadioInputOther,
    DtPilot,
    DtCoPilot,
    DtFirstOfficer,
    DtDriver,
    DtLoader,
    DtGunner,
    DtCommander,
    DtDigitalDataDevice,
    DtIntercom
};

enum DtAntennaPatternType
{
    DtOmniDirectional,
    DtBeam,
    DtSphericalHarmonic
};

enum DtRadioSpreadSpectrum
{
    DtRadioSpreadOther,
```

```
    DtFrequencyHopping,
    DtPseudoNoise,
    DtTimeHopping,
    DtPseudoNoiseFrequencyHopping,
    DtFrequencyHoppingTimeHopping,
    DtPseudoNoiseTimeHopping,
    DtPseudoNoiseFrequencyHoppingTimeHopping
};

enum DtRadioMajorModulation
{
    DtRadioMajorModOther,
    DtAmplitude,
    DtAmplitudeAndAngle,
#ifdef DtDISVERS != 3
    DtAngle = 3,
    DtCombination = 4,
    DtPulse = 5,
    DtUnmodulated = 6
#else
    DtCombination = 3,
    DtPulse = 4,
    DtUnmodulated = 5
#endif
};

enum DtRadioDetailedModulationAmplitude
{
    DtRadioDetailModOther,
    DtAfsk,
    DtAm,
    DtCw,
    DtDsb,
    DtIsb,
    DtLsb,
    DtSsbFull,
    DtSsbReduc,
    DtUsb,
    DtVsb
};

typedef DtRadioDetailedModulationAmplitude DtRadioDetailedModulation;

enum DtDetailedModulationAmplitudeAngle
{
    DtRadioModAmpAngOther,
    DtDetailedAmplitudeAndAngle
};
```

```
enum DtDetailedModulationAngle
{
    DtRadioModeAngleOther,
    DtFm,
    DtFsk,
    DtPm
};

enum DtDetailedModulationCombination
{
    DtRadioDetailModCombo,
    DtAmplitudeAnglePulse
};

enum DtDetailedModulationUnmodulated
{
    DtRadioDetailModUnmod,
    DtContinuous = 1
};

enum DtDetailedModulationType
{
    DtDetailedModulationDummy
};

typedef DtDetailedModulationUnmodulated DetailedModulationUnmodulated;

enum DtRadioSystem
{
    DtRadioSystemOther,
    DtGeneric,
    DtHq,
    DtHqII,
    DtHqIIA,
    DtSincgards
};

enum DtCryptoMode
{
    DtBasebandEncryption,
    DtDiphaseEncryption
};

enum DtRadioCryptoSystem
{
    DtRadioCryptoSysOther,
    DtRadioCryptoSysKY_28,
    DtRadioCryptoSysKY_8,
    DtRadioCryptoSysNarrowSpectrumSecureVoice_NSVE,
```

```
DtRadioCryptoSysWideSpectrumSecureVoice_WSVE,  
DtRadioCryptoSysSINCGARS_ICOM  
};  
  
enum DtEncodingClass  
{  
    DtSignalClassEncodedVoice,  
    DtSignalClassRawBinary,  
    DtSignalClassApplicationSpecific,  
    DtSignalClassPreRecorded  
};  
  
typedef DtEncodingClass DtRadioSignalEncoding;  
const DtEncodingClass DtEncodedVoice = DtEncodingClass(0);  
const DtEncodingClass DtRawBinary = DtEncodingClass(1);  
const DtEncodingClass DtApplicationSpecific = DtEncodingClass(2);  
const DtEncodingClass DtPreRecordedVoice = DtEncodingClass(3);  
  
enum DtRadioReceiverState  
{  
    DtRadioOff,  
    DtRadioOnNotReceiving,  
    DtRadioOn  
};  
  
enum DtTacticalDataLinkType  
{  
    DtTacticalDataLinkOther = 0,  
    DtAbbreviated_Command_And_Control = 15,  
    DtTdlGC3 = 99  
};  
  
enum DtUserProtocol  
{  
    DtUserProtocolOther = 0,  
    DtCCSIL = 1,  
    DtA2ATD_SINCGARS_ERF = 5,  
    DtA2ATD_CAC2 = 6,  
    DtBattle_Command = 20,  
    DtModSAF_Text_Radio = 177,  
    DtCCTT_SINCGARS_ERF_LOCKOUT = 200,  
    DtCCTT_SINCGARS_ERF_HOPSET = 201,  
    DtCCTT_SINCGARS_OTAR = 203,  
    DtCCTT_SINCGARS_DATA = 204,  
    DtModSAF_FWA_Forward_Air_Controller = 546,  
    DtModSAF_Threat_ADA_C3 = 832,  
    DtModSAF_Artillery_Fire_Control = 4570,  
    DtAGTS = 5361,  
    DtUserGC3 = 6000,  
    DtWNCP_data = 6010
```

```
};

enum DtEncodingType
{
    DtEncodingTypeOther = 0,
    DtEncoding_8_bit_mu_law = 1,
    DtCVSD_per_MIL_STD_188_113 = 2,
    DtADPCM_per_CCITT_G721 = 3,
    DtEncoding_16_bit_linear_PCM = 4,
    DtEncoding_8_bit_linear_PCM = 5,
    DtVQ_Vector_Quantization = 6
};

enum DtCreateObjectResult
{
    DtCreateObjectResultOther = 0
};

enum DtRemoveObjectResult
{
    DtRemoveObjectResultOther = 0
};

enum DtAttributeChangeResult
{
    DtAttributeChangeResultOther = 0
};

/* Section 7 */

enum DtDeadReckonTypes
{
    DtDrOther = 0,
    DtDrStatic = 1,
    DtDrDrmFpw = 2,
    DtDrDrmRpw = 3,
    DtDrDrmRvw = 4,
    DtDrDrmFvw = 5,
    DtDrDrmFpb = 6,
    DtDrDrmRpb = 7,
    DtDrDrmRvb = 8,
    DtDrDrmFvb = 9
};

enum DtViewerCommand
{
    DtViewerAbsolute= 0,
    DtViewerTether= 1,
    DtViewerCompass= 2,
    DtViewerMimic= 3,
```

```
DtViewerTurret= 4,
DtViewerOrbit= 5,
DtViewerTrack= 6,
DtViewerTetherTrack= 7,
DtViewerMimicTrack= 8,
DtViewerGroup= 9,
DtViewerModeless= 10
};

enum DtLgrCommand
{
    DtLgrPbFile= 0,
    DtLgrRecFile= 1,
    DtLgrQuit= 2,
    DtLgrStart= 3,
    DtLgrEnd= 4,
    DtLgrPlay= 5,
    DtLgrPause= 6,
    DtLgrStop= 7,
    DtLgrRecord= 8,
    DtLgrSetTime= 9,
    DtLgrSkipTime= 10,
    DtLgrTimeScale= 11,
    DtLgrPlayAction= 12,
    DtLgrRecordAction= 13,
    DtLgrEofAction= 14,
    DtLgrHeartbeat= 15,
    DtLgrTimeout= 16
};

enum DtAppearAttributeBitShift
{
    DtAppPaintSchemeBitShift = 0,
    DtAppMobilityKillBitShift = 1,
    DtAppFirePowerKillBitShift = 2,
    DtAppDamageBitShift = 3,
    DtAppSmokeBitShift = 5,
    DtAppTrailingEffectsBitShift = 7,
    DtAppHatchBitShift = 9,
    DtAppLightsBitShift = 12,
    DtAppFlamingBitShift = 15,
    #if DtDISVERS != 3
        DtAppFrozenBitShift = 21,
        DtAppPowerPlantStatusBitShift = 22,
    #endif
    #if !DtHLA
        DtAppFinalPduBitShift = 23,
    #endif
    DtAppDummyBitShift = 99
};
```



```

// The rest are entity type specific, and thus there are several
// names with the same value. Define them as consts below.

// This macro creates definitions that look like this:
// const DtAppearAttributeBitShift DtAppDensityBitShift =
//     DtAppearAttributeBitShift(16);

#define DtEXTEND_BIT_SHIFT_ENUM(attr, val) \
    const DtAppearAttributeBitShift DtApp##attr##BitShift = \
        DtAppearAttributeBitShift(val)

DtEXTEND_BIT_SHIFT_ENUM(ActiveState, 23);
DtEXTEND_BIT_SHIFT_ENUM(Launcher, 16);
DtEXTEND_BIT_SHIFT_ENUM(CamoType, 17);
DtEXTEND_BIT_SHIFT_ENUM(Concealed, 19);
DtEXTEND_BIT_SHIFT_ENUM(Tent, 24);
DtEXTEND_BIT_SHIFT_ENUM(Ramp, 25);
DtEXTEND_BIT_SHIFT_ENUM(AfterBurner, 16);
DtEXTEND_BIT_SHIFT_ENUM(LaunchFlash, 16);
#if DtDISVERS != 3
DtEXTEND_BIT_SHIFT_ENUM(LifeformState, 16);
DtEXTEND_BIT_SHIFT_ENUM(Weapon1, 24);
DtEXTEND_BIT_SHIFT_ENUM(Weapon2, 26);
#else
DtEXTEND_BIT_SHIFT_ENUM(LifeformState, 16);
DtEXTEND_BIT_SHIFT_ENUM(Weapon1, 18);
DtEXTEND_BIT_SHIFT_ENUM(Weapon2, 19);
#endif
DtEXTEND_BIT_SHIFT_ENUM(Density, 16);

// DMO enums used by 2.1.4 PDUs

// Minefield appearance

DtEXTEND_BIT_SHIFT_ENUM(MinefieldType, 0);
DtEXTEND_BIT_SHIFT_ENUM(MinefieldActiveStatus, 2);
DtEXTEND_BIT_SHIFT_ENUM(MinefieldLane, 3);
DtEXTEND_BIT_SHIFT_ENUM(MinefieldState, 13);

// Point, Linear, and Areal Object appearance

DtEXTEND_BIT_SHIFT_ENUM(ObjectPercentComplete, 0);
DtEXTEND_BIT_SHIFT_ENUM(ObjectDamage, 8);
DtEXTEND_BIT_SHIFT_ENUM(ObjectPredistributed, 10);
DtEXTEND_BIT_SHIFT_ENUM(ObjectState, 11);
DtEXTEND_BIT_SHIFT_ENUM(ObjectSmoking, 12);
DtEXTEND_BIT_SHIFT_ENUM(ObjectFlaming, 13);

// Point object specific appearance

```

```
// This is for the log crib, abatis, vehicle defilade, and infantry
// fighting position breach appearance.
DtEXTEND_BIT_SHIFT_ENUM(ObjectAbatisBreach, 0);

DtEXTEND_BIT_SHIFT_ENUM(ObjectBurstOpacity, 0);
DtEXTEND_BIT_SHIFT_ENUM(ObjectBurstSize, 8);
DtEXTEND_BIT_SHIFT_ENUM(ObjectBurstHeight, 16);
DtEXTEND_BIT_SHIFT_ENUM(ObjectBurstNumOfBursts, 24);
DtEXTEND_BIT_SHIFT_ENUM(ObjectBurstChemicalSmoke, 30);

DtEXTEND_BIT_SHIFT_ENUM(ObjectCraterSize, 0);

DtEXTEND_BIT_SHIFT_ENUM(ObjectRibbonBridgeNumOfSegments, 0);

// Linear object specific appearance

// This is for the tank ditch and concertina wire breach appearance
DtEXTEND_BIT_SHIFT_ENUM(ObjectTankDitchBreach, 0);
DtEXTEND_BIT_SHIFT_ENUM(ObjectTankDitchBreachLength, 16);
DtEXTEND_BIT_SHIFT_ENUM(ObjectTankDitchBreachLocation, 24);

DtEXTEND_BIT_SHIFT_ENUM(ObjectExhaustSmokeOpacity, 0);
DtEXTEND_BIT_SHIFT_ENUM(ObjectExhaustSmokeAttached, 8);
DtEXTEND_BIT_SHIFT_ENUM(ObjectExhaustSmokeChemical, 9);

DtEXTEND_BIT_SHIFT_ENUM(ObjectMinefieldLaneMarkerVisibleSide, 0);

// Areal object specific appearance

DtEXTEND_BIT_SHIFT_ENUM(ObjectMinefieldBreach, 0);
DtEXTEND_BIT_SHIFT_ENUM(ObjectMinefieldMineCount, 16);

// if you put spaces between the "<<" and the surrounding strings,
// enumstr gets confused and thinks the shift is another value.
enum DtAppearAttribute
{
    DtAppPaintScheme = 0x1<<DtAppPaintSchemeBitShift,
    DtAppMobilityKill = 0x1<<DtAppMobilityKillBitShift,
    DtAppFirePowerKill = 0x1<<DtAppFirePowerKillBitShift,
    DtAppDamage = 0x3<<DtAppDamageBitShift,
    DtAppSmoke = 0x3<<DtAppSmokeBitShift,
    DtAppTrailingEffects = 0x3<<DtAppTrailingEffectsBitShift,
    DtAppHatch = 0x7<<DtAppHatchBitShift,
    DtAppLights = 0x7<<DtAppLightsBitShift,
    DtAppFlaming = 0x1<<DtAppFlamingBitShift,
#ifdef DtDISVERS != 3
    DtAppFrozen = 0x1<<DtAppFrozenBitShift,
    DtAppPowerPlantStatus = 0x1<<DtAppPowerPlantStatusBitShift,
#endif
}
```

```

#if !DtHLA
    DtAppFinalPdu = 0x1<<DtAppFinalPduBitShift,
#endif
    DtAppDummy =          99
};

// This macro creates definitions that look like this:
// const DtAppearAttribute DtAppDensity =
//     DtAppearAttribute(16 << DtAppDensityBitShift);

#define DtEXTEND_ATTR_ENUM(attr, val) \
    const DtAppearAttribute attr = \
        DtAppearAttribute(val << attr##BitShift)

DtEXTEND_ATTR_ENUM(DtAppActiveState, 0x1);
DtEXTEND_ATTR_ENUM(DtAppLauncher, 0x1);
DtEXTEND_ATTR_ENUM(DtAppCamoType, 0x3);
DtEXTEND_ATTR_ENUM(DtAppConcealed, 0x1);
DtEXTEND_ATTR_ENUM(DtAppTent, 0x1);
DtEXTEND_ATTR_ENUM(DtAppRamp, 0x1);
DtEXTEND_ATTR_ENUM(DtAppAfterBurner, 0x1);
DtEXTEND_ATTR_ENUM(DtAppLaunchFlash, 0x1);
#if DtDISVERS != 3
DtEXTEND_ATTR_ENUM(DtAppLifeformState, 0xf);
DtEXTEND_ATTR_ENUM(DtAppWeapon1, 0x3);
DtEXTEND_ATTR_ENUM(DtAppWeapon2, 0x3);
#else
DtEXTEND_ATTR_ENUM(DtAppLifeformState, 0x3);
DtEXTEND_ATTR_ENUM(DtAppWeapon1, 0x1);
DtEXTEND_ATTR_ENUM(DtAppWeapon2, 0x1);
#endif
DtEXTEND_ATTR_ENUM(DtAppDensity, 0xf);

// DMO enums used by 2.1.4 PDUs

// Minefield appearance

DtEXTEND_ATTR_ENUM(DtAppMinefieldType, 0x3);
DtEXTEND_ATTR_ENUM(DtAppMinefieldActiveStatus, 0x1);
DtEXTEND_ATTR_ENUM(DtAppMinefieldLane, 0x1);
DtEXTEND_ATTR_ENUM(DtAppMinefieldState, 0x1);

// Point, Linear, and Areal Object appearance

DtEXTEND_ATTR_ENUM(DtAppObjectPercentComplete, 0xff);
DtEXTEND_ATTR_ENUM(DtAppObjectDamage, 0x3);
DtEXTEND_ATTR_ENUM(DtAppObjectPredistributed, 0x1);
DtEXTEND_ATTR_ENUM(DtAppObjectState, 0x1);
DtEXTEND_ATTR_ENUM(DtAppObjectSmoking, 0x1);
DtEXTEND_ATTR_ENUM(DtAppObjectFlaming, 0x1);

```

```
// Point object specific appearance

// This is for the log crib, abatis, vehicle defilade, and infantry
// fighting position breach appearance.
DtEXTEND_ATTR_ENUM(DtAppObjectAbatisBreach, 0x3);

DtEXTEND_ATTR_ENUM(DtAppObjectBurstOpacity, 0xff);
DtEXTEND_ATTR_ENUM(DtAppObjectBurstSize, 0xff);
DtEXTEND_ATTR_ENUM(DtAppObjectBurstHeight, 0xff);
DtEXTEND_ATTR_ENUM(DtAppObjectBurstNumOfBursts, 0x3f);
DtEXTEND_ATTR_ENUM(DtAppObjectBurstChemicalSmoke, 0x3);

DtEXTEND_ATTR_ENUM(DtAppObjectCraterSize, 0xff);

DtEXTEND_ATTR_ENUM(DtAppObjectRibbonBridgeNumOfSegments, 0xff);

// Linear object specific appearance

// This is for the tank ditch and concertina wire breach appearance
DtEXTEND_ATTR_ENUM(DtAppObjectTankDitchBreach, 0x3);
DtEXTEND_ATTR_ENUM(DtAppObjectTankDitchBreachLength, 0xff);
DtEXTEND_ATTR_ENUM(DtAppObjectTankDitchBreachLocation, 0xff);

DtEXTEND_ATTR_ENUM(DtAppObjectExhaustSmokeOpacity, 0xff);
DtEXTEND_ATTR_ENUM(DtAppObjectExhaustSmokeAttached, 0x1);
DtEXTEND_ATTR_ENUM(DtAppObjectExhaustSmokeChemical, 0x3);

DtEXTEND_ATTR_ENUM(DtAppObjectMinefieldLaneMarkerVisibleSide, 0x3);

// Areal object specific appearance

DtEXTEND_ATTR_ENUM(DtAppObjectMinefieldBreach, 0x3);
DtEXTEND_ATTR_ENUM(DtAppObjectMinefieldMineCount, 0xffff);

// Create an enumeration type, but since many of the names will have
// the same value, define them all as consts below.
enum DtAppearAttrVal
{
    DtAppearInvalid = -1 // For DEC alpha
};

#define DtEXTEND_ATTR_VAL_ENUM(name, val, attr) \
    DtEXTEND_ENUM(DtAppearAttrVal, name, \
        val << DtApp##attr##BitShift)

DtEXTEND_ATTR_VAL_ENUM(DtPaintSchemeUniformBitVal, 0, PaintScheme);
DtEXTEND_ATTR_VAL_ENUM(DtPaintSchemeCamouflageBitVal, 1, Paint-
Scheme);
```

```

DtEXTEND_ATTR_VAL_ENUM(DtMobilityKillNoneBitVal, 0, MobilityKill);
DtEXTEND_ATTR_VAL_ENUM(DtMobilityKillDisabledBitVal, 1, MobilityKill);

DtEXTEND_ATTR_VAL_ENUM(DtFirePowerKillNoneBitVal, 0, FirePowerKill);
DtEXTEND_ATTR_VAL_ENUM(DtFirePowerKillDisabledBitVal, 1, FirePowerKill);

DtEXTEND_ATTR_VAL_ENUM(DtDamageNoneBitVal, 0, Damage);
DtEXTEND_ATTR_VAL_ENUM(DtDamageSlightBitVal, 1, Damage);
DtEXTEND_ATTR_VAL_ENUM(DtDamageModerateBitVal, 2, Damage);
DtEXTEND_ATTR_VAL_ENUM(DtDamageDestroyedBitVal, 3, Damage);

DtEXTEND_ATTR_VAL_ENUM(DtSmokeNoneBitVal, 0, Smoke);
DtEXTEND_ATTR_VAL_ENUM(DtSmokeRisingBitVal, 1, Smoke);
DtEXTEND_ATTR_VAL_ENUM(DtSmokeEngineBitVal, 2, Smoke);
DtEXTEND_ATTR_VAL_ENUM(DtSmokeEngineAndPlumeBitVal, 3, Smoke);

DtEXTEND_ATTR_VAL_ENUM(DtTrailingEffectsNoneBitVal, 0, TrailingEffects);
DtEXTEND_ATTR_VAL_ENUM(DtTrailingEffectsSmallBitVal, 1, TrailingEffects);
DtEXTEND_ATTR_VAL_ENUM(DtTrailingEffectsMediumBitVal, 2, TrailingEffects);
DtEXTEND_ATTR_VAL_ENUM(DtTrailingEffectsLargeBitVal, 3, TrailingEffects);

DtEXTEND_ATTR_VAL_ENUM(DtHatchNABitVal, 0, Hatch);
DtEXTEND_ATTR_VAL_ENUM(DtHatchClosedBitVal, 1, Hatch);
DtEXTEND_ATTR_VAL_ENUM(DtHatchPoppedBitVal, 2, Hatch);
DtEXTEND_ATTR_VAL_ENUM(DtHatchPoppedAndPersonBitVal, 3, Hatch);
DtEXTEND_ATTR_VAL_ENUM(DtHatchOpenBitVal, 4, Hatch);
DtEXTEND_ATTR_VAL_ENUM(DtHatchOpenAndPersonBitVal, 5, Hatch);

DtEXTEND_ATTR_VAL_ENUM(DtLightsNoneBitVal, 0, Lights);
DtEXTEND_ATTR_VAL_ENUM(DtLightsRunningBitVal, 1, Lights);
DtEXTEND_ATTR_VAL_ENUM(DtLightsNavigationBitVal, 2, Lights);
DtEXTEND_ATTR_VAL_ENUM(DtLightsFormationBitVal, 3, Lights);

DtEXTEND_ATTR_VAL_ENUM(DtFlamingNoBitVal, 0, Flaming);
DtEXTEND_ATTR_VAL_ENUM(DtFlamingYesBitVal, 1, Flaming);

#if !DthLA
DtEXTEND_ATTR_VAL_ENUM(DtNotFinalPduBitVal, 0, FinalPdu);
DtEXTEND_ATTR_VAL_ENUM(DtFinalPduBitVal, 1, FinalPdu);
#endif

#if DtDISVERS != 3
DtEXTEND_ATTR_VAL_ENUM(DtNotFrozenBitVal, 0, Frozen);

```

```
DtEXTEND_ATTR_VAL_ENUM(DtFrozenBitVal, 1, Frozen);

DtEXTEND_ATTR_VAL_ENUM(DtPowerPlantOffBitVal, 0, PowerPlantStatus);
DtEXTEND_ATTR_VAL_ENUM(DtPowerPlantOnBitVal, 1, PowerPlantStatus);
#endif

DtEXTEND_ATTR_VAL_ENUM(DtActiveBitVal, 0, ActiveState);
DtEXTEND_ATTR_VAL_ENUM(DtDeactivatedBitVal, 1, ActiveState);

DtEXTEND_ATTR_VAL_ENUM(DtLauncherNotRaisedBitVal, 0, Launcher);
DtEXTEND_ATTR_VAL_ENUM(DtLauncherRaisedBitVal, 1, Launcher);

DtEXTEND_ATTR_VAL_ENUM(DtDesertCamouflageBitVal, 0, CamoType);
DtEXTEND_ATTR_VAL_ENUM(DtWinterCamouflageBitVal, 1, CamoType);
DtEXTEND_ATTR_VAL_ENUM(DtForestCamouflageBitVal, 2, CamoType);

DtEXTEND_ATTR_VAL_ENUM(DtNotConcealedBitVal, 0, Concealed);
DtEXTEND_ATTR_VAL_ENUM(DtConcealedBitVal, 1, Concealed);

DtEXTEND_ATTR_VAL_ENUM(DtTentNotExtendedBitVal, 0, Tent);
DtEXTEND_ATTR_VAL_ENUM(DtTentExtendedBitVal, 1, Tent);

DtEXTEND_ATTR_VAL_ENUM(DtRampUpBitVal, 0, Ramp);
DtEXTEND_ATTR_VAL_ENUM(DtRampDownBitVal, 1, Ramp);

DtEXTEND_ATTR_VAL_ENUM(DtAfterBurnerNotOnBitVal, 0, AfterBurner);
DtEXTEND_ATTR_VAL_ENUM(DtAfterBurnerOnBitVal, 1, AfterBurner);

DtEXTEND_ATTR_VAL_ENUM(DtLaunchFlashNotPresentBitVal, 0, LaunchFlash);
DtEXTEND_ATTR_VAL_ENUM(DtLaunchFlashPresentBitVal, 1, LaunchFlash);

DtEXTEND_ATTR_VAL_ENUM(DtLifeformNABitVal, 0, LifeformState);
DtEXTEND_ATTR_VAL_ENUM(DtLifeformStandingBitVal, 1, LifeformState);
DtEXTEND_ATTR_VAL_ENUM(DtLifeformWalkingBitVal, 2, LifeformState);
DtEXTEND_ATTR_VAL_ENUM(DtLifeformRunningBitVal, 3, LifeformState);
DtEXTEND_ATTR_VAL_ENUM(DtLifeformKneelingBitVal, 4, LifeformState);
DtEXTEND_ATTR_VAL_ENUM(DtLifeformProneBitVal, 5, LifeformState);
DtEXTEND_ATTR_VAL_ENUM(DtLifeformCrawlingBitVal, 6, LifeformState);
DtEXTEND_ATTR_VAL_ENUM(DtLifeformSwimmingBitVal, 7, LifeformState);
DtEXTEND_ATTR_VAL_ENUM(DtLifeformParachutingBitVal, 8, LifeformState);
DtEXTEND_ATTR_VAL_ENUM(DtLifeformJumpingBitVal, 9, LifeformState);
DtEXTEND_ATTR_VAL_ENUM(DtLifeformSittingBitVal, 10, LifeformState);
DtEXTEND_ATTR_VAL_ENUM(DtLifeformSquattingBitVal, 11, LifeformState);
DtEXTEND_ATTR_VAL_ENUM(DtLifeformCrouchingBitVal, 12, LifeformState);
DtEXTEND_ATTR_VAL_ENUM(DtLifeformWadingBitVal, 13, LifeformState);
```

```

DtEXTEND_ATTR_VAL_ENUM(DtWeapon1None, 0, Weapon1);
DtEXTEND_ATTR_VAL_ENUM(DtWeapon1Stowed, 1, Weapon1);
DtEXTEND_ATTR_VAL_ENUM(DtWeapon1Deployed, 2, Weapon1);
DtEXTEND_ATTR_VAL_ENUM(DtWeapon1InFirePosition, 3, Weapon1);

DtEXTEND_ATTR_VAL_ENUM(DtWeapon2None, 0, Weapon2);
DtEXTEND_ATTR_VAL_ENUM(DtWeapon2Stowed, 1, Weapon2);
DtEXTEND_ATTR_VAL_ENUM(DtWeapon2Deployed, 2, Weapon2);
DtEXTEND_ATTR_VAL_ENUM(DtWeapon2InFirePosition, 3, Weapon2);

DtEXTEND_ATTR_VAL_ENUM(DtDensityClearBitVal, 0, Density);
DtEXTEND_ATTR_VAL_ENUM(DtDensityHazyBitVal, 1, Density);
DtEXTEND_ATTR_VAL_ENUM(DtDensityDenseBitVal, 2, Density);
DtEXTEND_ATTR_VAL_ENUM(DtDensityVeryDenseBitVal, 3, Density);
DtEXTEND_ATTR_VAL_ENUM(DtDensityOpaqueBitVal, 4, Density);

// DMO used by 2.1.4 PDUs

// Minefield appearance

DtEXTEND_ATTR_VAL_ENUM(DtMixedMinefield, 0, MinefieldType);
DtEXTEND_ATTR_VAL_ENUM(DtPureAntiPersonnelMinefield, 1, Minefield-
Type);
DtEXTEND_ATTR_VAL_ENUM(DtPureAntiTankMinefield, 2, MinefieldType);

DtEXTEND_ATTR_VAL_ENUM(DtActiveMinefield, 0, MinefieldAc-
tiveStatus);
DtEXTEND_ATTR_VAL_ENUM(DtInactiveMinefield, 1, MinefieldAc-
tiveStatus);

DtEXTEND_ATTR_VAL_ENUM(DtMinefieldHasActiveLane, 0, MinefieldLane);
DtEXTEND_ATTR_VAL_ENUM(DtMinefieldHasInactiveLane, 1, Minefield-
Lane);

DtEXTEND_ATTR_VAL_ENUM(DtActiveMinefieldState, 0, MinefieldState);
DtEXTEND_ATTR_VAL_ENUM(DtDeactivatedMinefieldState, 1, Minefield-
State);

// Point, Linear, and Areal Object appearance

DtEXTEND_ATTR_VAL_ENUM(DtObjectNoDamage, 0, ObjectDamage);
DtEXTEND_ATTR_VAL_ENUM(DtObjectDamaged, 1, ObjectDamage);
DtEXTEND_ATTR_VAL_ENUM(DtObjectDestroyed, 2, ObjectDamage);

DtEXTEND_ATTR_VAL_ENUM(DtObjectCreatedDuringExercise, 0, ObjectPre-
distributed);
DtEXTEND_ATTR_VAL_ENUM(DtObjectPredistributed, 1, ObjectPredistrib-
uted);

DtEXTEND_ATTR_VAL_ENUM(DtObjectActive, 0, ObjectState);

```

```
DtEXTEND_ATTR_VAL_ENUM(DtObjectDeactivated, 1, ObjectState);

DtEXTEND_ATTR_VAL_ENUM(DtObjectNotSmoking, 0, ObjectSmoking);
DtEXTEND_ATTR_VAL_ENUM(DtObjectSmoking, 1, ObjectSmoking);

DtEXTEND_ATTR_VAL_ENUM(DtObjectNotFlaming, 0, ObjectFlaming);
DtEXTEND_ATTR_VAL_ENUM(DtObjectFlaming, 1, ObjectFlaming);

// Point object specific appearance

// This is for the log crib, abatis, vehicle defilade, and infantry
// fighting position breach appearance.
DtEXTEND_ATTR_VAL_ENUM(DtObjectAbatisNoBreaching, 0, ObjectAbatis-
Breach);
DtEXTEND_ATTR_VAL_ENUM(DtObjectAbatisBreached, 1, ObjectAbatis-
Breach);
DtEXTEND_ATTR_VAL_ENUM(DtObjectAbatisCleared, 2, ObjectAbatis-
Breach);

DtEXTEND_ATTR_VAL_ENUM(DtObjectBurstOtherSmoke, 0,
    ObjectBurstChemicalSmoke);
DtEXTEND_ATTR_VAL_ENUM(DtObjectBurstHydrochloricSmoke, 1,
    ObjectBurstChemicalSmoke);
DtEXTEND_ATTR_VAL_ENUM(DtObjectBurstWhitePhosphorousSmoke, 2,
    ObjectBurstChemicalSmoke);
DtEXTEND_ATTR_VAL_ENUM(DtObjectBurstRedPhosphorousSmoke, 3,
    ObjectBurstChemicalSmoke);

// Linear object specific appearance

DtEXTEND_ATTR_VAL_ENUM(DtObjectTankDitchBreachNoBreaching, 0,
    ObjectTankDitchBreach);
DtEXTEND_ATTR_VAL_ENUM(DtObjectTankDitchBreachSlightBreaching, 1,
    ObjectTankDitchBreach);
DtEXTEND_ATTR_VAL_ENUM(DtObjectTankDitchBreachModerateBreaching, 2,
    ObjectTankDitchBreach);
DtEXTEND_ATTR_VAL_ENUM(DtObjectTankDitchBreachCleared, 3,
    ObjectTankDitchBreach);

DtEXTEND_ATTR_VAL_ENUM(DtObjectExhaustSmokeNotAttached, 0,
    ObjectExhaustSmokeAttached);
DtEXTEND_ATTR_VAL_ENUM(DtObjectExhaustSmokeIsAttached, 1,
    ObjectExhaustSmokeAttached);

DtEXTEND_ATTR_VAL_ENUM(DtObjectExhaustOtherSmoke, 0,
    ObjectExhaustSmokeChemical);
DtEXTEND_ATTR_VAL_ENUM(DtObjectExhaustHydrochloricSmoke, 1,
    ObjectExhaustSmokeChemical);
DtEXTEND_ATTR_VAL_ENUM(DtObjectExhaustWhitePhosphorousSmoke, 2,
    ObjectExhaustSmokeChemical);
```



```

DtEXTEND_ATTR_VAL_ENUM(DtObjectExhaustRedPhosphorousSmoke, 3,
    ObjectExhaustSmokeChemical);

DtEXTEND_ATTR_VAL_ENUM(DtObjectMinefieldLeftSideVisible, 0,
    ObjectMinefieldLaneMarkerVisibleSide);
DtEXTEND_ATTR_VAL_ENUM(DtObjectMinefieldRightSideVisible, 1,
    ObjectMinefieldLaneMarkerVisibleSide);
DtEXTEND_ATTR_VAL_ENUM(DtObjectMinefieldBothSidesVisible, 2,
    ObjectMinefieldLaneMarkerVisibleSide);

DtEXTEND_ATTR_VAL_ENUM(DtObjectMinefieldNoBreaching, 0,
    ObjectMinefieldBreach);
DtEXTEND_ATTR_VAL_ENUM(DtObjectMinefieldBreached, 1,
    ObjectMinefieldBreach);
DtEXTEND_ATTR_VAL_ENUM(DtObjectMinefieldCleared, 2,
    ObjectMinefieldBreach);

// This is for the tank ditch and concertina wire breach appearance
// Areal object specific appearance

enum DtDamageState
{
    DtDamageNone,
    DtDamageSlight,
    DtDamageModerate,
    DtDamageDestroyed
};

enum DtTrailState
{
    DtTrailingEffectsNone,
    DtTrailingEffectsSmall,
    DtTrailingEffectsMedium,
    DtTrailingEffectsLarge
};

enum DtHatchState
{
    DtHatchNA,
    DtHatchClosed,
    DtHatchPopped,
    DtHatchPoppedAndPerson,
    DtHatchOpen,
    DtHatchOpenAndPerson
};

enum DtLightState
{
    DtLightsNone,
    DtLightsRunning,

```

```
    DtLightsNavigation,  
    DtLightsFormation  
};  
  
enum DtLifeformState  
{  
    DtLifeformNA,  
    DtLifeformStanding,  
    DtLifeformWalking,  
    DtLifeformRunning,  
    DtLifeformKneeling,  
    DtLifeformProne,  
    DtLifeformCrawling,  
    DtLifeformSwimming,  
    DtLifeformParachuting,  
    DtLifeformJumping,  
    DtLifeformSitting,  
    DtLifeformSquatting,  
    DtLifeformCrouching,  
    DtLifeformWading  
};  
  
enum DtCamouflageType  
{  
    DtDesertCamouflage,  
    DtWinterCamouflage,  
    DtForestCamouflage  
};  
  
enum DtWeaponState  
{  
    DtNoWeapon,  
    DtWeaponStowed,  
    DtWeaponDeployed,  
    DtWeaponInFirePosition  
};  
  
enum DtCapability  
{  
    DtAmmunitionSupplyCap,  
    DtFuelSupplyCap,  
    DtRecoveryCap,  
    DtRepairCap  
};  
  
enum DtDensityType  
{  
    DtDensityClear,  
    DtDensityHazy,  
    DtDensityDense,
```

```
DtDensityVeryDense,  
DtDensityOpaque  
};  
  
// Articulated parts  
enum DtArtParamType  
{  
    DtApPosition= 1,  
    DtApPositionRate= 2,  
    DtApExtension= 3,  
    DtApExtensionRate= 4,  
    DtApX= 5,  
    DtApXRate= 6,  
    DtApY= 7,  
    DtApYRate= 8,  
    DtApZ= 9,  
    DtApZRate= 10,  
    DtApAzimuth= 11,  
    DtApAzimuthRate= 12,  
    DtApElevation= 13,  
    DtApElevationRate= 14,  
    DtApRotation= 15,  
    DtApRotationRate= 16  
};  
  
enum DtArtPartType  
{  
    DtEntityBase= -1,  
    DtRudder = 1024,  
    DtLeftFlap = 1056,  
    DtRightFlap= 1088,  
    DtLeftAileron= 1120,  
    DtRightAileron= 1152,  
  
    DtPeriscope= 2048,  
    DtGenericAntenna= 2080,  
    DtSnorkel= 2112,  
  
    DtLandingGear= 3072,  
    DtTailHook= 3104,  
    DtSpeedBrake= 3136,  
    DtLeftWeaponBayDoor = 3168,  
    DtRightWeaponBayDoor= 3200,  
    DtTankAPCHatch= 3232,  
    DtWingSweep= 3164,  
  
    DtPrimaryTurret1= 4096,  
    DtPrimaryTurret2= 4128,  
    DtPrimaryTurret3= 4160,  
    DtPrimaryTurret4= 4192,
```

DtPrimaryTurret5= 4224,
DtPrimaryTurret6= 4256,
DtPrimaryTurret7= 4288,
DtPrimaryTurret8= 4320,
DtPrimaryTurret9= 4352,
DtPrimaryTurret10= 4384,

DtPrimaryGun1= 4416,
DtPrimaryGun2= 4448,
DtPrimaryGun3= 4480,
DtPrimaryGun4= 4512,
DtPrimaryGun5= 4544,
DtPrimaryGun6= 4576,
DtPrimaryGun7= 4608,
DtPrimaryGun8= 4640,
DtPrimaryGun9= 4672,
DtPrimaryGun10= 4704,

DtPrimaryLauncher1 = 4736,
DtPrimaryLauncher2 = 4768,
DtPrimaryLauncher3 = 4800,
DtPrimaryLauncher4 = 4832,
DtPrimaryLauncher5 = 4864,
DtPrimaryLauncher6 = 4896,
DtPrimaryLauncher7 = 4928,
DtPrimaryLauncher8 = 4960,
DtPrimaryLauncher9 = 4992,
DtPrimaryLauncher10 = 5024,

DtPrimaryDefenseSys1 = 5056,
DtPrimaryDefenseSys2 = 5088,
DtPrimaryDefenseSys3 = 5120,
DtPrimaryDefenseSys4 = 5152,
DtPrimaryDefenseSys5 = 5184,
DtPrimaryDefenseSys6 = 5216,
DtPrimaryDefenseSys7 = 5248,
DtPrimaryDefenseSys8 = 5280,
DtPrimaryDefenseSys9 = 5312,
DtPrimaryDefenseSys10= 5344,

DtPrimaryRadar1= 5376,
DtPrimaryRadar2= 5408,
DtPrimaryRadar3 = 5440,
DtPrimaryRadar4 = 5472,
DtPrimaryRadar5 = 5504,
DtPrimaryRadar6 = 5536,
DtPrimaryRadar7 = 5568,
DtPrimaryRadar8 = 5600,
DtPrimaryRadar9 = 5632,
DtPrimaryRadar10= 5664,

DtSecondaryTurret1= 5696,
DtSecondaryTurret2= 5728,
DtSecondaryTurret3= 5760,
DtSecondaryTurret4= 5792,
DtSecondaryTurret5= 5824,
DtSecondaryTurret6= 5856,
DtSecondaryTurret7= 5888,
DtSecondaryTurret8= 5920,
DtSecondaryTurret9= 5952,
DtSecondaryTurret10= 5984,

DtSecondaryGun1= 6016,
DtSecondaryGun2= 6048,
DtSecondaryGun3= 6080,
DtSecondaryGun4= 6112,
DtSecondaryGun5= 6144,
DtSecondaryGun6= 6176,
DtSecondaryGun7= 6208,
DtSecondaryGun8= 6240,
DtSecondaryGun9= 6272,
DtSecondaryGun10= 6304,

DtSecondaryLauncher1 = 6336,
DtSecondaryLauncher2 = 6368,
DtSecondaryLauncher3 = 6400,
DtSecondaryLauncher4 = 6432,
DtSecondaryLauncher5 = 6464,
DtSecondaryLauncher6 = 6496,
DtSecondaryLauncher7 = 6528,
DtSecondaryLauncher8 = 6560,
DtSecondaryLauncher9 = 6592,
DtSecondaryLauncher10 = 6624,

DtSecondaryDefenseSys1 = 6656,
DtSecondaryDefenseSys2 = 6688,
DtSecondaryDefenseSys3 = 6720,
DtSecondaryDefenseSys4 = 6752,
DtSecondaryDefenseSys5 = 6784,
DtSecondaryDefenseSys6 = 6816,
DtSecondaryDefenseSys7 = 6848,
DtSecondaryDefenseSys8 = 6880,
DtSecondaryDefenseSys9 = 6912,
DtSecondaryDefenseSys10= 6944,

DtSecondaryRadar1 = 6976,
DtSecondaryRadar2 = 7008,
DtSecondaryRadar3 = 7040,
DtSecondaryRadar4 = 7072,
DtSecondaryRadar5 = 7104,

```
    DtSecondaryRadar6      = 7136,  
    DtSecondaryRadar7      = 7168,  
    DtSecondaryRadar8      = 7200,  
    DtSecondaryRadar9      = 7232,  
    DtSecondaryRadar10     = 7264  
};  
  
// DMO enums used by 2.1.4 PDUs  
  
// Intercom Control PDU enums  
  
enum DtIntercomCtlType  
{  
    DtIntercomCtlStatus      = 1,  
    DtIntercomReqAck         = 2,  
    DtIntercomReqNoAck       = 3,  
    DtIntercomAck            = 4,  
    DtIntercomNack           = 5  
};  
  
enum DtIntercomCommClass  
{  
    DtIntercomSimCommChnl    = 0,  
    DtIntercomSimCommSupChnl = 1  
};  
  
enum DtIntercomCommType  
{  
    DtIntercomFDX            = 1,  
    DtIntercomHDX_RecOnly    = 2,  
    DtIntercomHDX_TransOnly  = 3,  
    DtIntercomHDX            = 4  
};  
  
enum DtIntercomCmdType  
{  
    DtIntercomNoCmd          = 0,  
    DtIntercomCmdStatus      = 1,  
    DtIntercomConnect        = 2,  
    DtIntercomDisconnect     = 3,  
    DtIntercomReset          = 4,  
    DtIntercomOn              = 5,  
    DtIntercomOff            = 6  
};  
  
enum DtIntercomTransLineState  
{  
    DtIntercomNotApplicable  = 0,  
    DtIntercomNotTransmitting = 1,  
    DtIntercomTransmitting   = 2  
};
```

```

};

enum DtIntercomDestLineStateCmd
{
    DtIntercomLS_NoCmd          = 0,
    DtIntercomSetLS_Trans       = 1,
    DtIntercomSetLS_NotTrans    = 2,
    DtIntercomRetLocLS_Ctl      = 3
};

enum DtIntercomCtlRecType
{
    DtIntercomEntDestRec        = 1,
    DtIntercomGrpDestRec        = 2,
    DtIntercomGrpAsgmtRec       = 3
};

// Environmental Process PDU enums

enum DtEnvironmentalModelType
{
    DtEnvModelInvalid = -1 // For DEC alpha
};

enum DtEnvironmentalStatusBit
{
    DtEnvironmentalStatusLast = 0,
    DtEnvironmentalStatusOn   = 1
};

// Gridded data PDU enums

enum DtGridDataRecType
{
    DtGridDataRecType0        = 0,
    DtGridDataRecType1        = 1,
    DtGridDataRecType2        = 2
};

enum DtGridAxisType
{
    DtGridAxisFixed           = 0,
    DtGridAxisVariable        = 1
};

// IFF/ATC/NAVAIDS PDU enums

enum DtIffAtcNav aidsSysName
{
    DtIffAtcNav aidsOtherSysName = 0,

```

```
DtIffAtcNav aidsMarkX           = 1,
DtIffAtcNav aidsMarkXII        = 2,
DtIffAtcNav aidsATCRBS         = 3,
DtIffAtcNav aidsSoviet         = 4,
DtIffAtcNav aidsModeS          = 5,
DtIffAtcNav aidsMarkX_XII_ATCRBS = 6,
DtIffAtcNav aidsMarkX_XII_ATCRBS_ModeS = 7
};

enum DtIffAtcNav aidsSysType
{
    DtIffAtcNav aidsOtherSysType      = 0,
    DtIffAtcNav aidsTransponder       = 1,
    DtIffAtcNav aidsInterrogator      = 2,
    DtIffAtcNav aidsSovietTransponder = 3,
    DtIffAtcNav aidsSovietInterrogator = 4
};

enum DtIffAtcNav aidsStatusBit
{
    DtIffAtcNav aidsSystemOnOff       = 0,
    DtIffAtcNav aidsParameter1        = 1,
    DtIffAtcNav aidsParameter2        = 2,
    DtIffAtcNav aidsParameter3        = 3,
    DtIffAtcNav aidsParameter4        = 4,
    DtIffAtcNav aidsParameter5        = 5,
    DtIffAtcNav aidsParameter6        = 6,
    DtIffAtcNav aidsOperationalStatus = 7
};

enum DtIffAtcNav aidsSysMode
{
    DtIffAtcNav aidsModeInvalid = -1 // For DEC alpha
};

enum DtIffAtcNav aidsChangeOptionsBit
{
    DtIffAtcNav aidsChangeIndicator = 0,
    DtIffAtcNav aidsAlternateMode4  = 1,
    DtIffAtcNav aidsAlternateMode5C = 2
};

enum DtIffAtcNav aidsParameterBit
{
    DtIffAtcNav aidsParameterTCAS      = 12,
    DtIffAtcNav aidsParameterStatus    = 13,
    DtIffAtcNav aidsParameterDamage    = 14,
    DtIffAtcNav aidsParameterMalfunction = 15
};
```



```

enum DtIffAtcNavaidAltParm4
{
    DtIffAtcNavaidAltParm4Other          = 0,
    DtIffAtcNavaidAltParm4Valid          = 1,
    DtIffAtcNavaidAltParm4Invalid        = 2,
    DtIffAtcNavaidAltParm4NoResponse     = 3
};

enum DtIffAtcNavaidModifier
{
    DtIffAtcNavaidModifierInvalid = -1 // For DEC alpha
};

enum DtIffAtcNavaidLayerSpecificInfo
{
    DtIffAtcNavaidLayerInfoInvalid = -1 // For DEC alpha
};

enum DtIffAtcNavaidParm1
{
    DtIffAtcNavaidParm1Invalid = -1 // For DEC alpha
};

enum DtIffAtcNavaidParm2
{
    DtIffAtcNavaidParm2Invalid = -1 // For DEC alpha
};

enum DtIffAtcNavaidApplicableModes
{
    DtIffAtcNavaidAppModeInvalid = -1 // For DEC alpha
};

// Underwater accoustic PDU enums

enum DtUnderwaterAcousticUpdate
{
    DtUnderwaterAcousticStateUpdate      = 0,
    DtUnderwaterAcousticChangedDataUpdate = 1
};

enum DtUnderwaterAcousticSystemName
{
    DtUnderwaterAcousticOtherSystem      = 0,
    DtUnderwaterAcousticAN_BQQ_5         = 1,
    DtUnderwaterAcousticAN_SSQ_62        = 2,
    DtUnderwaterAcousticAN_SQS_23         = 3,
    DtUnderwaterAcousticAN_SQS_26         = 4,
    DtUnderwaterAcousticAN_SQS_53         = 5,
    DtUnderwaterAcousticALFS              = 6,
};

```

```
DtUnderwaterAcousticLFA                = 7,
DtUnderwaterAcousticAN_AQS_901         = 8,
DtUnderwaterAcousticAN_AQS_902         = 9
};

enum DtUnderwaterAcousticFunction
{
    DtUnderwaterAcousticOtherFunction    = 0,
    DtUnderwaterAcousticPlatformSearch_detect_track = 1,
    DtUnderwaterAcousticNavigation        = 2,
    DtUnderwaterAcousticMineHunting       = 3,
    DtUnderwaterAcousticWeaponSearch_detect_track_detect = 4
};

enum DtUnderwaterAcousticActiveEmissionParmIndex
{
    DtUnderwaterAcousticActiveOtherIndex = 0
};

enum DtUnderwaterAcousticScanPattern
{
    DtUnderwaterAcousticScanPatternNotUsed = 0,
    DtUnderwaterAcousticConical            = 1,
    DtUnderwaterAcousticHelical            = 2,
    DtUnderwaterAcousticRaster             = 3,
    DtUnderwaterAcousticSectorSearch       = 4,
    DtUnderwaterAcousticContinuousSearch  = 5
};

enum DtUnderwaterAcousticPassiveParmIndex
{
    DtUnderwaterAcousticOtherPassiveIndex = 0
};

enum DtUnderwaterAcousticPropulsionPlantConfig
{
    DtUnderwaterAcousticOtherConfig        = 0,
    DtUnderwaterAcousticDiesel_electric    = 1,
    DtUnderwaterAcousticDiesel             = 2,
    DtUnderwaterAcousticBattery            = 3,
    DtUnderwaterAcousticNuclear            = 4,
    DtUnderwaterAcousticTurbineReduction    = 5,
    DtUnderwaterAcousticSteam              = 6,
    DtUnderwaterAcousticGasTurbine         = 7
};

enum DtUnderwaterAcousticHullMountedMaskerStatus
{
    DtUnderwaterAcousticHullMountedMaskerOff = 0,
    DtUnderwaterAcousticHullMountedMaskerOn  = 1
}
```

```

};

enum DtUnderwaterAcousticAdditionalPassiveActivityParmIndex
{
    DtUnderwaterAcousticOtherAdditionalIndex    = 0,
    DtUnderwaterAcousticGrahamsMKV_CoffeeMaker = 1
};

// Transfer Control Request PDU enums

enum DtTransferType
{
    DtOtherTransfer            = 0,
    DtCtlReqTransferEntity    = 1,
    DtAppReqTransferEntity    = 2,
    DtReqExchangeEntity       = 3,
    DtCtlReqTransferEnvProc   = 4,
    DtAppReqTransferEnvProc   = 5,
    DtReqExchangeEnvProc      = 6
};

// Reliable service PDU enums

enum DtRequiredReliabilityService
{
    DtAcknowledged    = 0,
    DtUnacknowledged = 1
};

// IsPartOf PDU enums

enum DtNature
{
    DtOtherNature                = 0,
    DtHostFireableMunition       = 1,
    DtMunitionCarriedAsCargo     = 2,
    DtFuelCarriedAsCargo         = 3,
    DtGunmountAttachedToHost     = 4,
    DtComputerGeneratedForcesCarriedAsCargo = 5,
    DtVehicleCarriedAsCargo      = 6,
    DtEmitterMountedOnHost       = 7,
    DtMobileCommandAndControlEntityCarriedAboardHost = 8,
    DtEntityStationedAt          = 9,
    DtTeamMemberInformationWith  = 10
};

enum DtPosition
{
    DtOtherPosition = 0,
    DtOnTopOf       = 1,

```

```
    DtInsideOf          = 2
};

enum DtStationName
{
    DtOtherStationName          = 0,
    DtAircraftWingstation       = 1,
    DtShipsForwardGunmountStarboard = 2,
    DtShipsForwardGunmountPort   = 3,
    DtShipsForwardGunmountCenterline = 4,
    DtShipsAftGunmountStarboard  = 5,
    DtShipsAftGunmountPort       = 6,
    DtShipsAftGunmountCenterline = 7,
    DtForwardTorpedoTube         = 8,
    DtAftTorpedoTube             = 9,
    DtBombBay                    = 10,
    DtCargoBay                   = 11,
    DtTruckBed                   = 12,
    DtTrailerBed                 = 13,
    DtWellDeck                   = 14,
    DtOnStation_RNG_BRG         = 15,
    DtOnStation_XYZ             = 16
};

// Environmental process PDU enums

enum DtObjectKind
{
    DtOtherObjectKind          = 0,
    DtObstacle                 = 1,
    DtPreparedPosition         = 2,
    DtObjectCulturalFeature    = 3,
    DtPassageway               = 4,
    DtTacticalSmoke            = 5,
    DtObstacleMarker           = 6,
    DtObstacleBreach           = 7
};

enum DtModificationField
{
    DtModLocation              = 0,
    DtModOrientation           = 1
};

// Minefield PDU enums

enum DtMinefieldDataFilterField
{
    DtMineGroundBurialDepthOffset = 0,
    DtMineWaterBurialDepthOffset  = 1,
```

```

    DtMineSnowBurialDepthOffset      = 2,
    DtMineOrientation                 = 3,
    DtMineThermalContrast              = 4,
    DtMineReflectance                 = 5,
    DtMineEmplacementAge              = 6,
    DtMineTripDetonationWireLocation = 7,
    DtMineFusing                      = 8,
    DtMineScalarDetectionCoefficient = 9,
    DtMinePaintScheme                 = 10
};

enum DtMinefieldFusing
{
    DtNoFuse                        = 0,
    DtOtherFuse                    = 1,
    DtPressureFuse                  = 2,
    DtMagneticFuse                 = 3,
    DtTiltRodFuse                  = 4,
    DtCommandFuse                  = 5,
    DtTripWireFuse                 = 6
};

enum DtMinefieldAntiHandlingDevice
{
    DtNoAntiHandlingDevice         = 0,
    DtAntiHandlingDevice           = 1
};

enum DtSensorTypeCategory
{
    DtOtherSensorTypeCategory = 0,
    DtOpticalSensorType       = 1,
    DtFLIR_SensorType         = 2,
    DtRADAR_SensorType        = 3,
    DtMagneticSensorType      = 4,
    DtLaserSensorType         = 5,
    DtSONAR_SensorType        = 6,
    DtPhysicalSensorType      = 7,
    DtMultispectralSensorType = 8
};

enum DtOpticalSensorTypeSubcategory
{
    DtUnaidedEyeActivelySearching      = 0,
    DtUnaidedEyeNotActivelySearching   = 1,
    DtBinoculars                       = 2,
    DtImageIntensifier                 = 3,
    DtHMMWV_OccActivelySearching       = 4,
    DtHMMWV_OccNotActivelySearching    = 5,
    DtTruckOccActivelySearching        = 6,

```

```
DtTruckOccNotActivelySearching          = 7,  
DtTrackedVehicleOccClosedHatchActSearching = 8,  
DtTrackedVehicleOccClosedHatchNotActSearching = 9,  
DtTrackedVehicleOccOpenHatchActSearching = 10,  
DtTrackedVehicleOccOpenHatchNotActSearching = 11  
};
```

```
enum DtFLIR_SensorTypeSubcategory  
{  
    DtGeneric3_5    = 0,  
    DtGeneric8_12   = 1,  
    DtASTAMIDS_I    = 2,  
    DtASTAMIDS_II   = 3,  
    DtGSTAMIDS3_5   = 4,  
    DtGSTAMIDS8_12  = 5,  
    DtHSTAMIDS3_5   = 6,  
    DtHSTAMIDS8_12  = 7,  
    DtCOBRA3_5      = 8,  
    DtCOBRA8_12     = 9  
};
```

```
enum DtRADAR_SensorTypeSubcategory  
{  
    DtRADAR_Generic = 0,  
    DtGenericGPR    = 1,  
    DtGSTAMIDS_I    = 2,  
    DtGSTAMIDS_II   = 3,  
    DtHSTAMIDS_I    = 4,  
    DtHSTAMIDS_II   = 5  
};
```

```
enum DtMagneticSensorTypeSubcategory  
{  
    DtMagneticGeneric = 0,  
    DtAN_PSS_11       = 1,  
    DtAN_PSS_12       = 2,  
    DtGSTAMIDS        = 3  
};
```

```
enum DtLaserSensorTypeSubcategory  
{  
    DtLaserGeneric = 0,  
    DtASTAMIDS     = 1  
};
```

```
enum DtSONARSensorTypeSubcategory  
{  
    DtSONAR_Generic = 0  
};
```

```

enum DtPhysicalSensorTypeSubcategory
{
    DtGenericProbe          = 0,
    DtProbeMetalContent     = 1,
    DtProbeNoMetalContent   = 2
};

enum DtMultispectralSensorTypeSubcategory
{
    DtMultispectralGeneric = 0
};

enum DtMinefieldAlgae
{
    DtNoAlgae                = 0,
    DtLightAlgae             = 1,
    DtModerateAlgae          = 2,
    DtHeavyAlgae             = 3
};

enum DtMinefieldPaintScheme
{
    DtOtherPaintScheme       = 0,
    DtStandardPaintScheme    = 1,
    DtCamouflageDesertPaintScheme = 2,
    DtCamouflageJunglePaintScheme = 3,
    DtCamouflageSnowPaintScheme = 4,
    DtCamouflageGravelPaintScheme = 5,
    DtCamouflagePavementPaintScheme = 6,
    DtCamouflageSandPaintScheme = 7,
    DtNaturalWoodPaintScheme = 8,
    DtClearPaintScheme       = 9,
    DtRedPaintScheme         = 10,
    DtBluePaintScheme        = 11,
    DtGreenPaintScheme       = 12,
    DtOlivePaintScheme       = 13,
    DtWhitePaintScheme       = 14,
    DtTanPaintScheme         = 15,
    DtBlackPaintScheme       = 16,
    DtYellowPaintScheme      = 17,
    DtBrownPaintScheme       = 18
};

enum DtMinefieldProtocol
{
    DtHeartbeatMode = 0,
    DtQRP_Mode      = 1
};

// Live entity protocol PDU enums

```

```
enum DtSubProtocol
{
    DtSubProtocolInvalid = -1 // For DEC alpha
};
```

```
enum DtCharacterSet
{
    DtUnusedCharacterSet      = 0,
    DtASCII_CharacterSet      = 1
};
```

```
enum DtLE_TSPI_FieldFlag
{
    DtLE_TSPI_Reserved        = 7,
    DtLE_TSPI_SystemSpecific  = 6,
    DtLE_TSPI_MeasuredSpeed   = 5,
    DtLE_TSPI_DeadReckoning   = 4,
    DtLE_TSPI_OrientationError = 3,
    DtLE_TSPI_PositionError   = 2,
    DtLE_TSPI_EntityOrientation = 1,
    DtLE_TSPI_EntityLinearVelocity = 0
};
```

```
enum DtLE_AppearanceField1Flag
{
    DtLE_AppearanceFlagField2 = 7,
    DtLE_AppearanceIR          = 6,
    DtLE_AppearanceVisual      = 5,
    DtLE_AppearanceCapabilities = 4,
    DtLE_AppearanceMarking     = 3,
    DtLE_AppearanceAlternativeType = 2,
    DtLE_AppearanceType        = 1,
    DtLE_AppearanceForceID     = 0
};
```

```
enum DtLE_AppearanceField2Flag
{
    DtLE_AppearanceAudio      = 1,
    DtLE_AppearanceEM         = 0
};
```

```
enum DtLE_FireFieldFlag
{
    DtLE_FireLocation          = 6,
    DtLE_FireBurstQuantityRate = 5,
    DtLE_FireBurstWarheadFuse  = 4,
    DtLE_FireEventSiteApp      = 3,
    DtLE_FireMunitionID        = 2,
    DtLE_FireMunitionSiteApp   = 1,
};
```



```

    DtLE_FireTargetID          = 0
};

enum DtLE_DetonFieldFlag
{
    DtLE_DetonField2          = 7,
    DtLE_DetonLocationEntityCoords = 6,
    DtLE_DetonBurstQuantityRate = 5,
    DtLE_DetonBurstWarheadFuse  = 4,
    DtLE_DetonEventSiteApp      = 3,
    DtLE_DetonMunitionID        = 2,
    DtLE_DetonMunitionSiteApp    = 1,
    DtLE_DetonTargetID          = 0
};

enum DtLE_DetonField2Flag
{
    DtLE_DetonEventNumber      = 1,
    DtLE_DetonMunitionOrientation = 0
};

// IsGroupOf PDU enums

enum DtGroupedEntityCategory
{
    DtUndefinedGroupEntityCategory = 0,
    DtBasicGroundCombatVehicle     = 1,
    DtEnhancedGroundCombatVehicle  = 2,
    DtBasicGroundCombatSoldier     = 3,
    DtEnhancedGroundCombatSoldier  = 4,
    DtBasicRotorWingAircraft        = 5,
    DtEnhancedRotorWingAircraft     = 6,
    DtBasicFixedWingAircraft        = 7,
    DtEnhancedFixedWingAircraft     = 8,
    DtGroundLogisticsVehicle        = 9
};

enum DtRestStatus
{
    DtSoldierNotRested           = 0,
    DtSoldier1HourSleep          = 1,
    DtSoldier2HourSleep          = 2,
    DtSoldier3HourSleep          = 3,
    DtSoldier4HourSleep          = 4,
    DtSoldier5HourSleep          = 5,
    DtSoldier6HourSleep          = 6,
    DtSoldier7HourSleep          = 7,
    DtSoldierFullyRested         = 8
};

```

// Aggregate State PDU enums

```
enum DtAggregateState
{
    DtOtherAggregateState      = 0,
    DtAggregated               = 1,
    DtDisaggregated            = 2,
    DtFullyDisaggregated        = 3,
    DtPseudoDisaggregated      = 4,
    DtPartiallyDisaggregated    = 5
};
```

```
enum DtAggregateFormation
{
    DtOtherAggregateFormation  = 0,
    DtAssemblyFormation        = 1,
    DtVeeFormation             = 2,
    DtWedgeFormation           = 3,
    DtLineFormation            = 4,
    DtColumnFormation          = 5
};
```

disPduFactory.h

```

#if !DtHLA

#include "pduFactory.h"

// class DtDisPduFactory:
//
// Instances of DtDisPduFactory are used to create DIS PDUs.
// This class knows how to create the various DIS pdus.

class DT_DLL_VL DtDisPduFactory : public DtPduFactory
{
public:

    // default constructor
    // Adds creators for the various DIS pdus.
    DtDisPduFactory();

    // destructor
    virtual ~DtDisPduFactory();

    // copy constructor
    DtDisPduFactory(const DtDisPduFactory& orig);

    // assignment operator
    DtDisPduFactory& operator=(const DtDisPduFactory& orig);

public:

#if DtDISVERS == 3
    // Adds creators for all DIS 2.0.3 PDUs to fact.
    // Includes ViewControl and LgrControl
    static void addDis203Creators(DtPduFactory* fact);
#else
    // Adds creators for all DIS 2.0.4 / IEEE1278.1 PDUs to fact.
    // Includes ViewControl and LgrControl
    static void addDis204Creators(DtPduFactory* fact);

    // Adds creators for all DIS 2.1.4 / IEEE1278.1a PDUs to fact.
    // (Does not include 2.0.4 / IEEE1278.1 PDUs).
    static void addDis214Creators(DtPduFactory* fact);
#endif
};
#endif

```

dRefEmSysList.h

```
#if !DtHLA

#include "dRefObjList.h"
#include "dRefLEmitrSys.h"
#include "globalObjDes.h"

// class DtReflectedEmitterSystemList:
//
// Instances of this class provide a protocol independant interface
// to a
// list of reflected emitters in a DIS environment

#if DtDISVERS != 3
#else
#define DtElectromagneticEmissionPdu DtEmissionPdu
#endif

class DT_DLL_VL DtReflectedEmitterSystemList : public DtReflectedObjectList
{
public:

    // Constructor
    DtReflectedEmitterSystemList(DtExerciseConn* exConn);

    // Destructor
    virtual ~DtReflectedEmitterSystemList();

    // lookup an emitter from its object ID and system ID
    virtual DtReflectedEmitterSystem* lookup(
        const DtGlobalObjectDesignator& emittingEntity,
        int systemId) const;

    // get the first and last elements of the entity list
    virtual DtReflectedEmitterSystem* first() const;
    virtual DtReflectedEmitterSystem* last() const;

    // Emitter added callback registration/deregistration
    virtual void addEmitterAdditionCallback(
        DtEmitterSystemCallbackFunction cb, void* userData);
    virtual void removeEmitterAdditionCallback(
        DtEmitterSystemCallbackFunction cb, void* userData);

    // Emitter removed callback registration/deregistration
    virtual void addEmitterRemovalCallback(
        DtEmitterSystemCallbackFunction cb, void* userData);
```

```

    virtual void removeEmitterRemovalCallback(
        DtEmitterSystemCallbackFunction cb, void* userData);

    // Print the SRs of all objects
    virtual void print();

protected:

    // Assignment operator is protected because it is not fully
    implemented
    DtReflectedEmitterSystemList& operator=(
        const DtReflectedEmitterSystemList& orig);

    // Copy constructor is protected because it is not fully imple-
    mented
    DtReflectedEmitterSystemList(const DtReflectedEmitterSystemList&
    orig);

    // Create a new object from a state message.
    virtual DtReflectedObject* newReflectedObj(const DtStateMsg&
    msg);

    // process an incoming state message
    virtual DtReflectedObject* processStateMessage(const DtStateMsg&
    msg);

    // Called after an entity is added and before addition callbacks
    are
    // processed. Right now, this function only calls DtInfo.
    virtual void objectAdded(DtReflectedObject* obj);

    // Get a hash key from a state message.
    virtual const DtBaseHashKey&
        hashKeyFromStateMessage(const DtStateMsg* msg);

protected:

    // callback for the exercise connection
    static void emEmissionCb(DtElectromagneticEmissionPdu* pdu, void*
    usr);

};
#endif

```

dRef/AggList.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "dRefObjList.h"
#include "dReflectedAgg.h"

// class DtReflectedAggregateList:
//
// Instances of this class provide a protocol independant interface
// to a
// list of reflected aggregates in a DIS environment

class DT_DLL_VL DtReflectedAggregateList : public DtReflectedObjectList
{
public:

    // Constructor
    DtReflectedAggregateList(DtExerciseConn* exConn,
        DtBoolean smoothCapable = DtFalse);

    // Destructor
    virtual ~DtReflectedAggregateList();

    // lookup an aggregate from its object ID
    virtual DtReflectedAggregate* lookup(const DtObjectId& id) const;

    // get the first and last elements of the aggregate list
    virtual DtReflectedAggregate* first() const;
    virtual DtReflectedAggregate* last() const;

    // Aggregate added callback registration/deregistration
    virtual void addAggregateAdditionCallback(
        DtAggregateCallbackFunction cb, void* userData);
    virtual void removeAggregateAdditionCallback(
        DtAggregateCallbackFunction cb, void* userData);

    // Aggregate removed callback registration/deregistration
    virtual void addAggregateRemovalCallback(
        DtAggregateCallbackFunction cb, void* userData);
    virtual void removeAggregateRemovalCallback(
        DtAggregateCallbackFunction cb, void* userData);

    // Print the SRs of all objects
    virtual void print();
};
```

```

    // process an incoming aggregate state pdu. Then calls down to
    base class
    // processStateMessage().
    virtual DtReflectedAggregate* processAggregateState(
        const DtAggregateStatePdu& pdu);

protected:

    // Assignment operator is protected because it is not fully
    implemented
    DtReflectedAggregateList& operator=(
        const DtReflectedAggregateList& orig);

    // Copy constructor is protected because it is not fully imple-
    mented
    DtReflectedAggregateList(const DtReflectedAggregateList& orig);

    // Create a new object from a state message.
    virtual DtReflectedObject* newReflectedObj(const DtStateMsg&
msg);

    // process an incoming state message
    virtual DtReflectedObject* processStateMessage(const DtStateMsg&
msg);

    // Called after an aggregate is added and before addition call-
    backs are
    // processed. Right now, this function only calls DtInfo.
    virtual void objectAdded(DtReflectedObject* obj);

    // Get a hash key from a state message.
    virtual const DtBaseHashKey&
        hashKeyFromStateMessage(const DtStateMsg* msg);

    // Create a new reflected aggregate
    virtual DtReflectedAggregate* newReflectedAggregate(
        const DtEntityIdentifier& id, const DtEntityType& type) const;

    // Called after an aggregate is added and before addition call-
    backs are
    // processed. Right now, this function only calls DtInfo.
    virtual void aggregateAdded(DtReflectedAggregate* agg);

protected:

    // callback for the exercise connection
    static void aggregateStateCb(DtAggregateStatePdu* pdu, void*
usr);

protected:

```

```
DtBoolean mySmoothCapable;  
DtObjectIdHashKey* myHashKey;  
};  
  
#endif  
#endif
```


dReflectedAgg.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "dReflectedObj.h"
#include "dAggDec.h"
#include "aggStatePdu.h"
#include "globalObjDes.h"
#include "objIdHashKey.h"

// Callback function that takes a DtReflectedAggregate and user data
typedef void (*DtAggregateCallbackFunction)(
    DtReflectedAggregate* obj, void* userData);

// class DtReflectedAggregate:
//
// Instances of this class provide a protocol independant interface
// to a
// reflected aggregate in a DIS environment

class DT_DLL_VL DtReflectedAggregate : public DtReflectedObject
{
public:
    // Constructor
    DtReflectedAggregate(DtExerciseConn* conn,
        const DtEntityIdentifier& id,
        const DtEntityType& type);

    // Destructor
    virtual ~DtReflectedAggregate();

    // Returns a pointer to the AggregateStateRepository - through
    // which you can inspect current state information.
    virtual DtAggregateStateRepository* aggregateStateRep() const;

    // Synonym for aggregateStateRep()
    virtual DtAggregateStateRepository* asr() const;

    // Returns a pointer to the StateRepository - through
    // which you can inspect current state information.
    // Implementation for abstract base class. Calls asr().
    virtual DtStateRepository* stateRep() const;

    // Returns a pointer to the Decoder being used to decode
    // state messages into our ASR.

```

```
virtual DtAggregateStateDecoder* decoder();

// Returns the aggregate identifier
virtual const DtEntityIdentifier& aggregateID() const;

// Get the aggregate's id. (For DIS, DtObjectId is DtEntityIdentifier).
virtual const DtObjectId& id() const;

// Get the aggregate's object's global object designator -
// for DIS, it's the same as its DtEntityIdentifier.
virtual const DtGlobalObjectDesignator& globalId() const;

// Process an aggregate state PDU
// Implementation for abstract base class.
virtual void processStateMessage(const DtStateMsg& msg);

// get the next and previous elements in the reflected aggregate
list
virtual DtReflectedAggregate* next() const;
virtual DtReflectedAggregate* prev() const;

// get the next and previous elements, wrap at the ends of the
list
virtual DtReflectedAggregate* wrapNext() const;
virtual DtReflectedAggregate* wrapPrev() const;

// Entity state callback registration/deregistration
virtual void addPostUpdateCallback(
    DtAggregateCallbackFunction cb, void* userData);
virtual void removePostUpdateCallback(
    DtAggregateCallbackFunction cb, void* userData);

// Get a hashkey for this object.
virtual const DtBaseHashKey& hashKey() const;

// Get a string identification of this object.
virtual const DtString& idString() const;

// Process an entity state PDU. Also calls down to base class
// processStateMessage().
virtual void processAggregateState(const DtAggregateStatePdu&
pdu);

protected:

// copy constructor
DtReflectedAggregate(const DtReflectedAggregate& orig);

// assignment operator
```

```
DtReflectedAggregate& operator=(const DtReflectedAggregate&  
orig);
```

```
protected:
```

```
    DtEntityIdentifier myId;  
    DtAggregateStateRepository* myAggregateStateRep;  
    DtAggregateStateDecoder* myAggregateDecoder;  
    DtObjectIdHashKey myHashKey;
```

```
};  
#endif  
#endif
```

dReflectedEnt.h

```
#if !DtHLA

#include "dReflectedObj.h"
#include "dEntityDec.h"
#include "esPdu.h"
#include "globalObjDes.h"
#include "objIdHashKey.h"

// Callback function that takes a DtReflectedEntity and user data
typedef void (*DtEntityCallbackFunction)(
    DtReflectedEntity* obj, void* userData);

// class DtReflectedEntity:
//
// Instances of this class provide a protocol independant interface
// to a
// reflected entity in a DIS environment

class DT_DLL_VL DtReflectedEntity : public DtReflectedObject
{
public:

    // Constructor
    DtReflectedEntity(DtExerciseConn* conn,
        const DtEntityIdentifier& id,
        const DtEntityType& type);

    // Destructor
    virtual ~DtReflectedEntity();

    // Returns a pointer to the EntityStateRepository - through
    // which you can inspect current state information.
    virtual DtEntityStateRepository* entityStateRep() const;

    // Synonym for entityStateRep()
    virtual DtEntityStateRepository* esr() const;

    // Returns a pointer to the StateRepository - through
    // which you can inspect current state information.
    // Implementation for abstract base class. Calls esr().
    virtual DtStateRepository* stateRep() const;

    // Returns a pointer to the Decoder being used to decode
    // state messages into our ESR.
    virtual DtEntityStateDecoder* decoder();
}
```

```

    // Returns the entity identifier
    virtual const DtEntityIdentifier& entityId() const;

    // Get the entity's id. (For DIS, DtObjectId is DtEntityIdentifier).
    virtual const DtObjectId& id() const;

    // Get the entity's local id. For DIS - same as id().
    virtual const DtObjectId& localId() const;

    // Get the entity's object's global object designator -
    // for DIS, it's the same as its DtEntityIdentifier.
    virtual const DtGlobalObjectDesignator& globalId() const;

    // Process an entity state PDU. Calls processEntityState().
    virtual void processStateMessage(const DtStateMsg& msg);

    // get the next and previous elements in the reflected entity
    list
    virtual DtReflectedEntity* next() const;
    virtual DtReflectedEntity* prev() const;

    // get the next and previous elements, wrap at the ends of the
    list
    virtual DtReflectedEntity* wrapNext() const;
    virtual DtReflectedEntity* wrapPrev() const;

    // Entity state callback registration/deregistration
    virtual void addPostUpdateCallback(
        DtEntityCallbackFunction cb, void* userData);
    virtual void removePostUpdateCallback(
        DtEntityCallbackFunction cb, void* userData);

    // Get a hashkey for this object.
    virtual const DtBaseHashKey& hashKey() const;

    // Get a string identification of this object.
    virtual const DtString& idString() const;

    // Process an entity state PDU. Also calls down to base class
    // processStateMessage().
    virtual void processEntityState(const DtEntityStatePdu& pdu);

protected:

    // copy constructor
    DtReflectedEntity(const DtReflectedEntity& orig);

    // assignment operator
    DtReflectedEntity& operator=(const DtReflectedEntity& orig);

```

protected:

```
DtEntityIdentifier myId;  
DtEntityStateRepository* myEntityStateRep;  
DtEntityStateDecoder* myEntityDecoder;  
DtObjectIdHashKey myHashKey;
```

```
};  
#endif
```

dReflectedObj.h

```

#if !DtHLA

#include "chooseDefines.h"
#include "vllist.h"
#include "vlTime.h"
#include "refObjCbInfo.h"
#include "hostStructs.h"
#include "str.h"

// class DtReflectedObject:
//
// Instances of DtReflectedObject are used to represent a simulation
// entity reflected across a network (i.e. this is not an full
// simulation running on this machine).

class DT_DLL_VL DtReflectedObject : public DtListedObj
{
public:

    // constructor
    DtReflectedObject(DtExerciseConn* conn);

    // destructor
    virtual ~DtReflectedObject();

    // Returns a pointer to the StateRepository - through
    // which you can inspect current state information.
    virtual DtStateRepository* stateRep() const = 0;

    // Synonym for stateRep()
    virtual DtStateRepository* sr() const;

    // Returns the value of simulation time at the time
    // an update was last received for the entity.
    virtual DtTime lastSimTimeUpdated() const;

    // Process an entity state PDU
    virtual void processStateMessage(const DtStateMsg& msg);

    // Object state callback registration/deregistration
    virtual void baseAddPostUpdateCallback(
        DtReflectedObjectCallbackFunction cb, void* userData);
    virtual void baseRemovePostUpdateCallback(
        DtReflectedObjectCallbackFunction cb, void* userData);

    // Get the next remote object.

```

```
virtual DtReflectedObject* nextObj() const;

// Get the previous remote object.
virtual DtReflectedObject* prevObj() const;

// Get the next remote object.
// Will cycle back to the first remote object.
virtual DtReflectedObject* wrapNextObj() const;

// Get the previous remote object.
// Will cycle back to the last remote object.
virtual DtReflectedObject* wrapPrevObj() const;

// Get a hashkey for this object.
virtual const DtBaseHashKey& hashKey() const = 0;

// Get a string identification of this object.
virtual const DtString& idString() const = 0;

protected:

    // copy constructor
    DtReflectedObject(const DtReflectedObject& orig);

    // assignment operator
    DtReflectedObject& operator=(const DtReflectedObject& orig);

    // process the post update callbacks
    virtual void processPostUpdateCallbacks();

protected:

    DtExerciseConn* myConn;
    DtIntrusiveList myStateUpdateCallbacks;
    DtTime myLastSimTimeUpdated;

};
#endif
```


dReflEmitrSys.h

```

#if !DtHLA

#include "emitterSysSR.h"
#include "dReflectedObj.h"
#include "sysHashKey.h"

// Callback function that takes a DtReflectedObject and user data
typedef void (*DtEmitterSystemCallbackFunction)(
    DtReflectedEmitterSystem* obj, void* userData);

// class DtReflectedEmitterSystem:
//
// Instances of DtReflectedEmitterSystem represent EmitterSystem
// object simulated by remote applications. State data can be
// viewed
// through it's state repository.

class DT_DLL_VL DtReflectedEmitterSystem : public DtReflectedObject
{
public:

    // Constructor
    DtReflectedEmitterSystem(DtExerciseConn* conn,
        const DtGlobalObjectDesignator& emittingEntity,
        DtU8 systemNumber);

    // destructor
    virtual ~DtReflectedEmitterSystem();

    // Get the emitter system's state repository.
    virtual DtEmitterSystemRepository* emitterSysRep() const;

    // Synonym for emitterSysRep()
    virtual DtEmitterSystemRepository* esr() const;

    //s a pointer to the StateRepository - through
    D [(//)-60which you can inspect current state information.
    // Implementation for abstract base class. Calls esr.
    virtual DtStateRepository* stateRep()) const;

    //s an electronic emissions[(//)-60PDU
    D [(//)-60Implementation for abstract base class.
    virtual void processStateMessage(const DtStateMsg& msg);

    // Get the next remote emitterSystem.

```

```
// Get the previous remote emitterSystem.
virtual DtReflectedEmitterSystem* prev() const;

// Get the next remote emitterSystem.
// Will cycle back to the first remote emitterSystem.
virtual DtReflectedEmitterSystem* wrapNext() const;

// Get the previous remote emitterSystem.
// Will cycle back to the last remote emitterSystem.
virtual DtReflectedEmitterSystem* wrapPrev() const;

// emitterSystem state callback registration/deregistration
virtual void addPostUpdateCallback(
    DtEmitterSystemCallbackFunction cb, void* userData);

virtual void removePostUpdateCallback(
    DtEmitterSystemCallbackFunction cb, void* userData);

// Get a hashkey for this object.
virtual const DtBaseHashKey& hashKey() const;

// Get a string identification of this object.
virtual const DtString& idString() const;

protected:

    // copy constructor
    DtReflectedEmitterSystem(const DtReflectedEmitterSystem& orig);

    // assignment operator
    DtReflectedEmitterSystem& operator=(
        const DtReflectedEmitterSystem& orig);

protected:

    DtEmitterSystemRepository myStateRepository;
    DtSystemHashKey myHashKey;
    DtString myIdString;

};
#endif
```

dReflEntList.h

```

#if !DtHLA

#include "dRefObjList.h"
#include "dReflectedEnt.h"

// class DtReflectedEntityList:
//
// Instances of this class provide a protocol independant interface
// to a
// list of reflected entities in a DIS environment

class DT_DLL_VL DtReflectedEntityList : public DtReflectedObjectList
{
public:

    // Constructor
    DtReflectedEntityList(DtExerciseConn* exConn,
        DtBoolean smoothCapable = DtFalse);

    // Destructor
    virtual ~DtReflectedEntityList();

    // lookup an entity from its object ID
    virtual DtReflectedEntity* lookup(const DtObjectId& id) const;

    // get the first and last elements of the entity list
    virtual DtReflectedEntity* first() const;
    virtual DtReflectedEntity* last() const;

    // Entity added callback registration/deregistration
    virtual void addEntityAdditionCallback(
        DtEntityCallbackFunction cb, void* userData);
    virtual void removeEntityAdditionCallback(
        DtEntityCallbackFunction cb, void* userData);

    // Entity removed callback registration/deregistration
    virtual void addEntityRemovalCallback(
        DtEntityCallbackFunction cb, void* userData);
    virtual void removeEntityRemovalCallback(
        DtEntityCallbackFunction cb, void* userData);

    // Print the SRs of all objects
    virtual void print();

    // process an incoming entity state pdu. Then calls down to base
    class

```

```
// processStateMessage().
virtual DtReflectedEntity* processEntityState(const DtEntityS-
tatePdu& pdu);

protected:

    // Assignment operator is protected because it is not fully
    implemented
    DtReflectedEntityList& operator=(
        const DtReflectedEntityList& orig);

    // Copy constructor is protected because it is not fully imple-
    mented
    DtReflectedEntityList(const DtReflectedEntityList& orig);

    // Create a new object from a state message. Calls newReflected-
    dEntity()
    // to create a new entity, and then tells the esr to use a
    smoother if
    // smoothing is enabled.
    virtual DtReflectedObject* newReflectedObj(const DtStateMsg&
    msg);

    // process an incoming state message. Calls processEntityS-
    tate().
    virtual DtReflectedObject* processStateMessage(const DtStateMsg&
    msg);

    // Called after an entity is added and before addition callbacks
    are
    // processed. Calls entityAdded().
    virtual void objectAdded(DtReflectedObject* obj);

    // Called when an entity sends its final PDU. Currently, this
    removes
    // and deletes the entity.
    virtual void finalPduIsSetAction(DtReflectedEntity* entity);

    // Get a hash key from a state message.
    virtual const DtBaseHashKey&
        hashKeyFromStateMessage(const DtStateMsg* msg);

    // Create a new reflected entity
    virtual DtReflectedEntity* newReflectedEntity(const DtEntityIden-
    tifier& id,
        const DtEntityType& type) const;

    // Called after an entity is added and before addition callbacks
    are
    // processed. Right now, this function only calls DtInfo.
```

```
    virtual void entityAdded(DtReflectedEntity* ent);  
  
protected:  
  
    // callback for the exercise connection  
    static void entityStateCb(DtEntityStatePdu* pdu, void* usr);  
  
protected:  
  
    DtBoolean mySmoothCapable;  
    DtObjectIdHashKey* myHashKey;  
};  
  
#endif
```

dRefObjList.h

```
#if !DtHLA

#include "chooseDefines.h"
#include "MachineTypes.h"
#include "vlTime.h"
#include "refObjCbInfo.h"
#include "hashlist.h"

// class DtReflectedObjectList:
//
// Instances of DtReflectedObjectList are list of instances of
// classes
// derived from DtReflectedObject. A DtReflectedObjectList manages
// the comings
// and goings of its reflected objects based on discoveries and
// removes
// received from the exercise connection.

class DT_DLL_VL DtReflectedObjectList
{
public:

    // Constructor
    DtReflectedObjectList(DtExerciseConn* exConn);

    // destructor
    // Warning: makes non-virtual call to removeAndDelete
    virtual ~DtReflectedObjectList();

    // Get the first/last object in the list.
    virtual DtReflectedObject* firstObj() const;
    virtual DtReflectedObject* lastObj() const;

    // Get number of objects
    virtual int count() const;

    // Get my exerciseConn.
    virtual DtExerciseConn* exerciseConn() const;

    // Object added callback registration/deregistration
    virtual void addObjectAdditionCallback(
        DtReflectedObjectCallbackFunction cb, void* userData);
    virtual void removeObjectAdditionCallback(
        DtReflectedObjectCallbackFunction cb, void* userData);

    // Object removed callback registration/deregistration
```

```

virtual void addObjectRemovalCallback(
    DtReflectedObjectCallbackFunction cb, void* userData);
virtual void removeObjectRemovalCallback(
    DtReflectedObjectCallbackFunction cb, void* userData);

// Print the SRs of all objects
virtual void print();

// Set whether or not timeout processing occurs.
// Defaults to off.
virtual void setTimeoutProcessing(DtBoolean onOff);

// Set/Get the remote object timeout period.
virtual void setTimeoutInterval(DtTime time);
virtual DtTime timeoutInterval();

protected:

    // copy constructor
    DtReflectedObjectList(const DtReflectedObjectList& orig);

    // assignment operator
    DtReflectedObjectList& operator=(const DtReflectedObjectList&
orig);

    // Create a new object from a state message.
    virtual DtReflectedObject* newReflectedObj(const DtStateMsg& msg)
= 0;

    // process an incoming state message
    virtual DtReflectedObject* processStateMessage(const DtStateMsg&
msg);

    // Remove any remote objects which have not been updated during
    // the timeout period. Gets called automatically by a postDrain-
    Callback
    // registered by the constructor. Users should not need to call
    this.
    virtual void doTimeouts();

    // Remove the remote object from the list if found and delete it.
    // Prints a message on deletion if specified.
    virtual void removeAndDelete(DtReflectedObject* obj,
        char* msg = NULL);

    // Called after a remote object has been added to the list.
    // Defaults to do nothing.
    // Subclasses can override this function to add functionality on
    // addition.
    virtual void objectAdded(DtReflectedObject* object);

```

```
// Returns whether or not the object should be timed out.
// Assumes that timeout processing is enabled.
virtual DtBoolean shouldBeTimedOut(DtReflectedObject* object);

// Empty the object list and the timeout list.
// Non-virtual since meant to be called from destructor.
// Warning: makes non-virtual call to removeAndDelete
void emptyLists();

// Process object addition/removal callbacks
virtual void processObjectAdditionCallbacks(
    DtReflectedObject* object);
virtual void processObjectRemovalCallbacks(
    DtReflectedObject* object);

// Get a hash key from a state message.
virtual const DtBaseHashKey&
    hashKeyFromStateMessage(const DtStateMsg* msg) = 0;

// Determines if a PDU should be processed or not. Right now,
this function
// returns 1.
virtual int acceptPdu(const DtStateMsg&) const;

protected:

    // calls doTimeouts
    static void doTimeoutsPostDrainCb(void* usr);

protected:

    DtExerciseConn* myExConn;
    DtIntrusiveList myObjectAddedCallbacks;
    DtIntrusiveList myObjectRemovedCallbacks;
    DtIntrusiveList myObjectList;
    DtHashlist myObjectHashlist;
    DtTime myTimeoutInterval;
    DtBoolean myTimeoutProcessingFlag;
};
#endif // DtHLA
```


dscvObjCbInfo.h

```

#if DtHLA

#include "RTI.hh"
#include "stateMsg.h"
#include "hlaObj.h"

// class DtDiscoverObjectCbInfo:
//
// Instances of DtDiscoverObjectCbInfo are used for providing
// a callback when an object which meets the federate's subscription
// criteria is discovered.

typedef void (*DtDiscoverObjectCb)(DtHlaObject* obj, void* usr);

class DT_DLL_VL DtDiscoverObjectCbInfo
{
public:

    // constructor
    DtDiscoverObjectCbInfo(RTI::ObjectClassHandle h,
                          DtDiscoverObjectCb cb,
                          void* usr);

    // destructor
    virtual ~DtDiscoverObjectCbInfo();

    // copy constructor
    DtDiscoverObjectCbInfo(
        const DtDiscoverObjectCbInfo& orig);

    // assignment operator
    DtDiscoverObjectCbInfo& operator=(
        const DtDiscoverObjectCbInfo& orig);

    // Check whether one callback info is equal to another.
    virtual int match(const DtDiscoverObjectCbInfo* const other)
const;

    // Check whether one callback info is equal to another.
    virtual int operator==(const DtDiscoverObjectCbInfo& other)
const;

    // Get the object class handle
    virtual const RTI::ObjectClassHandle& objectClass() const;

    // Get the callback function.
    virtual const DtDiscoverObjectCb& cbFunction() const;

```

```
// Get the user data.  
virtual void* userData() const;  
  
protected:  
  
    RTI::ObjectClassHandle myClass;  
    DtDiscoverObjectCb myFunc;  
    void* myUsr;  
};  
#endif
```

dStateMsg.h

```
#if !DtHLA

#include "pdu.h"

#include "chooseDefines.h"

// class DtStateMsg:
//
// Instances of DtStateMsg are used to represent state data
// communicated via the network in a DIS (as opposed to HLA) envi-
// ronment.

class DT_DLL_VL DtStateMsg : public DtPdu
{
public:

    // default constructor
    DtStateMsg();

    // destructor
    virtual ~DtStateMsg();

    // copy constructor
    DtStateMsg(const DtStateMsg& orig);

    // assignment operator
    DtStateMsg& operator=(const DtStateMsg& orig);
};
#endif
```

elecEmitPdu.h

```
#if !DtHLA

#if DtDISVERS != 3

#include "hostStructs.h"
#include "stateMsg.h"
#include "mathTypes.h"
#include "packets/elecEmitP.h"
#define DtElectromagneticEmissionStaticSize 28

#define DtEMISSION_SUCCESS 0
#define DtBAD_SYS_NUM 1
#define DtBAD_BEAM_NUM 2
#define DtBAD_TARGET_NUM 3
#define DtBAD_SYS_ADDR 4
#define DtBAD_BEAM_ADDR 5
#define DtBAD_TARGET_ADDR 6
#define DtINCONSISTENT_SYS_LENGTHS_PDU_LENGTH 7
#define DtINCONSISTENT_BEAM_LENGTHS_SYS_LENGTH 8
#define DtINCONSISTENT_NUM_TARGETS_BEAM_LENGTH 9

class DT_DLL_VL DtFundamentalParameterData
{
public:
    float frequency;
    float frequencyRange;
    float effectiveRadiatedPower;
    float pulseRepetitionFrequency;
    float pulseWidth;
    float beamAzCenter;
    float beamAzSweep;
    float beamElCenter;
    float beamElSweep;
    float beamSweepSync;
    void printData() const;
};

class DT_DLL_VL DtTrackJamTarget
{
public:
    DtEntityIdentifier entityId;
    int emitterId;
    int beamId;
    void printData() const;
};

class DT_DLL_VL DtElectromagneticEmissionPdu;
```

```

typedef void (*DtElectromagneticEmissionPduCb)(DtElectromagneticEmissionPdu *pdu, void *usr);

class DT_DLL_VL DtElectromagneticEmissionPdu : public DtStateMsg
{
public:

/***** STANDARD CONSTRUCTORS *****/

    DtElectromagneticEmissionPdu(DtBufferPtr buffer =
DtUSE_INTERNAL_BUFFER);
    DtElectromagneticEmissionPdu(const DtNetElectromagneticEmissionPdu *initial,
DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    static DtPdu *create(const DtNetPacket *initial,
DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    static void addCallback(DtExerciseConn *conn,
DtElectromagneticEmissionPduCb cb, void *usr);
    static void removeCallback(DtExerciseConn *conn,
DtElectromagneticEmissionPduCb cb, void *usr);

/***** INSPECTOR FUNCTIONS *****/

    const DtEntityIdentifier &emittingEntityId() const;
    const DtEventID &eventId() const;
    DtEmissionPduType stateUpdateIndicator() const;
    int numSystems() const;

/***** FOR A PARTICULAR SYSTEM *****/
    int systemDataWords(int sysNum, int *retcode = NULL) const;
    int numBeams(int sysNum, int *retcode = NULL) const;
    DtEmitterSysName systemName(int sysNum, int *retcode = NULL)
const;
    DtEmissionFunc systemFunction(int sysNum, int *retcode = NULL)
const;
    int emitterId(int sysNum, int *retcode = NULL) const;
    DtConstVector location(int sysNum, int *retcode = NULL) const;

/***** FOR A PARTICULAR BEAM *****/
    int beamDataWords(int sysNum, int beamNum, int *retcode = NULL)
const;
    int beamId(int sysNum, int beamNum, int *retcode = NULL) const;
    int beamParameterIndex(int sysNum, int beamNum,
int *retcode = NULL) const;
    const DtFundamentalParameterData
&parameterData(int sysNum, int beamNum, int *retcode = NULL)
const;
    int beamFunction(int sysNum, int beamNum, int *retcode = NULL)
const;

```

```
int numTargetsInBeam(int sysNum, int beamNum,
    int *retcode = NULL) const;
DtTrackingJamming trackingAllTargets(int sysNum,
    int beamNum,
    int *retcode = NULL) const;
unsigned long jammingModeSequence(int sysNum,
    int beamNum,
    int *retcode = NULL) const;
const DtTrackJamTarget &trackJamTarget(int sysNum, int beamNum,
    int targetNum,
    int *retcode = NULL) const;

/***** MUTATOR FUNCTIONS *****/

void setEmittingEntityId(const DtEntityIdentifier &id);
void setEventId(const DtEventID &id);
void setStateUpdateIndicator(DtEmissionPduType indicator);
void setNumSystems(int num);

/***** FOR A PARTICULAR SYSTEM *****/
void setNumBeams(int sysNum, int numBeams, int *retcode = NULL);
void setSystemName(int sysNum, DtEmitterSysName name,
    int *retcode = NULL);
void setSystemFunction(int sysNum, DtEmissionFunc func,
    int *retcode = NULL);
void setEmitterId(int sysNum, int id, int *retcode = NULL);
void setLocation(int sysNum, DtConstVector loc, int *retcode =
NULL);

/***** FOR A PARTICULAR BEAM *****/
void setBeamId(int sysNum, int beamNum, int id, int *retcode =
NULL);
void setBeamParameterIndex(int sysNum, int beamNum, int bpIndex,
    int *retcode = NULL);
void setParameterData(int sysNum, int beamNum,
    const DtFundamentalParameterData &data,
    int *retcode = NULL);
void setBeamFunction(int sysNum, int beamNum,
    int func, int *retcode = NULL);
void setNumTargetsInBeam(int sysNum, int beamNum,
    int num, int *retcode = NULL);
void setTrackingAllTargets(int sysNum, int beamNum,
    DtTrackingJamming trackJamAll,
    int *retcode = NULL);
void setJammingModeSequence(int sysNum, int beamNum,
    unsigned long seq, int *retcode = NULL);
void setTrackJamTarget(int sysNum, int beamNum, int targetNum,
    const DtTrackJamTarget &targ, int *retcode = NULL);

void printData() const;
```

```
protected:
    DtPduKind internalGetPduKind() const {return DtElectromagneticE-
missionPduKind;}
    DtEntityIdentifier cacheId;
    DtEventID cacheEvent;
    DtVector cacheLoc;
    DtFundamentalParameterData cacheData;
    DtTrackJamTarget cacheTarget;
    DtNetSystemData *emitterSystemAddr(int sysNum, int *retcode =
NULL) const;
    DtNetBeamData *emitterBeamAddr(int sysNum, int beamNum,
int *retcode = NULL) const;
    DtNetEmitterTrackJam *targetAddr(int sysNum, int beamNum,
int targetNum, int *retcode = NULL) const;
    int computedSystemDataLength(int sysNum) const;
    int computedBeamDataLength(int sysNum, int beamNum) const;
    void printSystem(int i) const;
    void printBeam(int i, int j) const;
    int addressInPacket(void *addr) const;
    int computeStatus() const;
};

#endif // DtDISVERS
#endif // DtHLA
```

embedSystemSR.h

```
#include "disEnums.h"
#include "packets/transmitterP.h"
#include "hostStructs.h"
#include "stateRep.h"

// class DtEmbeddedSystemRepository:
//
// Instances of DtEmbeddedSystemRepository are

class DT_DLL_VL DtEmbeddedSystemRepository : public DtStateRepository
{
public:

    // default constructor
    DtEmbeddedSystemRepository();

    // copy constructor
    DtEmbeddedSystemRepository(const DtEmbeddedSystemRepository&
orig);

    // destructor
    virtual ~DtEmbeddedSystemRepository();

    // assignment operator
    DtEmbeddedSystemRepository& operator=(
        const DtEmbeddedSystemRepository& orig);

    // return an empty object of the same type as this
    virtual DtStateRepository* clone() const;

    static DtEmbeddedSystemRepository* create();

    // Set/Get the entity identifier.
    virtual void setEntityId(const DtEntityIdentifier& id);
    virtual const DtEntityIdentifier& entityId() const;

    // Set/Get the host identifier.
    virtual void setHostId(const DtGlobalObjectDesignator& id);
    virtual const DtGlobalObjectDesignator& hostId() const;

    // Set/Get the relative position.
    virtual void setRelativePosition(DtConstVector position);
    virtual DtConstVector relativePosition() const;

    virtual void printData() const;
```



```
//embedded system class attributes
protected:
    DtEntityIdentifier      myEntityId;
    DtGlobalObjectDesignator myHostId;
    DtVector                myRelativePosition;
};
```

emissionPdu.h

```
#if !DtHLA

#if DtDISVERS == 3

#include "hostStructs.h"
#include "stateMsg.h"
#include "mathTypes.h"
#include "packets/emmissionP.h" /* sic */

#define DtEmissionStaticSize 28

#define DtEMISSION_SUCCESS 0
#define DtBAD_SYS_NUM 1
#define DtBAD_BEAM_NUM 2
#define DtBAD_TARGET_NUM 3
#define DtBAD_SYS_ADDR 4
#define DtBAD_BEAM_ADDR 5
#define DtBAD_TARGET_ADDR 6
#define DtINCONSISTENT_SYS_LENGTHS_PDU_LENGTH 7
#define DtINCONSISTENT_BEAM_LENGTHS_SYS_LENGTH 8
#define DtINCONSISTENT_NUM_TARGETS_BEAM_LENGTH 9

class DT_DLL_VL DtFundamentalParameterData
{
public:
    float frequency;
    float frequencyRange;
    float effectiveRadiatedPower;
    float pulseRepetitionFrequency;
    float pulseWidth;
    float beamAzCenter;
    float beamAzSweep;
    float beamElCenter;
    float beamElSweep;
    float beamSweepSync;
    void printData() const;
};

class DT_DLL_VL DtTrackJamTarget
{
public:
    DtEntityIdentifier entityId;
    int emitterId;
    int beamId;
    void printData() const;
};

class DT_DLL_VL DtEmissionPdu;
```

```

typedef void (*DtEmissionPduCb)(DtEmissionPdu *pdu, void *usr);

class DT_DLL_VL DtEmissionPdu : public DtStateMsg
{
public:

/***** STANDARD CONSTRUCTORS *****/

    DtEmissionPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtEmissionPdu(const DtNetEmissionPdu *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    static DtPdu *create(const DtNetPacket *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    static void addCallback(DtExerciseConn *conn,
        DtEmissionPduCb cb, void *usr);
    static void removeCallback(DtExerciseConn *conn,
        DtEmissionPduCb cb, void *usr);

/***** INSPECTOR FUNCTIONS *****/

    const DtEntityIdentifier &emittingEntityId() const;
    const DtEventID &eventId() const;
    DtEmissionPduType stateUpdateIndicator() const;
    int numSystems() const;

/***** FOR A PARTICULAR SYSTEM *****/
    int systemDataWords(int sysNum, int *retcode = NULL) const;
    int numBeams(int sysNum, int *retcode = NULL) const;
    DtEmitterSysName systemName(int sysNum, int *retcode = NULL)
const;
    DtEmissionFunc systemFunction(int sysNum, int *retcode = NULL)
const;
    int emitterId(int sysNum, int *retcode = NULL) const;
    DtConstVector location(int sysNum, int *retcode = NULL) const;

/***** FOR A PARTICULAR BEAM *****/
    int beamDataWords(int sysNum, int beamNum, int *retcode = NULL)
const;
    int beamId(int sysNum, int beamNum, int *retcode = NULL) const;
    int beamParameterIndex(int sysNum, int beamNum,
        int *retcode = NULL) const;
    const DtFundamentalParameterData
&parameterData(int sysNum, int beamNum, int *retcode = NULL)
const;
    int beamFunction(int sysNum, int beamNum, int *retcode = NULL)
const;
    int numTargetsInBeam(int sysNum, int beamNum,
        int *retcode = NULL) const;
    DtTrackingJamming trackingAllTargets(int sysNum,

```

```
        int beamNum,
        int *retcode = NULL) const;
unsigned long jammingModeSequence(int sysNum,
        int beamNum,
        int *retcode = NULL) const;
const DtTrackJamTarget &trackJamTarget(int sysNum, int beamNum,
        int targetNum,
        int *retcode = NULL) const;

/***** MUTATOR FUNCTIONS *****/

void setEmittingEntityId(const DtEntityIdentifier &id);
void setEventId(const DtEventID &id);
void setStateUpdateIndicator(DtEmissionPduType indicator);
void setNumSystems(int num);

/***** FOR A PARTICULAR SYSTEM *****/
void setNumBeams(int sysNum, int numBeams, int *retcode = NULL);
void setSystemName(int sysNum, DtEmitterSysName name,
        int *retcode = NULL);
void setSystemFunction(int sysNum, DtEmissionFunc func,
        int *retcode = NULL);
void setEmitterId(int sysNum, int id, int *retcode = NULL);
void setLocation(int sysNum, DtConstVector loc, int *retcode =
NULL);

/***** FOR A PARTICULAR BEAM *****/
void setBeamId(int sysNum, int beamNum, int id, int *retcode =
NULL);
void setBeamParameterIndex(int sysNum, int beamNum, int bpIndex,
        int *retcode = NULL);
void setParameterData(int sysNum, int beamNum,
        const DtFundamentalParameterData &data,
        int *retcode = NULL);
void setBeamFunction(int sysNum, int beamNum,
        int func, int *retcode = NULL);
void setNumTargetsInBeam(int sysNum, int beamNum,
        int num, int *retcode = NULL);
void setTrackingAllTargets(int sysNum, int beamNum,
        DtTrackingJamming trackJamAll,
        int *retcode = NULL);
void setJammingModeSequence(int sysNum, int beamNum,
        unsigned long seq, int *retcode = NULL);
void setTrackJamTarget(int sysNum, int beamNum, int targetNum,
        const DtTrackJamTarget &targ, int *retcode = NULL);

void printData() const;
protected:
DtPduKind internalGetPduKind() const {return DtEmissionPduKind;}
DtEntityIdentifier cacheId;
```

```
DtEventID cacheEvent;
DtVector cacheLoc;
DtFundamentalParameterData cacheData;
DtTrackJamTarget cacheTarget;
DtNetSystemData *emitterSystemAddr(int sysNum, int *retcode =
NULL) const;
DtNetBeamData *emitterBeamAddr(int sysNum, int beamNum,
int *retcode = NULL) const;
DtNetEmitterTrackJam *targetAddr(int sysNum, int beamNum,
int targetNum, int *retcode = NULL) const;
int computedSystemDataLength(int sysNum) const;
int computedBeamDataLength(int sysNum, int beamNum) const;
void printSystem(int i) const;
void printBeam(int i, int j) const;
int addressInPacket(void *addr) const;
int computeStatus() const;
};

#endif // DtDISVERS
#endif // !DtHLA
```

emitterSysDec.h

```
#if DtHLA

#ifndef DtEmitterSystemDecoder_H_
#define DtEmitterSystemDecoder_H_

#include "chooseDefines.h"
#include "baseEmbedDec.h"

// This is used to declare attribute decoding functions
// which are members of the class.
#define DtDECLARE_EMITTERSYS_ATTR_DECODER(attrName) \
    DtDECLARE_ATTR_DECODER(DtEmitterSystemRepository, attrName)

// class DtEmitterSystemDecoder:
//
// Instances of DtEmitterSystemDecoder are

class DT_DLL_VL DtEmitterSystemDecoder : public DtBaseEmbeddedSystemDecoder
{
public:

    // default constructor
    DtEmitterSystemDecoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtEmitterSystemDecoder();

    // copy constructor
    DtEmitterSystemDecoder(const DtEmitterSystemDecoder& orig);

    // assignment operator
    DtEmitterSystemDecoder&operator=(const DtEmitterSystemDecoder&
orig);

public:

    // Create a DtBaseEntityDecoder. Caller is responsible for deletion.
    static DtHlaStateDecoder* create(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

protected:
```

```

    // Add decoding functions for the attributes.
    // Note: Non-virtual since meant to be called from constructors
to
    //          add decoders.
    void addMyDecoders();

    // static decode<AttributeName> member functions.
    DtDECLARE_EMITTERSYS_ATTR_DECODER(EmitterFunction);
    DtDECLARE_EMITTERSYS_ATTR_DECODER(EmitterName);
    DtDECLARE_EMITTERSYS_ATTR_DECODER(EmitterNumber);
    DtDECLARE_EMITTERSYS_ATTR_DECODER(EventID);
};
// These macros are used to define simple attribute decoder func-
tions
// (members of a derived DtHlaStateDecoder that decode
// individual attributes). The simple decoding is just to
// convert the value pulled out of the ahvps to a host type using
// the cast expression castExpr, and pass it to the mutator func-
tion.
#define DtDEFINE_SIMPLE_EMITTERSYS_ATTR_DECODER_WITH_CAST( \
    attrName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_ATTR_DECODER_WITH_CAST( \
        DtEmitterSystemDecoder, DtEmitterSystemRepository, \
        attrName, netType, mutator, castExpr)

#define DtDEFINE_SIMPLE_EMITTERSYS_ATTR_DECODER( \
    attrName, netType, inspector) \
    DtDEFINE_SIMPLE_EMITTERSYS_ATTR_DECODER_WITH_CAST( \
        attrName, netType, inspector, (netType))

#endif    //DtEmitterSystemDecoder_H_

```

emitterSysEnc.h

```
#if DtHLA

#ifndef DtEmitterSystemEncoder_H_
#define DtEmitterSystemEncoder_H_

#include "chooseDefines.h"
#include "baseEmbedEnc.h"

// This is used to declare static member functions for determining
// whether or not an attribute needs to be encoded.
#define DtDECLARE_EMITTERSYS_ATTR_CHECKER(attrName) \
    DtDECLARE_ATTR_CHECKER(DtEmitterSystemRepository, attrName)

// This is used to declare attribute encoding functions
// which are members of the class.
#define DtDECLARE_EMITTERSYS_ATTR_ENCODER(attrName) \
    DtDECLARE_ATTR_ENCODER(DtEmitterSystemRepository, attrName)

// class DtEmitterSystemEncoder:
//
// Instances of DtEmitterSystemEncoder are

class DT_DLL_VL DtEmitterSystemEncoder : public DtBaseEmbeddedSystemEncoder
{
public:

    // default constructor
    DtEmitterSystemEncoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtEmitterSystemEncoder();

    // copy constructor
    DtEmitterSystemEncoder(const DtEmitterSystemEncoder& orig);

    // assignment operator
    DtEmitterSystemEncoder& operator=(const DtEmitterSystemEncoder&
orig);

public:

    // Create a DtBaseEntityEncoder. Caller is responsible for deletion.
    static DtHlaStateEncoder* create(
```



```

        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

protected:

    DtDECLARE_EMITTERSYS_ATTR_CHECKER(EmitterFunction);
    DtDECLARE_EMITTERSYS_ATTR_CHECKER(EmitterName);
    DtDECLARE_EMITTERSYS_ATTR_CHECKER(EmitterNumber);
    DtDECLARE_EMITTERSYS_ATTR_CHECKER(EventID);

    DtDECLARE_EMITTERSYS_ATTR_ENCODER(EmitterFunction);
    DtDECLARE_EMITTERSYS_ATTR_ENCODER(EmitterName);
    DtDECLARE_EMITTERSYS_ATTR_ENCODER(EmitterNumber);
    DtDECLARE_EMITTERSYS_ATTR_ENCODER(EventID);
};
// This macro is used to define simple attribute encoder functions
// (members of a derived DtHlaStateEncoder that encode
// individual attributes). The simple encoding is just to implicitly
// convert the value returned by the inspector to a netType,
// and add it to the ahvps.
#define DtDEFINE_SIMPLE_EMITTERSYS_ATTR_ENCODER( \
    attrName, netType, inspector) \
    DtDEFINE_SIMPLE_ATTR_ENCODER(DtEmitterSystemEncoder, \
        DtEmitterSystemRepository, \
        attrName, netType, inspector)

// This macro is used to define simple attribute checker functions
// (members of a derived DtHlaStateEncoder that check whether
// a particular attribute is out of date. The simple implementation
// is to just compare the values returned by the two StateReps'
// inspector functions.
#define DtDEFINE_SIMPLE_EMITTERSYS_ATTR_CHECKER( \
    attrName, inspector) \
    DtDEFINE_SIMPLE_ATTR_CHECKER(DtEmitterSystemEncoder, \
        DtEmitterSystemRepository, \
        attrName, inspector)

#endif //DtEmitterSystemEncoder_H_

```

emitterSysSR.h

```
#include "embedSystemSR.h"
#include "emitttrBeamSR.h"
#include "vllList.h"

//error codes for looking up individual beams
enum DtEmitterSysRetCode
{
    DtEmitterSuccess,
    DtBadBeamId,
    DtDuplicateBeamId
};

// class DtEmitterSystemRepository:
//
// Instances of DtEmitterSystemRepository are

class DT_DLL_VL DtEmitterSystemRepository : public DtEmbeddedSystem-
Repository
{
public:

    // default constructor
    DtEmitterSystemRepository();

    // destructor
    virtual ~DtEmitterSystemRepository();

    // copy constructor
    DtEmitterSystemRepository(const DtEmitterSystemRepository& orig);

    // assignment operator
    DtEmitterSystemRepository& operator=(const DtEmitterSystemReposi-
tory& orig);

    // return an empty object of the same type as this
    virtual DtStateRepository* clone() const;

    static DtEmitterSystemRepository* create();

    // Set/Get the emitter function.
    virtual void setEmitterFunction(DtEmissionFunc function);
    virtual DtEmissionFunc emitterFunction() const;

    // Set/Get the emitter name.
    virtual void setEmitterName(DtEmitterSysName name);
    virtual DtEmitterSysName emitterName() const;
```

```

// Set/Get the emitter number.
virtual void setEmitterNumber(DtU8 number);
virtual DtU8 emitterNumber() const;

// Set/Get Event Id - not used until RPR 0.7
virtual void setEventId(const DtEventId& id);
virtual const DtEventId& eventId() const;

virtual void printData() const;

// Return a pointer to the first BeamSR with specified beamID
virtual DtEmitterBeamRepository* beamSRById(
    DtU8 beamId,
    DtEmitterSysRetCode* retcode = NULL);

// add/remove a new emitter beam within this system
// if add a beam with duplicate ID, retcode will be set
// to DtDuplicateBeam, and beamSRById will return
// the first beamSR with the given ID.
virtual DtEmitterBeamRepository* addBeam(
    DtU8 beamId = 0, DtBeamType type = DtRegularBeam,
    DtEmitterSysRetCode *retcode = NULL);
virtual DtEmitterSysRetCode removeBeam(DtU8 beamId);

// Returns number of beams
virtual int numBeams() const;

// Returns DtList of DtEmitterBeamStateRepositories
virtual DtList* beamList();

// Returns DtTrue if the give BSR is in this ESR's list of BSRs
DtBoolean isMember(DtEmitterBeamRepository* bsr) const;

protected:
    virtual void purgeBeamList();

protected:
    DtEventId                myEventId;
    DtEmissionFunc            myEmitterFunction;
    DtEmitterSysName          myEmitterName;
    DtU8                      myEmitterNumber;

    //list of beams in this system.
    DtList                    myBeamList;
};

```

emitttrBeamSR.h

```
#include "stateRep.h"
#include "hostStructs.h"
#include "disEnums.h"
#include "vllList.h"

#if DtHLA
#include "objDesList.h"
typedef DtGlobalObjectDesignator DtTrackJamDesignator;
typedef DtGlobalObjectDesignatorList DtTrackJamDesignatorList;
#else
#include "tkJmDesLst.h"
#endif

enum DtBeamType
{
    DtRegularBeam = -1,
    DtRadarBeam = 1,
    DtTrackBeam = 1,
    DtJamBeam = 0
};

// class DtEmitterBeamRepository:
//
// Instances of DtEmitterBeamRepository are

class DT_DLL_VL DtEmitterBeamRepository : public DtStateRepository
{
public:

    // default constructor
    DtEmitterBeamRepository();

    // destructor
    virtual ~DtEmitterBeamRepository();

    // copy constructor
    DtEmitterBeamRepository(const DtEmitterBeamRepository& orig);

    // assignment operator
    DtEmitterBeamRepository& operator=(
        const DtEmitterBeamRepository& orig);

    // return an empty object of the same type as this
    virtual DtStateRepository* clone() const;

    static DtEmitterBeamRepository* create();
};
```

```
// Set/Get Azimuth Center
virtual void setAzimuthCenter(DtFloat32 center);
virtual DtFloat32 azimuthCenter() const;

// Set/Get Azimuth Sweep
virtual void setAzimuthSweep(DtFloat32 sweep);
virtual DtFloat32 azimuthSweep() const;

// Set/Get Beam Function
virtual void setBeamFunction(DtBeamFunc func);
virtual DtBeamFunc beamFunction() const;

// Set/Get Beam Id
virtual void setBeamId(DtU8 id);
virtual DtU8 beamId() const;

// Set/Get Beam Parameter Index
virtual void setBeamParameterIndex(DtU16 index);
virtual DtU16 beamParameterIndex() const;

// Set/Get Elevation Center
virtual void setElevationCenter(DtFloat32 center);
virtual DtFloat32 elevationCenter() const;

// Set/Get Elevation Sweep
virtual void setElevationSweep(DtFloat32 sweep);
virtual DtFloat32 elevationSweep() const;

// Set/Get Emitting System Id
virtual void setEmittingSystemId(const DtGlobalObjectDesignator&
id);
virtual const DtGlobalObjectDesignator& emittingSystemId() const;

// Set/Get ERP
virtual void setERP(DtFloat32 erp);
virtual DtFloat32 ERP() const;

// Set/Get Entity Id - not used after RPR 0.5
virtual void setEntityId(const DtEntityIdentifier& id);
virtual const DtEntityIdentifier& entityId() const;

// Set/Get Event Id - not used until RPR 0.7
virtual void setEventId(const DtEventId& id);
virtual const DtEventId& eventId() const;

// Set/Get Frequency
virtual void setFrequency(DtFloat32 freq);
virtual DtFloat32 frequency() const;

// Set/Get Frequency Range
```

```
virtual void setFrequencyRange(DtFloat32 freq);
virtual DtFloat32 frequencyRange() const;

// Set/Get PRF
virtual void setPRF(DtFloat32 prf);
virtual DtFloat32 PRF() const;

// Set/Get Pulse Width
virtual void setPulseWidth(DtFloat32 width);
virtual DtFloat32 pulseWidth() const;

// Set/Get Sweep Sync
virtual void setSweepSync(DtFloat32 sync);
virtual DtFloat32 sweepSync() const;

// Set/Get beam type - track, jam or neither.
virtual void setBeamType(DtBeamType flag);
virtual DtBeamType beamType() const;

// The next two accessors should only be called if the beam type
// is DtTrackBeam or DtJamBeam.

// Set/Get Jamming Mode Sequence
virtual void setJammingModeSequence(DtU32 sequence);
virtual DtU32 jammingModeSequence() const;

// Get set of track jam targets. Returns pointer to internal
object.
// Do not delete. It will be deleted when the SR is deleted.
virtual const DtTrackJamDesignatorList& trackJamTargets() const;
virtual DtTrackJamDesignatorList* trackJamTargets();

// Set/Get the high density track/jam flag
virtual void setHighDensity(DtBoolean val);
virtual DtBoolean highDensity() const;

virtual void printData() const;

protected:

// Prints track/jam target information
virtual void printTrackJam() const;

protected:
// Common to emitter beams and track jam beams
DtFloat32          myAzimuthCenter;
DtFloat32          myAzimuthSweep;
DtBeamFunc         myBeamFunction;
DtU8               myBeamId;
DtU16              myBeamParameterIndex;
```

```
DtFloat32          myElevationCenter;
DtFloat32          myElevationSweep;
DtGlobalObjectDesignator myEmittingSystemId;
DtFloat32          myERP;
DtEntityIdentifier myEntityId;
DtEventId          myEventId;
DtFloat32          myFrequency;
DtFloat32          myFrequencyRange;
DtFloat32          myPRF;
DtFloat32          myPulseWidth;
DtFloat32          mySweepSync;

// For Track Jam Beams
DtU32              myJammingModeSequence;
DtTrackJamDesignatorList myTrackJamTargets;
DtBeamType         myBeamType;
DtBoolean          myHighDensity;
};
```

emptyFomMap.h

```
#if DtHLA

#include "RTI.hh"
#include "hostStructs.h"
#include "hashlist.h"
#include "fomMapper.h"

#include "chooseDefines.h"

// class DtEmptyFomMapper:
//
// An instances of DtEmptyFomMapper is a DtFomMapper that has no
// mappings
// registered between FOM classes and VR-Link classes. Mappings can
// be added
// later.

class DT_DLL_VL DtEmptyFomMapper : public DtFomMapper
{
public:

    // Default constructor. Typically, most initialization takes
    place within
    // init(), which is called by DtExerciseConn's constructor.
    DtEmptyFomMapper();

    // destructor
    virtual ~DtEmptyFomMapper();

    // Initialize the instance, once an exercise connection is avail-
    able.
    // Called by DtExerciseConn constructor.
    virtual void init(DtExerciseConn* conn);

    // Returns the default state decoder factory
    DtStateDecoderFactory* dfltStateDecoderFactory();

    // Returns the default state encoder factory
    DtStateEncoderFactory* dfltStateEncoderFactory();

    // Returns the default interaction decoder factory
    DtInteractionDecoderFactory* dfltInteractionDecoderFactory();

    // Returns the default state interaction factory
    DtInteractionEncoderFactory* dfltInteractionEncoderFactory();

    // Returns the default HlaObject factory
```



```
DtHlaObjectFactory* dfltHlaObjectFactory();

// Returns the default Interaction factory
DtInteractionFactory* dfltInteractionFactory();

public:

    // A Creator function that returns a DtEmptyFomMapper configured
    with
    // no mappings at all.
    static DtFomMapper* create(void* usr = NULL);

protected:

    // copy constructor - not implemented
    DtEmptyFomMapper(const DtEmptyFomMapper& orig);

    // assignment operator - not implemented
    DtEmptyFomMapper& operator=(const DtEmptyFomMapper& orig);

protected:

    // Default factories and other objects
    DtStateDecoderFactory* myDfltDecFact;
    DtStateEncoderFactory* myDfltEncFact;
    DtInteractionDecoderFactory* myDfltInterDecFact;
    DtInteractionEncoderFactory* myDfltInterEncFact;
    DtHlaObjectFactory* myDfltObjFact;
    DtInteractionFactory* myDfltInterFact;
};
#endif
```

encFactory.h

```
#if DtHLA

#include "hStateEncoder.h"
#include "MachineTypes.h"

// class DtStateEncoderFactory:
//
// Instances of DtStateEncoderFactory are used to create encoders
// which
// know how to encode a state repository (or host representation)
// into an HLA state message (or network representation).

class DT_DLL_VL DtStateEncoderFactory
{
public:

    // constructor
    DtStateEncoderFactory(DtExerciseConn* exConn);

    // destructor
    virtual ~DtStateEncoderFactory();

    // copy constructor
    DtStateEncoderFactory(const DtStateEncoderFactory& orig);

    // assignment operator
    DtStateEncoderFactory& operator=(const DtStateEncoderFactory&
orig);

    // Create an encoder for the specified object class.
    // Note: Caller is responsible for deleting.
    // Returns NULL if encoder creation function is not in the
factory.
    virtual DtHlaStateEncoder* createEncoder(RTI::ObjectClassHandle
hand);

    // Add a encoder prototype to the factory. The encoder prototype
    // will be cloned by createEncoder when called for the appro-
priate
    // handle. Caller deletes the encoder, but AFTER the encoder
    // factory no longer needs it. Returns any previous encoder
which was
    // replaced by the add.
    virtual DtHlaStateEncoder* addEncoder(
        RTI::ObjectClassHandle hand,
        DtHlaStateEncoder* encoder);


```

```

// This function is used to add an attribute encoder function
// to several prototype encoders all at once - namely, those
// associated with the object class denoted by hand, as well
// as all of its subclasses. If applyToSubclasses is DtFalse,
// then the function is only registered with the one encoder
// prototype, making it equivalent to
//   encoder(hand)->addEncoder(hand, encoder));
virtual void addAttributeEncoder(
    RTI::ObjectClassHandle classHand,
    RTI::AttributeHandle attrHand,
    DtHlaStateEncoder::DtAttributeEncoder encoder,
    DtBoolean applyToSubclasses = DtTrue);

// Same as above, but identify class by name
virtual void addAttributeEncoder(
    char* className,
    RTI::AttributeHandle attrHand,
    DtHlaStateEncoder::DtAttributeEncoder func,
    DtBoolean applyToSubclasses = DtTrue);

// Same as above, but identify attr by name
virtual void addAttributeEncoder(
    RTI::ObjectClassHandle classHand,
    char* attrName,
    DtHlaStateEncoder::DtAttributeEncoder func,
    DtBoolean applyToSubclasses = DtTrue);

// Same as above, but identify class and attr by name
virtual void addAttributeEncoder(
    char* className,
    char* attrName,
    DtHlaStateEncoder::DtAttributeEncoder func,
    DtBoolean applyToSubclasses = DtTrue);

// This function is used to add an attribute checker function
// to several prototype encoders all at once - namely, those
// associated with the object class denoted by hand, as well
// as all of its subclasses. If applyToSubclasses is DtFalse,
// then the function is only registered with the one encoder
// prototype, making it equivalent to
//   encoder(hand)->addChecker(hand, encoder));
virtual void addAttributeChecker(
    RTI::ObjectClassHandle classHand,
    RTI::AttributeHandle attrHand,
    DtHlaStateEncoder::DtAttributeChecker encoder,
    DtBoolean applyToSubclasses = DtTrue);

// Same as above, but identify class by name
virtual void addAttributeChecker(
    char* className,

```

```
    RTI::AttributeHandle attrHand,
    DtHlaStateEncoder::DtAttributeChecker func,
    DtBoolean applyToSubclasses = DtTrue);

// Same as above, but identify attr by name
virtual void addAttributeChecker(
    RTI::ObjectClassHandle classHand,
    char* attrName,
    DtHlaStateEncoder::DtAttributeChecker func,
    DtBoolean applyToSubclasses = DtTrue);

// Same as above, but identify class and attr by name
virtual void addAttributeChecker(
    char* className,
    char* attrName,
    DtHlaStateEncoder::DtAttributeChecker func,
    DtBoolean applyToSubclasses = DtTrue);

// Returns the encoder prototype associated with handle by a
previous
// call to addEncoder. If defaultToBase is DtTrue, then if there
is
// no encoder registered with hand, rather than returning NULL,
// we search up the class hierarchy, and return the encoder
registered
// with the most specific superclass for which a encoder has
// been registered.
virtual DtHlaStateEncoder* encoder(RTI::ObjectClassHandle hand,
    DtBoolean defaultToBase = DtFalse);

protected:

    // myProtos is an array of DtHlaObject::maxObjectClasses() items.
    DtHlaStateEncoder** myProtos;
    DtExerciseConn* myExConn;
};
#endif
```

entitySR.h

```

#include "hostStructs.h"
#include "mathTypes.h"
#include "artPartList.h"
#include "attPartList.h"
#include "baseEntitySR.h"

// class DtEntityStateRepository:
//
// Instances of DtEntityStateRepository are used to maintain entity
// state. It provides a fat interface which covers various types of
// entities. As a result, individual fields may not apply depending
// on the type of entity.

class DT_DLL_VL DtEntityStateRepository : public
DtBaseEntityStateRepository
{
public:

    // default constructor
    DtEntityStateRepository();

    // destructor
    virtual ~DtEntityStateRepository();

    // copy constructor
    DtEntityStateRepository(const DtEntityStateRepository& orig);

    // assignment operator
    DtEntityStateRepository& operator=(const DtEntityStateRepository&
orig);

    // return an empty object of the same type as this
    virtual DtStateRepository* clone() const;

    // Set/Get the force type.
    virtual void setForceId(DtForceType force);
    virtual DtForceType forceId() const;

    // Set/Get the alternate entity type or guise.
    virtual void setGuise(const DtEntityType& type);
    virtual const DtEntityType& guise() const;

#if DtDISVERS != 3
    // Set/Get whether frozen or not.
    virtual void setFrozen(DtBoolean onOff);
    virtual DtBoolean frozen() const;
#endif
}

```

```
// Set/Get the character set used for the marking.
// Defaults to 1.
virtual void setMarkingCharSet(int charSet);
virtual int markingCharSet() const;

// Set/Get the marking text.
virtual void setMarkingText(const char* text);
virtual const char* markingText() const;

#if DtDISVERS != 3
// Set/Get whether entity's power plant is on or not.
virtual void setPowerPlantOn(DtBoolean onOff);
virtual DtBoolean powerPlantOn() const;
#endif

// Set/Get the damage state.
virtual void setDamageState(DtDamageState s);
virtual DtDamageState damageState() const;

// Set/Get whether engine smoke is on or not.
virtual void setEngineSmokeOn(DtBoolean onOff);
virtual DtBoolean engineSmokeOn() const;

// Set/Get whether flames are present or not.
virtual void setFlamesPresent(DtBoolean onOff);
virtual DtBoolean flamesPresent() const;

// Set/Get whether entity has fuel supplying capability.
virtual void setHasFuelSupplyCap(DtBoolean onOff);
virtual DtBoolean hasFuelSupplyCap() const;

// Set/Get whether entity has recovery capability.
virtual void setHasRecoveryCap(DtBoolean onOff);
virtual DtBoolean hasRecoveryCap() const;

// Set/Get whether entity has repair capability.
virtual void setHasRepairCap(DtBoolean onOff);
virtual DtBoolean hasRepairCap() const;

// Set/Get the entity's hatch state.
virtual void setHatchState(DtHatchState s);
virtual DtHatchState hatchState() const;

// Set/Get whether the entity is immobilized or not.
virtual void setImmobilized(DtBoolean onOff);
virtual DtBoolean immobilized() const;

// Set/Get entity life form state.
virtual void setLifeformState(DtLifeformState s);
```

```
virtual DtLifeformState lifeformState() const;

// Set/Get light state of entity.
virtual void setLightState(DtLightState s);
virtual DtLightState lightState() const;

// Set/Get light state of entity. (Environment entities).
virtual void setDensity(DtDensityType dens);
virtual DtDensityType density() const;

// Set/Get whether entity has ramp deployed or not.
virtual void setRampDeployed(DtBoolean onOff);
virtual DtBoolean rampDeployed() const;

// Set/Get whether a smoke plume is present or not.
virtual void setSmokePlumePresent(DtBoolean onOff);
virtual DtBoolean smokePlumePresent() const;

// Set/Get whether a tent is deployed or not.
virtual void setTentDeployed(DtBoolean onOff);
virtual DtBoolean tentDeployed() const;

// Set/Get trailing effect state.
virtual void setTrailState(DtTrailState s);
virtual DtTrailState trailState() const;

// Set/Get camouflage type.
virtual void setCamouflageType(DtCamouflageType type);
virtual DtCamouflageType camouflageType() const;

// Set/Get whether entity is fire power disabled or not.
virtual void setFirePowerDisabled(DtBoolean onOff);
virtual DtBoolean firePowerDisabled() const;

// Set/Get whether entity is concealed or not.
virtual void setIsConcealed(DtBoolean onOff);
virtual DtBoolean isConcealed() const;

// Set/Get whether entity has after burner on or not.
virtual void setAfterburnerOn(DtBoolean onOff);
virtual DtBoolean afterburnerOn() const;

// Set/Get whether entity has ammunition supply capability or
not.
virtual void setHasAmmunitionSupplyCap(DtBoolean onOff);
virtual DtBoolean hasAmmunitionSupplyCap() const;

// Set/Get whether entity has launcher raised or not.
virtual void setLauncherRaised(DtBoolean onOff);
virtual DtBoolean launcherRaised() const;
```

```
// Set/Get whether launcher flash is present or not.
virtual void setLauncherFlashPresent(DtBoolean onOff);
virtual DtBoolean launcherFlashPresent() const;

// Set/Get primary weapon deployment state.
virtual void setPrimaryWeaponState(DtWeaponState s);
virtual DtWeaponState primaryWeaponState() const;

// Set/Get secondary weapon deployment state.
virtual void setSecondaryWeaponState(DtWeaponState s);
virtual DtWeaponState secondaryWeaponState() const;

// initialize articulated parts
virtual void initArtParts(const DtArtPartSpec* specList);

// get the articulated parts list
virtual const DtArtPartList* artPartList() const;
virtual DtArtPartList* artPartList();

// get the attached parts list
virtual const DtAttachedPartList* attPartList() const;
virtual DtAttachedPartList* attPartList();

#if DtHLA
protected:
#endif

// Set/Get the appearance field.
virtual void setAppearance(DtU32 app);
virtual DtU32 appearance() const;

// Set/Get an appearance attribute.
// attr is the appearance bitfield mask
// val is the hex number to be or'ed in
virtual void setAppearanceAttribute(const DtAppearAttribute attr,
    const DtAppearAttrVal val);
virtual DtAppearAttrVal appearanceAttribute(
    const DtAppearAttribute attr) const;

// Set/Get the capabilities field.
virtual void setCapabilities(DtU32 cap);
virtual DtU32 capabilities() const;

public:

// Print the state data.
virtual void printData() const;

public:
```



```
static DtEntityStateRepository* create();

protected:

    // Cast off const-ness for trivial internal change.
    // Non-virtual since can't override on return type.
    DtEntityStateRepository* self() const;

    // Set the marking text.
    // Non-virtual since meant to be used by constructor.
    void internalSetMarkingText(const char* text);

protected:

    DtArtPartList myArtPartList;
    DtAttachedPartList myAttPartList;

    DtForceType myForceId;
    DtEntityType myGuise;
    DtU32 myAppearance;
    int myMarkingCharSet;
    // myMarkingText is an array of theMaxMarkingSize.
    // String length is theMaxMarkingSize - 1 (doesn't include NULL).
    char* myMarkingText;
    DtU32 myCapabilities;

protected:

    static const int theMaxMarkingSize;

};
```

EntityTypes.h

```
/*
 * We use enumerated types instead of manifest constants
 * so that cross-type assignment mistakes can be caught
 * at compile time.
 * -
 */

#include "packets/disVersion.h"

/*****
 * DtEntityKinds
 *****/

enum DtEntityKind
{
    DtEntityKindOther =0,
    DtPlatform =1,
    DtMunition =2,
    DtLifeForm =3,
    DtEnvironmental =4,
    DtCulturalFeature =5,
    DtSupply =6
};

/*****
 * Domains
 *****/

enum DtPlatformDomain
{
    DtPlatformDomainOther =0,
    DtPlatformDomainLand =1,
    DtPlatformDomainAir =2,
    DtPlatformDomainSurface=3,
    DtPlatformDomainSubsurface =4,
    DtPlatformDomainSpace =5
};

enum DtMunitionDomain
{
    DtMunitionDomainOther =0,
    DtAntiAir =1,
    DtAntiArmor =2,
    DtAntiGuidedMunition =3,
    DtAntiRadar =4,
    DtAntiSatellite =5,
    DtAntiShip =6,
```

```

        DtAntiSubmarine =7,
        DtAntiPersonnel = 8,
#if DtDISVERS == 3
        DtBattlefieldSupport8 =8,
        DtBattlefieldSupport9 =9,
        DtStrategic =9,
        DtMiscellaneous =10,
        DtBomb = 12
#else
        DtBattlefieldSupport =9,
        DtStrategic =10
#endif
};

enum DtLifeFormDomain
{
    DtLifeFormDomainOther =0,
    DtLifeFormDomainLand =1,
    DtLifeFormDomainAir =2,
    DtLifeFormDomainSurface =3,
    DtLifeFormDomainSubsurface =4,
    DtLifeFormDomainSpace =5
};

enum DtEnvironmentalDomain
{
    DtEnvironmentalDomainOther =0,
    DtEnvironmentalDomainLand =1,
    DtEnvironmentalDomainAir =2,
    DtEnvironmentalDomainSurface =3,
    DtEnvironmentalDomainSubsurface =4,
    DtEnvironmentalDomainSpace =5
};

enum DtCulturalFeatureDomain
{
    DtCulturalFeatureDomainOther =0,
    DtCulturalFeatureDomainLand =1,
    DtCulturalFeatureDomainAir =2,
    DtCulturalFeatureDomainSurface =3,
    DtCulturalFeatureDomainSubsurface =4,
    DtCulturalFeatureDomainSpace =5
};

/*****
 * Country codes
 *****/

enum DtCountryCode
{

```

```
    DtCountryCodeOther =0,
    DtFrance = 71,
    DtIraq = 102,
    DtUnionOfSovietSocialistRepublics = 222, /* Hard to keep
current, isn't it? */
    DtUnitedKingdom =224,
    DtUnitedStates =225
};

/*****
 * Platforms
 *****/

enum DtPlatformLand
{
    DtTank = 1,
    DtArmoredFightingVehicle =2,
    DtArmoredUtilityVehicle =3,
    DtSelfPropelledArtillery =4,
    DtTowedArtillery =5,
    DtSmallWheeledUtilityVehicle =6,
    DtLargeWheeledUtilityVehicle =7,
    DtSmallTrackedUtilityVehicle =8,
    DtLargeTrackedUtilityVehicle =9,
#ifdef DtDISVERS == 3
    DtAmphibiousAssaultVehicle =50,
    DtMiscellaneousWeaponSystem =51,
#endif
    DtMortar =10
};

enum DtPlatformAir
{
    DtFighter =1,
    DtAttack =2,
    DtBomber =3,
    DtCargo =4,
    DtAntiSubmarinePatrol =5,
#ifdef DtDISVERS == 3
    DtAttackHelicopter = 6,
    DtUtilityHelicopter =7,
    DtCargoHelicopter =50,
    DtObservationHelicopter =51,
    DtSeaGoingHelicopter =52,
    DtElectronicWarfare = 53,
    DtUnmannedAirVehicle = 54,
    DtSpyPlanes =54,
    DtSpecialOperations =55
#else
    DtAttackHelicopter = 20,
```

```

    DtUtilityHelicopter =21,
    DtSeaGoingHelicopter =22,
    DtElectronicWarfare = 6,
    DtUnmannedAirVehicle = 50,
    DtReconnaissance = 8,
    DtSpecialOperations =25
#endif
};

enum DtPlatformSurface
{
    DtCarrier = 1,
    DtCommandShip = 2,
    DtGuidedMissileCruiser = 3,
    DtGuidedMissileDestroyer = 4,
    DtDestroyer = 5,
    DtGuidedMissileFrigate = 6,
    DtPatrolCraft = 7,
    DtMineCounterMeasureShip = 8,
    DtDockLandingShip = 9,
    DtTankLandingShip = 10,
    DtLandingCraft = 11,
    DtFrigate = 50,
    DtBattleship = 51,
    DtHeavyCruiser = 52,
    DtDestroyerTender = 53,
    DtAmphibiousAssaultShip = 54,
    DtAmphibiousCargoShip = 55,
    DtAmphibiousTransportDock = 56,
    DtAmmunitionShip = 57,
    DtCombatStoresShip = 58
};

#if DtDISVERS == 3
enum DtPlatformSubsurface
{
    DtStrategicMissileSubmarine = 1,
    DtAttackSubmarine = 2,
    DtResearchSubmarine = 3
};
#endif

/*
 * Any Country
 */

enum DtPlatformLandAnySmallWheeledUtilityVehicle
{
    DtCivilianCar = 0
};

```

```
#if DtDISVERS == 3
enum DtPlatformLandAnyLargeWheeledUtilityVehicle
{
    DtTruck = 0,
    DtBus = 14
};
#endif

/*
 * US.
 */

enum DtPlatformLandUSTank
{
#if DtDISVERS == 3
    DtVirtualPrototype = 5,
#endif
    DtM1 = 1
};

enum DtPlatformLandUSArmoredFightingVehicle
{
#if DtDISVERS == 3
    DtLOSAT = 3,
    DtM2 = 3,
    DtM3 = 4,
    DtM113 = 5
#else
    DtM2 = 1,
    DtM3 = 1,
    DtM113 = 3
#endif
};

#if DtDISVERS == 3
enum DtLOSATSpecific
{
    DtSpecificLOSAT = 1
};
#endif

enum DtPlatformLandUSArmoredUtilityVehicle
{
    DtM88 = 1
};

enum DtPlatformLandUSSelfPropelledArtillery
{
    DtM109 = 3
}
```

```
};

#if DtDISVERS == 3
enum DtPlatformLandUSSelfTowedArtillery
{
    DtPatriotLauncher = 5,
    DtThAADLauncher = 6
};
#endif

enum DtPlatformLandUSSmallWheeledUtilityVehicle
{
    DtUSCivilianCar = 0,
#if DtDISVERS == 3
    DtM577 = 2,
#endif
    DtHMMWV = 1
};

#if DtDISVERS == 3
enum DtPlatformLandUSLargeWheeledUtilityVehicle
{
    DtM977HEMTT = 2,
    DtM978HEMTT = 3,
    DtPatriotRadarTrailer = 12
};
#endif

enum DtPlatformLandUSMiscellaneousWeaponSystem
{
    Dt20mmVulcanAirDefenseSystem = 3 /* Alias FAADS */
};
#endif

enum DtPlatformAirUSFighter{
    DtF117 = 1,
    DtF16 = 3,
#if DtDISVERS == 3
    DtF15 = 7,
    DtF15E = 7,
    DtF18 = 15
#else
    DtF15 = 5,
    DtF15E = 5,
    DtF18 = 9
#endif
};

enum DtF16
{
    DtBasicF16 = 0,
```

```
    DtF16C = 3
};

enum DtPlatformAirUSAttack
{
    DtA10 = 4
};

enum DtPlatformAirUSCargo
{
    DtC130 = 1,
#if DtDISVERS == 3
    DtKC130 = 7,
    DtV22 = 8
#else
    DtKC130 = 1
#endif
};

enum DtPlatformAirUSAttackHelicopter
{
    DtAH64 = 1,
    DtAH1W = 2,
    DtRAH66 = 3
};

enum DtAH1WSpecific
{
#if DtDISVERS == 3
    DtSpecificAH1W = 3
#else
    DtSpecificAH1W = 10
#endif
};

enum DtPlatformAirUSObservationHelicopter
{
    DtOH58 = 1
};

enum DtPlatformAirUSUnmannedAirVehicle
{
    DtUAV = 0
};

#if DtDISVERS == 3
enum DtPlatformAirUSElectronicWarfare
{
    DtE3 = 4,
    DtJSTARS = 6
}
```



```
};

enum DtPlatformAirUSSpyPlanes
{
    DtU2 = 3
};
#endif

enum DtPlatformAirUSSpecialOperations
{
    DtMH60 = 1,
#if DtDISVERS == 3
    DtBiplane = 5,
#endif
    DtMH53 = 3
};

enum DtPlatformSurfaceUSCarrier
{
    DtCV59 = 4
};

enum DtCV59Specific
{
    DtSpecificCV59 = 1
};

enum DtPlatformSurfaceUSGuidedMissileCruiser
{
#if DtDISVERS == 3
    DtCG47AegisCruiser
#else
    DtTiconderogaClass
#endif
};

enum DtAegisCruiserSpecific
{
    DtTiconderoga = 1,
    DtYorktown = 2,
    DtVincennes = 3,
    DtValleyForge = 4,
    DtThomasSGates = 5
};

enum DtPlatformSurfaceUSGuidedMissileFrigate
{
    DtFFG7 = 1
};
```

```
enum DtFFG7Specific
{
    DtSpecificFFG7 = 1
};

enum DtPlatformSurfaceUSAmphibiousAssaultShip
{
    #if DtDISVERS == 3
        DtLHA = 1
    #else
        DtTarawaClass = 1
    #endif
};

enum DtLHASpecific
{
    DtTarawa = 1,
    DtSaipan = 2,
    DtBelleauWood = 3,
    DtNassau = 4,
    DtPeleliu = 5
};

enum DtPlatformSubsurfaceUSStrategicMissileSubmarine
{
    DtSBBNOhio = 1
};

/*
 * France.
 */

enum DtPlatformAirFranceUtilityHelicopter
{
    #if DtDISVERS == 3
        DtGazelle = 3
    #else
        DtGazelle = 6
    #endif
};

/*
 * UK
 */

enum DtPlatformLandUKTank
{
    #if DtDISVERS == 3
        DtChallengerMBT = 3
    #else
```

```
        DtChallengerMBT      = 2
#endif
};

/*
 * USSR.
 */

enum DtPlatformLandUSSRTank
{
    DtT72 = 2,
#if DtDISVERS == 3
    DtT64 = 4
#else
    DtT64 = 3
#endif
};

enum DtT72Specific
{
    DtBasicT72= 1
};

enum DtPlatformLandUSSRArmoredFightingVehicle
{
    DtBMP1 = 1,
    DtBMP2 = 2,
    DtBRDM1 = 3,
#if DtDISVERS == 3
    DtMCV80 = 3,
    DtACRV2 = 10,
#endif
    DtMTLB = 7
};

#if DtDISVERS == 3
enum DtMTLBSpecific
{
    DtMTLBUTrackedVehicle = 17
};
#endif

enum DtPlatformLandUSSRSelfPropelledArtillery
{
    DtSS1ScudMissileLauncher = 10,
    DtZSU23_4 = 18,
    DtSA6Launcher = 19,
    DtSAMSite = 19,
#if DtDISVERS == 3
    DtZSU30 = 23,
```

```
    DtSA10FlapLidRadar = 38,
#else
    DtSA10FlapLidRadar = 42,
#endif
    DtSA8LauncherRadar = 41
};

enum DtSA10Specific
{
    DtSpecificSA10 = 1
};

enum DtSA8Specific
{
    DtSpecificSA8 = 1
};

enum DtPlatformLandUSSRSelfTowedArtillery
{
    DtD20 = 3
};

#if DtDISVERS == 3
enum DtPlatformLandUSSRSmallWheeledUtilityVehicle
{
    DtSA2FanSongRadarTrailer = 8,
    DtSA3LowBlowRadarTrailer = 9
};
#endif

enum DtPlatformLandUSSRLargeWheeledUtilityVehicle
{
    DtZIL157 = 6,
    DtGAZ66 = 8,
    DtUral375 = 9
};

#if DtDISVERS == 3
enum DtPlatformLandUSSRSmallTrackedUtilityVehicle
{
    DtSA6StraightFlushRadar = 10
};

enum DtPlatformLandUSSRLargeTrackedUtilityVehicle
{
    DtATTHeavyTrackedVehicle = 5
};
#endif

enum DtPlatformAirUSSRAttackHelicopter
```

```

{
    DtMI28 = 1,
    DtMI24 = 2
};

enum DtPlatformAirUSSRFighter
{
    DtMiG29 = 2,
    DtMiG23 = 5
};

enum DtPlatformAirUSSRAttack
{
    DtSU25 = 8
};

enum DtPlatformAirUSSRBomber
{
    DtTU26 = 3,
    DtTU95 = 5
};

#if DtDISVERS == 3
enum DtPlatformAirUSSRElectronicWarfare
{
    DtEA3B = 2
};

enum DtEA3BSpecific
{
    DtSpecificEA3B = 2
};

enum DtPlatformSubsurfaceUSSRAttackSubmarine
{
    DtSSTangoClass = 20
};
#endif

/*****
 * Munitions.
 *****/

enum DtMunitionAntiAir
{
    DtMunitionGuided = 1,
    DtMunitionUnguided = 2
};

enum DtMunitionAntiArmor

```

```
{
    DtAntiArmorGuided = 1,
    DtAntiArmorUnguided = 2
};

enum DtMunitionAntiRadar
{
    DtAntiRadarGuided = 1,
    DtAntiRadarUnguided = 2
};

enum DtMunitionAntiShip
{
    DtAntiShipGuided = 1,
    DtAntiShipUnguided = 2
};

enum DtMunitionAntiSubmarine
{
    DtAntiSubmarineGuided = 1,
    DtAntiSubmarineUnguided = 2
};

enum DtMunitionAntiPersonnel
{
    DtAntiPersonnelGuided = 1,
    DtAntiPersonnelUnguided = 2
};

enum DtMunitionBattlefieldSupport
{
    DtBattlefieldSupportGuided = 1,
    DtBattlefieldSupportUnguided = 2
};

#if DtDISVERS == 3
enum DtMunitionBomb
{
    DtBombGuided = 1,
    DtBombUnguided = 2
};
#endif

/*
 * US.
 */

enum DtMunitionUSAntiAirGuided
{
    DtSidewinder = 1,
```

```
DtAMRAAM = 2,  
DtHawk = 6,  
DtPhoenix = 8,  
DtRoland = 11,  
DtSeaSparrow = 12,  
DtSparrow = 13,  
DtStinger = 15,  
#if DtDISVERS == 3  
    DtThAAD = 20,  
#endif  
    DtPatriot = 16  
};  
  
#if DtDISVERS == 3  
enum DtSidewinderSpecific  
{  
    DtAIM9 = 0,  
    DtAIM9M = 1  
};  
  
enum DtMunitionUSAntiAirUnguided  
{  
    DtM60Cannon = 1,  
    Dt20mmAntiAirUnguided = 1  
};  
#endif  
  
enum DtMunitionUSAntiArmorGuided  
{  
    DtBGM71TOW = 1,  
    DtHellfire = 3,  
#if DtDISVERS == 3  
    DtLOSATMunition = 7,  
    DtFOGM = 8,  
#endif  
    DtMaverick = 4  
};  
  
#if DtDISVERS == 3  
enum DtMaverickSpecific  
{  
    DtAGM65 = 0,  
    DtAGM65D = 1  
};  
#endif  
  
enum DtMunitionUSAntiArmorUnguided  
{  
#if DtDISVERS == 3  
    Dt25mm = 1,
```

```
        Dt105m = 3,
        Dt120mm = 4
#else
        Dt25mm = 3,
        Dt105m = 10,
        Dt120mm = 13
#endif
};

enum DtMunitionUSAntiRadarGuided
{
    DtAGM88HARM = 1
};

#if DtDISVERS == 3
enum DtMunitionUSAntiShipGuided
{
    DtHarpoon84A = 1,
    DtHarpoon84B = 2,
    DtHarpoon84C = 3,
    DtHarpoon84D = 4,
    DtMk48 = 6
};
#endif

enum DtMunitionUSAntiSubmarineGuided
{
    DtMk48AntiSubmarine = 2
};

enum DtMunitionUSAntiPersonnelUnguided
{
    Dt7_62mm = 2
};

enum DtMunitionUSBattlefieldSupportUnguided
{
    DtBattlefield20mm = 1,
    DtBattlefield30mm = 3,
    DtBattlefield105mm = 10,
#if DtDISVERS == 3
    Dt2_75inRocket = 19,
    DtUnguidedM60Cannon = 20,
#endif
    DtBattlefield120mm = 11
};

enum DtMunitionUSBattlefieldSupportGuided
{
    DtBGM109Tomahawk = 6
};
```



```
};

#if DtDISVERS == 3
enum DtTomahawkSpecific
{
    DtBasicTomahawk = 0,
    DtTomahawkBooster = 1,
    DtTomahawkWings = 2
};

enum DtMunitionUSBombUnguided
{
    DtCBU1 = 1,
    DtCBU2 = 2,
    DtCBU3 = 3,
    DtCBU4 = 4,
    DtCBU5 = 5,
    DtCBU6 = 6,
    DtCBU7 = 7,
    DtMk82 = 9,
    DtMk84 = 11
};
#endif

/*
 * USSR
 */

enum DtMunitionUSSRAntiAirGuided
{
    DtAA3Anan = 3,
    DtAA9Amos = 9,
#if DtDISVERS == 3
    DtSA2Guideline = 13,
    DtSA3Goa = 14,
    DtSA4Ganef = 15,
    DtSA6Gainful = 17,
    DtSA7Grail = 18,
    DtSA8Gecko = 19,
    DtSA9Gaskin = 20,
    DtSA10Grumble = 21
#else
    DtSA2Guideline = 14,
    DtSA3Goa = 15,
    DtSA4Ganef = 16,
    DtSA6Gainful = 18,
    DtSA7Grail = 19,
    DtSA8Gecko = 20,
    DtSA9Gaskin = 21,
    DtSA10Grumble = 22
#endif
};
```

```
#endif
};

#if DtDISVERS == 3
enum DtMunitionUSSRAntiAirUnguided
{
    DtZSU30AAA = 2
};

enum DtMunitionUSSRAntiArmorUnguided
{
    Dt30mm = 3
};

enum DtMunitionUSSRBattlefieldSupport
{
    DtSS1 = 10/* Alias SCUD */
};

enum DtMunitionUSSRBattlefieldSupportUnguided
{
    DtBattlefield25mmUSSR = 3,
    DtBattlefield120mUSSR = 15
};
#endif

enum DtMunitionUSSRAntiPersonnelUnguided
{
    Dt7_62mmUSSR = 2
};

#if DtDISVERS == 3
enum DtMunitionUSSRAntiSubmarineGuided
{
    DtAntiSubmarineE53 = 1
};

enum DtE53Specific
{
    DtET80A = 1,
    DtE5383 = 2,
    DtE5371 = 3
};
#endif

/*
 * UK
 */

enum DtMunitionUKAntiPersonnelUnguided
```

```

{
    Dt7_62mmUK = 1
};

enum DtMunitionUKBattlefieldSupportUnguided
{
    DtBattlefield105mmUK = 3
};

/*****
 * Life Forms.
 *****/

enum DtLifeFormLand
{
    DtDismountedInfantry = 1
};

enum DtNumberOfIndividualLifeForms
{
    DtFireTeam = 101,
    DtSquad = 102,
    DtPlatoon = 103,
    DtCompany = 104,
    DtBattalion = 105,
    DtRegiment = 106,
    DtDivision = 107,
    DtCorps = 108,
    DtArmy = 109
};

/*****
 * Environmental
 *****/

enum DtEnvironmentalAir
{
    DtEnvironmentalDomainAirOther = 0,
    DtSmoke = 1,
    DtFog = 2,
    DtCloud = 3,
    DtCloudWithRainFalling = 4,
    DtCloudWithSnowFalling = 5
};

/*****
 * Cultural Feature
 *****/

enum DtCulturalFeatureLand

```

```
{
    DtBridgeConcreteTwoLane = 1,
    DtBuildingOneStory = 7,
    DtBuildingTwoStory = 8,
    DtBuildingThreeStory = 9,
    DtBuildingFourStory = 10,
    DtDragonsTeeth = 20
};

/*****
 * Supply Types
 *****/

enum DtSupplyCategories
{
    DtSupplyOther = 0,
    DtSupplyFuels = 1,
    DtSupplyOils = 2,
    DtSupplyLubricants = 3,
    DtSupplyFood = 4,
    DtSupplySpareParts = 5,
    DtSupplyPersonnel = 6
};

#if DtDISVERS == 3
enum DtSupplyFuels
{
    DtSupplyFuelOther = 0,
    DtGasoline = 1,
    DtDieselFuel = 2,
    DtJP4 = 3,
    DtFuelOil = 4
};
#endif

/*****
 * Used by both Environmental and Cultural features
 *****/

enum DtSizeQualifier
{
    DtOther = 0,
    DtVerySmall = 20,
    DtSmall = 40,
    DtMedium = 60,
    DtLarge = 80,
    DtVeryLarge = 100/* You want fries with that? */
};

/*****/
```

```

* Some useful macros
*****/

#define DtSameEntityKind(a,b) \
    ((a)->DtEntityKind == (b)->DtEntityKind && \
    (a)->DtDomain == (b)->DtDomain && \
    (a)->DtCountry == (b)->DtCountry && \
    (a)->DtCategory == (b)->DtCategory && \
    (a)->DtSubCategory == (b)->DtSubCategory && \
    (a)->DtSpecific == (b)->DtSpecific && \
    (a)->DtExtra == (b)->DtExtra)

#define DtIsLandPlatform(a) \
    ((a)->DtEntityKind == DtPlatform && \
    (a)->DtDomain == DtPlatformDomainLand)

#define DtIsLandLifeForm(a) \
    ((a)->DtEntityKind == DtLifeForm && \
    (a)->DtDomain == DtLifeFormDomainLand)

#define DtIsAirPlatform(a) \
    ((a)->DtEntityKind == DtPlatform && \
    (a)->DtDomain == DtPlatformDomainAir)

#define DtIsTank(a) \
    (DtIsLandPlatform(a) && (a)->DtCategory == DtTank)

#if DtDISVERS == 3
#define DtIsRotaryWing(a) \
    (DtIsAirPlatform(a) && \
    (a)->DtCategory > DtAntiSubmarinePatrol && \
    (a)->DtCategory < DtElectronicWarfare)

#define DtIsFixedWing(a) \
    (DtIsAirPlatform(a) && \
    ((a)->DtCategory < DtAttackHelicopter || \
    (a)->DtCategory > DtSeaGoingHelicopter))

#else

#define DtIsRotaryWing(a) \
    (DtIsAirPlatform(a) && \
    (a)->DtCategory >= 20 && \
    (a)->DtCategory < 30)

#define DtIsFixedWing(a) \
    (DtIsAirPlatform(a) && \
    (!DtIsRotaryWing(a)))

#endif

```

```
#if DtDISVERS == 3
#define antiPersonell 1
#else
#define antiPersonell 8
#endif

#define DtEntityIsAntiPersonnel(a) \
(((a)->DtEntityKind == DtMunition) && ((a)->DtDomain == antiPer-
sonell))
```

enum2string.h

```
#include "disEnums.h"

const DT_DLL_VL char *DtEnumString(DtPduKind kind);
const DT_DLL_VL char *DtAddEnumString(DtPduKind kind,
                                       const char *str, int warn = 0);

#define DtADD_ENUM_STRING(k) DtAddEnumString(k, #k + 2)
```

envDec.h

```
#if DtHLA

#include "baseEntDec.h"

// class DtEnvironmentEntityDecoder:
//
// Instances of DtEnvironmentEntityDecoder are used to decode an
// environmental entity state message into a state repository in an
// HLA
// (as opposed to DIS) environment.

class DT_DLL_VL DtEnvironmentEntityDecoder : public DtBaseEntityDe-
coder
{
public:

    // constructor
    DtEnvironmentEntityDecoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtEnvironmentEntityDecoder();

    // copy constructor
    DtEnvironmentEntityDecoder(const DtEnvironmentEntityDecoder&
orig);

    // assignment operator
    DtEnvironmentEntityDecoder& operator=(
        const DtEnvironmentEntityDecoder& orig);

public:

    // Create a DtEnvironmentEntityDecoder.
    // User is responsible for deletion.
    static DtHlaStateDecoder* create(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

protected:

    // Add decoding functions for the attributes.
    // Note: Non-virtual since meant to be called from constructors
to
    //      add decoders.
    void addMyDecoders();
}
```



```
    // static decode<AttributeName> member functions.
    DtDECLARE_ENTITY_ATTR_DECODER(Density);
};
// These macros are used to define simple attribute decoder func-
// tions
// (members of a derived DtHlaStateDecoder that decode
// individual attributes). The simple decoding is just to
// convert the value pulled out of the ahvps to a host type using
// the cast expression castExpr, and pass it to the mutator func-
// tion.
#define DtDEFINE_SIMPLE_ENVENT_ATTR_DECODER_WITH_CAST( \
    attrName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_ATTR_DECODER_WITH_CAST( \
        DtEnvironmentEntityDecoder, DtEntityStateRepository, \
        attrName, netType, mutator, castExpr)

#define DtDEFINE_SIMPLE_ENVENT_ATTR_DECODER( \
    attrName, netType, inspector) \
    DtDEFINE_SIMPLE_ENVENT_ATTR_DECODER_WITH_CAST( \
        attrName, netType, inspector, (netType))

#endif
```

envEnc.h

```
#if DtHLA

#include "baseEntEnc.h"
#include "entitySR.h"

// class DtEnvironmentEntityEncoder:
//
// Instances of DtEnvironmentEntityEncoder are used to encode an
// environment entity's entity's state repository into a state
// message
// in an HLA environment.

class DT_DLL_VL DtEnvironmentEntityEncoder : public DtBaseEntityEncoder
{
public:

    // constructor
    DtEnvironmentEntityEncoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtEnvironmentEntityEncoder();

    // copy constructor
    DtEnvironmentEntityEncoder(const DtEnvironmentEntityEncoder&
orig);

    // assignment operator
    DtEnvironmentEntityEncoder& operator=(
        const DtEnvironmentEntityEncoder& orig);

public:

    // Create a DtEnvironmentEntityEncoder.
    // Caller is responsible for deletion.
    static DtHlaStateEncoder* create(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

protected:

    // static need<Attribute> member functions.
    DtDECLARE_ENTITY_ATTR_CHECKER(Density);

    // static decode<Attribute> member functions.
```

```
DtDECLARE_ENTITY_ATTR_ENCODER(Density);
};
// This macro is used to define simple attribute encoder functions
// (members of a derived DtHlaStateEncoder that encode
// individual attributes). The simple encoding is just to implicitly
// convert the value returned by the inspector to a netType,
// and add it to the ahvps.
#define DtDEFINE_SIMPLE_ENVENT_ATTR_ENCODER( \
    attrName, netType, inspector) \
    DtDEFINE_SIMPLE_ATTR_ENCODER(DtEnvironmentEntityEncoder, \
        DtEntityStateRepository, \
        attrName, netType, inspector)

// This macro is used to define simple attribute checker functions
// (members of a derived DtHlaStateEncoder that checker whether
// a particular attribute is out of date. The simple implementation
// is to just compare the values returned by the two StateReps'
// inspector functions.
#define DtDEFINE_SIMPLE_ENVENT_ATTR_CHECKER( \
    attrName, inspector) \
    DtDEFINE_SIMPLE_ATTR_CHECKER(DtEnvironmentEntityEncoder, \
        DtEntityStateRepository, \
        attrName, inspector)

#endif
```

enviroTypeRec.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "hostStructs.h"

// class DtEnvironmentTypeRecord:
//
// Instances of DtEnvironmentTypeRecord are used by the
// DtGriddedDataPdu for specifying environmental type.

class DT_DLL_VL DtEnvironmentTypeRecord : public DtEntityType
{
public:

    // default constructor
    DtEnvironmentTypeRecord();

    // destructor
    virtual ~DtEnvironmentTypeRecord();

    // copy constructor
    DtEnvironmentTypeRecord(const DtEnvironmentTypeRecord& orig);

    // assignment operators
    DtEnvironmentTypeRecord& operator=(const DtEnvironmentTypeRecord&
orig);
    DtEnvironmentTypeRecord& operator=(const DtNetEntityType& orig);

    // Set/Get the environmental class
    virtual void setClassOfEnv(int cls);
    virtual int classOfEnv() const;

protected:

    // Get/Set country. This field is used for the environmental
    class.
    virtual void setCountry(int country);
    virtual int country() const;

};
#endif
#endif
```

envProcPdu.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "hostStructs.h"
#include "pdu.h"
#include "exerciseConn.h"

#include "packets/envProcP.h"

// class DtEnvironmentalProcessPdu:
//
// Instances of DtEnvironmentalProcessPdu are used to send informa-
// tion about
// environmental effects and processes.

// DtEnvironmentalProcessPdu callback function signature.
typedef void (*DtEnvironmentalProcessPduCb)(
    DtEnvironmentalProcessPdu* pdu, void* usr);

class DT_DLL_VL DtEnvironmentalProcessPdu : public DtPdu
{
public:
    // Constructors
    DtEnvironmentalProcessPdu(DtBufferPtr buffer =
DtUSE_INTERNAL_BUFFER);
    DtEnvironmentalProcessPdu(const DtNetEnvironmentalProcessPdu*
initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtEnvironmentalProcessPdu(const DtEnvironmentalProcessPdu& orig);

    // destructor
    virtual ~DtEnvironmentalProcessPdu();

    // Assignment operator
    DtEnvironmentalProcessPdu& operator=(const DtEnvironmentalPro-
cessPdu& orig);

    // Set/Get originating environmental process ID
    virtual void setEnvironmentalProcessID(const DtEntityIdentifier&
entity);
    virtual const DtEntityIdentifier& environmentalProcessID() const;

    // Set/Get environment type
    virtual void setEnvironmentType(const DtEntityType& type);

```

```
virtual const DtEntityType& environmentType() const;

// Set/Get the model type
virtual void setModelType(DtEnvironmentalModelType value);
virtual DtEnvironmentalModelType modelType() const;

// Set/Get the environmental status (as an 8-bit value)
virtual void setEnvironmentalStatus(DtU8 value);
virtual DtU8 environmentalStatus() const;

// Set/Get an environmental status bit
virtual void setEnvironmentalStatusBit(
    DtEnvironmentalStatusBit bit, DtBoolean value);
virtual DtBoolean environmentalStatusBit(
    DtEnvironmentalStatusBit bit) const;

// Set/Get number of environment records
virtual DtBoolean setNumberOfEnvironmentalRecords(DtU16 newNum-
berOfRecords);
virtual DtU16 numberOfEnvironmentalRecords() const;

// Set/Get sequence number
virtual void setSequenceNumber(DtU16 value);
virtual DtU16 sequenceNumber() const;

// Set the size of an environmental record (in bits). Returns
DtFalse if
// the size is invalid.
virtual DtBoolean setRecordBitSize(int recordIndex, int newRe-
cordBitSize);

// Get a pointer to an environmental record. Returns DtFalse if
the record
// index is out of range.
virtual DtNetEnvironmentalRecord* recordPointer(int recordIndex);

// Print the data contained by this PDU
virtual void printData() const;

public:

// Create a DtEnvironmentalProcessPdu.
// Caller is responsible for deletion.
static DtPdu* create(const DtNetPacket* initial,
    DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

// Add (register)/Remove (deregister) function to be called when
// pdu is received.
static void addCallback(DtExerciseConn* conn,
    DtEnvironmentalProcessPduCb cb, void* usr);
```

```
static void removeCallback(DtExerciseConn* conn,
    DtEnvironmentalProcessPduCb cb, void* usr);
```

protected:

```
// Returns the internal PDU kind
virtual DtPduKind internalGetPduKind() const;
```

```
// Casts the netPdu element to our network struct
// Non-virtual since you can't override based on return type
DtNetEnvironmentalProcessPdu* netPduStruct() const;
```

```
// Casts the 'this' pointer for changing elements in a const
function
// Non virtual since you can't override based on return type
DtEnvironmentalProcessPdu* self() const;
```

```
// Returns a pointer to the first byte of variable length data
virtual char* var() const;
```

```
// Returns an offset in bytes from the beginning of the variable
length
// data region to the beginning of a particular record.
virtual int recordOffset(int recordIndex) const;
```

protected:

```
// Size of PDU with zero data bytes.
static int theStaticSize;
```

protected:

```
// Temporary objects for reference returns
DtEntityIdentifier myEntityID;
DtEntityType myEntityType;
```

```
};
#endif
#endif
```

esPdu.h

```
#if !DtHLA

#include "hostStructs.h"
#include "pdu.h"
#include "packets/entityStateP.h"
#include "artPartList.h"
#include "mathTypes.h"
#include "disEnums.h"
#include "stateMsg.h"

#include "chooseDefines.h"

#define DtEntityStateStaticSize 144
#define DtINCREMENT -1

#define DtNO_SUCH_ART_PART 0
#define DtART_PART_SUCCESS 1

class DT_DLL_VL DtEntityStatePdu;
typedef void (*DtEntityStatePduCb)(DtEntityStatePdu *pdu, void
*usr);

class DT_DLL_VL DtEntityStatePdu : public DtStateMsg
{
public:
/***** STANDARD CONSTRUCTORS *****/

    DtEntityStatePdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtEntityStatePdu(const DtNetEntityStatePdu *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    static DtPdu *create(const DtNetPacket *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    static void addCallback(DtExerciseConn *conn,
        DtEntityStatePduCb cb, void *usr);
    static void removeCallback(DtExerciseConn *conn,
        DtEntityStatePduCb cb, void *usr);

/***** ADDITIONAL CONSTRUCTORS *****/
    DtEntityStatePdu(const DtEntityIdentifier &id,
        const DtEntityType &type,
        const DtEntityType &guise,
        DtDeadReckonTypes algorithm,
        DtForceType forceId,
        const DtArtPartSpec *specList,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtEntityStatePdu(const DtEntityIdentifier &id,
```



```

        const DtEntityType &type,
        const DtEntityType &guise,
        DtDeadReckonTypes algorithm,
        DtForceType forceId,
        const DtArtPartSpec *artList,
        int *stationList,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
DtEntityStatePdu(const DtArtPartSpec *specList,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

/***** INSPECTOR FUNCTIONS *****/
const DtEntityIdentifier &id() const;
DtForceType          forceId() const;
int                  numArtParams() const;
int                  numAttachedParams() const;
int                  numArtPlusAttachedParams() const;
const DtEntityType   &entityType() const;
const DtEntityType   &guise() const;
DtConstVector        velocity() const;
DtConstVector        location() const;
const DtTaitBryan    &orientation() const;
DtU32                appearance() const;
DtAppearAttrVal
appearanceAttribute(const DtAppearAttribute &attr) const;
DtDeadReckonTypes    algorithm() const;
DtConstVector        acceleration() const;
DtConstVector        rotationalVelocity() const;
int                  markingCharSet() const;
const char           *markingText() const;
DtU32                capabilities() const;
const DtArtPartSpec  *artPartSpecs() const;

// returns a DtArtPartList containing the parts described in
this PDU.
const DtArtPartList& artPartList() const;

int      *attachedPartStations() const;
int      getArtParamVal(int paramIndex, DtFloat32 *value) const;
int      getArtParamVal(int partType, int param, DtFloat32
*value) const;
int      getArtParamChange(int paramIndex, int *change) const;
int      getArtParamChange(int partType, int param, int *change)
const;
int      attachedPartChange(int station) const;
int      getAttachedType(int station, DtEntityType *type) const;

/***** MUTATOR FUNCTIONS *****/

void      setId(const DtEntityIdentifier &id);
void      setForceId(DtForceType force);

```

```
void    setEntityType(const DtEntityType &type);
void    setGuise(const DtEntityType &type);
void    setVelocity(DtConstVector Velocity);
void    setLocation(DtConstVector Location);
void    setOrientation(const DtTaitBryan &Orient);
void    setAppearance(DtU32 app);
void    setAppearanceAttribute(const DtAppearAttribute attr,
    const DtAppearAttrVal val);
void    setAlgorithm(DtDeadReckonTypes alg);
void    setAcceleration(DtConstVector Acceleration);
void    setRotationalVelocity(DtConstVector BodyRates);
void    setMarkingText(const char *text);
void    setMarkingCharSet(int charSet);
void    setCapabilities(DtU32 cap);
int      setArtParamVal(int partType, int param, DtFloat32 value,
    int change = DtINCREMENT);
int      setArtParamVal(int paramIndex, DtFloat32 value,
    int change = DtINCREMENT);
void     initArtAndAttachedParts(const DtArtPartSpec *artList,
    const int* stationList);
void     initArtAndAttachedParts(const DtArtPartList &apList,
    const int* stationList);
void     initArtParts(const DtArtPartList &apList);
int      setAttachedPartChange(int station, int change);
int      setAttachedPartType(int station, const DtEntityType
&type);

    void printData() const;
protected:
    DtPduKind internalGetPduKind() const {return DtEntityStatePdu-
Kind;}
    void      setNumArtPlusAttachedParams(int num);
    DtNetArtPartRecord *artPartRecord(int partType, int param)
const;
    DtNetArtPartRecord *attachedPartRecord(int station) const;
    void      setArtParamVal(DtNetArtPartRecord *part, DtFloat32
value,
        int change);
    float      getArtParamVal(DtNetArtPartRecord *part) const;
    DtEntityIdentifier myId;
    DtEntityType type, theGuise;
    DtVector vel, loc, acc;
    DtTaitBryan ori;
    DtBodyRates rvel;
    char markingBuff[DtMaxEntityMarkingLength+1];

    // Does not store state.
    DtArtPartList myArtPartList;
};
#endif
```

esUpdatePdu.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "hostStructs.h"
#include "pdu.h"
#include "exerciseConn.h"
#include "artPartList.h"
#include "attPartList.h"

#include "packets/esUpdateP.h"

// class DtEntityStateUpdatePdu:
//
// Instances of DtEntityStateUpdatePdu are used to send non-static
// information
// about a particular entity.

// DtEntityStateUpdatePdu callback function signature.
typedef void (*DtEntityStateUpdatePduCb)(
    DtEntityStateUpdatePdu* pdu, void* usr);

class DT_DLL_VL DtEntityStateUpdatePdu : public DtPdu
{
public:
    // Constructors
    DtEntityStateUpdatePdu(DtBufferPtr buffer =
DtUSE_INTERNAL_BUFFER);
    DtEntityStateUpdatePdu(const DtNetEntityStateUpdatePdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtEntityStateUpdatePdu(const DtEntityStateUpdatePdu& orig);

    // destructor
    virtual ~DtEntityStateUpdatePdu();

    // Create a DtEntityStateUpdatePdu.
    // Caller is responsible for deletion.
    static DtPdu* create(const DtNetPacket* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // Add (register)/Remove (deregister) function to be called when
    // pdu is received.
    static void addCallback(DtExerciseConn* conn,
        DtEntityStateUpdatePduCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,

```

```
DtEntityStateUpdatePduCb cb, void* usr);
// Assignment operator
DtEntityStateUpdatePdu& operator=(const DtEntityStateUpdatePdu&
orig);

// Set/Get entity ID
virtual void setEntityID(const DtEntityIdentifier& id);
virtual const DtEntityIdentifier& entityID() const;

// Get the number of articulation parameter records (this field
is
// automatically set by the PDU when the articulated part list or
attached
// part list is set)
virtual DtU8 numRecords() const;

// Set/Get velocity
virtual void setVelocity(DtConstVector velocity);
virtual DtConstVector velocity() const;

// Set/Get location
virtual void setLocation(DtConstVector coords);
virtual DtConstVector location() const;

// Set/Get orientation
virtual void setOrientation(const DtTaitBryan& orientation);
virtual const DtTaitBryan& orientation() const;

// Set/Get appearance (as a 32 bit integer)
virtual void setAppearance(DtU32 app);
virtual DtU32 appearance() const;

// Set/Get appearance by attribute
virtual void setAppearanceAttribute(
    const DtAppearAttribute attr, const DtAppearAttrVal val);
virtual DtAppearAttrVal appearanceAttribute(
    const DtAppearAttribute attr) const;

// The articulated parts and attached parts information are both
optional.
// Either or both can be set.

// Set/Get articulated parts parameters.
// Parameters are passed in the form of an articulated parts
list.
// Returns DtFalse if the operation could not be performed.
virtual DtBoolean setArtPartParameters(const DtArtPartList&
partList);
virtual DtBoolean getArtPartParameters(DtArtPartList& partList)
const;
```

```
// Set/Get attached parts parameters.  
// Parameters are passed in the form of an attached parts list.  
// Returns DtFalse if the operation could not be performed.  
virtual DtBoolean setAttPartParameters(const DtAttachedPartList&  
partList);  
virtual DtBoolean getAttPartParameters(DtAttachedPartList&  
partList) const;  
  
// Print the data contained by this PDU  
virtual void printData() const;
```

protected:

```
// Returns the internal PDU kind  
virtual DtPduKind internalGetPduKind() const;  
  
// Casts the netPdu element to our network struct  
virtual DtNetEntityStateUpdatePdu* netPduStruct() const;  
  
// Casts the 'this' pointer for changing elements in a const  
function  
// Non-virtual since you can't override based on return type  
DtEntityStateUpdatePdu* self() const;  
  
// Returns a pointer to the first byte of variable length data  
virtual char* var() const;  
  
// Set number of records  
virtual void setNumRecords(DtU8 value);  
  
// Returns the offset (in bytes) of a field in the variable data  
region.  
virtual int artParmOffset() const;  
virtual int attParmOffset() const;
```

protected:

```
// Size of PDU with zero parameter records.  
static int theStaticSize;
```

protected:

```
// Number of articulated part parameters for this PDU  
int myNumArtPartParams;  
  
// Number of attached part parameters for this PDU  
int myNumAttPartParams;  
  
// Temporary objects for reference returns
```

```
DtEntityIdentifier myEntityID;  
DtVector myVector;  
DtTaitBryan myEulerAngles;  
  
};  
#endif  
#endif
```

Euler.h

```

#include "LibMatrix.h"

/* Vrlink (and DIS) Euler angles are defined as successive right-hand
 * rotations about
 * reference Z axis heading or psi
 * the rotated Y axis pitch or theta
 * body X axis roll or phi
 *
 * This is the standard sequence of rotations for vehicle design and
 * simulation, and is known to some as the Tait-Bryan sequence.
 *
 * The body axis system for DIS is defined by
 * X forward
 * Y right
 * Z down
 *
 * The reference axis system is defined by some "fixed" or "world"
 * frame. When topographic-down is chosen as the reference frame,
 * you get heading relative to north, and pitch and roll relative to
 * the local horizon. See topoCoord.h.
 */

// Given Euler angles between a body and a reference frame, produce
// the transformation matrix that converts from the body frame to
// the
// reference frame (or visa-versa).
DT_DLL_MATRIX void DtEuler_to_BodyToRef(const DtTaitBryan &euler,
DtDcmRef body2ref);
DT_DLL_MATRIX void DtEuler_to_RefToBody(const DtTaitBryan &euler,
DtDcmRef ref2body);

// Given the transformation matrix that converts from a body frame
// to
// a reference frame (or visa-versa), produce the Euler angles
// between
// the body and the reference frames.
DT_DLL_MATRIX void DtBodyToRef_to_Euler(DtConstDcm body2ref,
DtTaitBryan *euler);
DT_DLL_MATRIX void DtRefToBody_to_Euler(DtConstDcm ref2body,
DtTaitBryan *euler);

// Given Euler angles with respect to reference frame "in", and the
// transformation matrix between frame "in" and frame "out", produce
// the Euler angles with respect to reference frame "out".
DT_DLL_MATRIX void DtEulerToEuler(const DtTaitBryan &in,
DtConstDcm in2out,
DtTaitBryan *out);

```

eventId.h

```
#include "NetStructs.h"

#if DtHLA

class DT_DLL_VL DtEventId
{
public:
    // Construct a DtEventId from a character string consisting of
    // the event number followed by a colon, and then an object name.
    // For example: DtEventId event("36:Object1");
    DtEventId(const char* str);

    // Construct a DtEventId from a name and an event number.
    DtEventId(const DtGlobalObjectDesignator& des, int eventNum);

    // Construct a DtEventId from network representation.
    DtEventId(const DtNetEventId& netId);

    // Same as DtEventId("", 0);
    DtEventId();

    // Set from network representation/
    void setFromNet(const DtNetEventId& netId);

    // Convert to network representation. Returns ptr to internal
    memory.
    DtNetEventId* netRep() const;

    // equivalence operators.
    int operator==(const DtEventId &otherGuy) const;
    int operator!=(const DtEventId &otherGuy) const;

    // Set/Get event number component
    void setEventNum(int eventNum);
    int eventNum() const;

    // Set/Get object designator component
    const DtGlobalObjectDesignator& objectId() const;
    void setObjectId(const DtGlobalObjectDesignator& name);

    // Set/Get object designator component
    const DtGlobalObjectDesignator& globalId() const;
    void setGlobalId(const DtGlobalObjectDesignator& name);

    // Convert to string.
    char *string() const;
```



```

public:
    // Return a null DtEventId.
    static const DtEventId& nullId();

protected:

    // initialize
    void init(const DtGlobalObjectDesignator& des, int eventNum);

protected:

    DtString myObjName;
    int myEventNum;
};
#else

class DT_DLL_VL DtEventId
{
public:
    DtEventId(const char* str);
    DtEventId(int site, int host, int eventNum);
    DtEventId(DtNetEventId netId);
    DtEventId();

    void init(int site, int host, int eventNum);

    void setFromNet(DtNetEventId netId);
    operator DtNetEventId () const;

    int site() const;
    int host() const;
    int eventNum() const;

    void setSite(int site);
    void setHost(int host);
    void setEventNum(int eventNum);

    char *string() const;

    int operator==(const DtEventId &otherGuy) const;
    int operator!=(const DtEventId &otherGuy) const;

    static const DtEventId& nullId();

public:

    // Public for backwards compatibility.
    DtSimulationAddress DtSimulator;
    DtU16 DtEvent;
};

```

```
#endif
```

```
typedef DtEventId DtEventID;
```

eventReptPdu.h

```

#ifndef !DtHLA

#include "hostStructs.h"
#include "pduWithData.h"
#define DtEventReportStaticSize 40

class DT_DLL_VL DtEventReportPdu;
typedef DtEventReportPdu DtEventReportInteraction;
typedef void (*DtEventReportPduCb)(DtEventReportPdu *pdu, void
*usr);

class DT_DLL_VL DtEventReportPdu : public DtPduWithData
{
public:

/***** STANDARD CONSTRUCTORS *****/

    DtEventReportPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtEventReportPdu(const DtNetEventReportPdu *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    static DtPdu *create(const DtNetPacket *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    static void addCallback(DtExerciseConn *conn,
        DtEventReportPduCb cb, void *usr);
    static void removeCallback(DtExerciseConn *conn,
        DtEventReportPduCb cb, void *usr);

/***** INSPECTOR FUNCTIONS *****/

    DtEventType eventType() const;

/***** MUTATOR FUNCTIONS *****/

    void setEventType(DtEventType type);

    void printData() const;

protected:

    DtPduKind internalGetPduKind() const {return DtEventReportPdu-
Kind;}
    DtNetDataPduTail *dataSection() const;
};
#endif

```

eventReptRPdu.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "hostStructs.h"
#include "pduWithData.h"
#include "exerciseConn.h"

// class DtEventReptRPdu:
//
// Instances of DtEventReptRPdu are used to send/receive
// reliable event report PDUs.

// DtEventReptRPdu callback function signature.
typedef void (*DtEventReptRPduCb)(DtEventReptRPdu* pdu, void* usr);

class DT_DLL_VL DtEventReptRPdu : public DtPduWithData
{
public:

    // Constructors
    DtEventReptRPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtEventReptRPdu(const DtNetEventReportPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtEventReptRPdu(const DtEventReptRPdu& orig);

    // destructor
    virtual ~DtEventReptRPdu();

    // Assignment operator
    DtEventReptRPdu& operator=(const DtEventReptRPdu& orig);

    // Set/Get event type.
    virtual void setEventType(DtEventType type);
    virtual DtEventType eventType() const;

    // Print the data contained by this PDU
    virtual void printData() const;

public:

    // Create a DtEventReptRPdu.
    // Caller is responsible for deletion.
    static DtPdu* create(const DtNetPacket* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // Add (register)/Remove (deregister) function to be called when
    // pdu is received.
```

```
static void addCallback(DtExerciseConn* conn,
    DtEventReptRPduCb cb, void* usr);
static void removeCallback(DtExerciseConn* conn,
    DtEventReptRPduCb cb, void* usr);

protected:

    // Returns the internal PDU kind
    virtual DtPduKind internalGetPduKind() const;

    // Returns the data portion of the net structure.
    virtual DtNetDataPduTail* dataSection() const;

protected:

    static int theStaticSize;

};
#endif
#endif
```

fedReader.h

```
#include "stdio.h"
#include "fedTarget.h"

enum DtFedReadStatus
{
    DtFedReadOK,
    DtCouldNotOpenFed,
    DtErrorReadingFed
};

// class DtFedReader:
//
// Instances of DtFedReader are used to read a FED file, and fill in
// a
// DtFedTarget with the information read.

class DT_DLL_VLUTIL DtFedReader
{
public:

    // Constructor
    DtFedReader();

    // Destructor
    virtual ~DtFedReader();

    // copy constructor
    DtFedReader(const DtFedReader& orig);

    // assignment operator
    DtFedReader& operator=(const DtFedReader& orig);

    // Read the FED file fedFileName, filling in intMgr and objMgr
    virtual DtFedReadStatus readFed(const char* fedFileName,
        DtFedTarget* fedTarget);

protected:

    // Returns 1 after reading a line containing str
    // Returns 0 if hit EOF without finding such a line
    virtual int findString(FILE* fp, const char* str);

    // Read a line from the file. Returns NULL if hit EOF.
    // Returns a ptr to static memory that is reused in subsequent
    // calls.
    virtual char* readLine(FILE* fp);
```


fedTarget.h

```
#include "MachineTypes.h"

enum DtTransportType
{
    DtBest_Effort,
    DtReliable
};

enum DtOrderType
{
    DtReceive,
    DtTimestamp
};

// class DtFedTarget:
//
// DtFedTarget is an abstract class representing some representation
// of FED file data that can be filled in by a DtFedReader.

class DT_DLL_VLUTIL DtFedTarget
{
public:

    // Constructor
    DtFedTarget();

    // Destructor
    virtual ~DtFedTarget();

    // copy constructor
    DtFedTarget(const DtFedTarget& orig);

    // assignment operator
    DtFedTarget& operator=(const DtFedTarget& orig);

    // Define a class.  Used when initializing from FED file.
    virtual void addInterClass(const char* name,
        const char* baseName, DtTransportType transport,
        DtOrderType order) = 0;

    // Define a class.  Used when initializing from FED file.
    virtual void addObjClass(const char* name,
        const char* baseName) = 0;

    // Define a parameter of a class.
    // Used when initializing from FED file.
    virtual void addParameter(const char* className,
```



```
    const char* paramName) = 0;  
  
    // Define an attribute of a class.  
    // Used when initializing from FED file.  
    virtual void addAttribute(const char* className,  
        const char* attributeName, DtTransportType transport,  
        DtOrderType order) = 0;  
};
```

filename.h

```
#include "str.h"

// class DtFilename:
//
// Instances of DtFilename are DtStrings which convert filenames to
// the
// native format.
class DT_DLL_VLUTIL DtFilename : public DtString
{
public:
    // default constructor
    DtFilename();

    // constructor
    DtFilename(const char* str);

    // destructor
    virtual ~DtFilename();

    // copy constructor
    DtFilename(const DtFilename& orig);

    // assignment operator
    DtFilename& operator=(const DtFilename& orig);

    // Returns DtTrue if the filename is an absolute filename, or a
    relative
    // filename, respectively.
    virtual DtBoolean absolute() const;
    virtual DtBoolean relative() const;

    // Constructs a new filename by combining the path
    // with filename, which is an absolute or relative path.
    // If the filename is an absolute filename, then filename is
    returned.
    // If the filename is a relative filename, then path is prepended
    to it.
    static DtFilename combine(const DtFilename& path, const DtFile-
    name& filename);

    // returns the separator used in representing directories.
    static char separator();

protected:
    // Replaces slashes.  On unix, this replaces backslashes with
    forward
    // slashes.  Vice versa on win32.  Hail Mr Gates.
```

```
    // This is not virtual because we call this from the constructors
and
    // destructor.
    void replaceSlashes();

protected:
    static const char theSeparator;
};
```

firePdu.h

```
#if !DtHLA

#include "hostStructs.h"
#include "interaction.h"
#include "packets/fireP.h"
#include "mathTypes.h"
#define DtFireStaticSize 96

#include "chooseDefines.h"

class DT_DLL_VL DtFirePdu;
typedef DtFirePdu DtFireInteraction;
typedef void (*DtFireInteractionCb)(DtFireInteraction *inter, void
*usr);

class DT_DLL_VL DtFirePdu : public DtInteraction
{
public:

    /***** STANDARD CONSTRUCTORS *****/

    DtFirePdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtFirePdu(const DtNetFirePdu *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    static DtPdu *create(const DtNetPacket *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    static void addCallback(DtExerciseConn *conn,
        DtFireInteractionCb cb, void *usr);
    static void removeCallback(DtExerciseConn *conn,
        DtFireInteractionCb cb, void *usr);

    /***** ADDITIONAL CONSTRUCTORS *****/
    DtFirePdu(const DtEntityIdentifier &attacker,
        const DtEntityIdentifier &target,
        const DtEntityIdentifier &munition,
        const DtEventID &eventId,
        DtConstVector location,
        const DtBurstDescriptor &burst,
        DtConstVector linVelocity,
        const double range,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    /***** INSPECTOR FUNCTIONS *****/

    const DtObjectId &attackerId() const;
    const DtObjectId &targetId() const;
```

```

    const DtObjectId &munitionId() const;
    const DtEventID &eventId() const;
#if DtDISVERS >= 5
    unsigned long fireMissionIndex() const;
#endif
    DtConstVector location() const;
    DtConstVector velocity() const;
    double range() const;
    DtDetonatorFuze fuseType() const;
    const DtEntityType &munitionType() const;
    int quantity() const;
    int rate() const;
    DtWarheadType warheadType() const;

// NOT USED IN HLA
    const DtBurstDescriptor &burst() const;

    /***** MUTATOR FUNCTIONS *****/

    void setAttackerId(const DtObjectId &id);
    void setTargetId(const DtObjectId &id);
    void setMunitionId(const DtObjectId &id);
    void setEventId(const DtEventID &id);
#if DtDISVERS >= 5
    void setFireMissionIndex(unsigned long i);
#endif
    void setLocation(DtConstVector location);
    void setVelocity(DtConstVector velocity);
    void setRange(double range);
    void setFuseType(DtDetonatorFuze f);
    void setMunitionType(const DtEntityType &t);
    void setQuantity(int q);
    void setRate(int r);
    void setWarheadType(DtWarheadType t);
// NOT USED IN HLA
    void setBurst(const DtBurstDescriptor &burst);
    void printData() const;

protected:
    DtPduKind internalGetPduKind() const {return DtFirePduKind;}
    DtObjectId att;
    DtObjectId tar;
    DtObjectId mun;
    DtEventID event;
    DtVector loc;
    DtVector vel;
    DtBurstDescriptor theBurst;
    DtEntityType myMunType;
};
#endif

```

fom.h

```
#if DtHLA
#include "MachineTypes.h"
#include "chooseDefines.h"

#include "fedTarget.h"
#include "RTI.hh"

// #include "chooseDefines.h"

// class DtFom:
//
// An instances of DtFom is used to represent a particular FOM, as
// described
// by a FED file. Object classes are represented by DtObjClass-
// Descs;
// Interaction classes by DtInterClassDescs.
// Being a DtFedTarget, a DtFom can be filled in by
// a DtFedReader. Alternatively, you can fill it in programati-
// cally,
// with addObjClass, addInterClass, addParameter and addAttribute.
// Once
// so initialized, you can query for a class descriptor associated
// with a class.

class DT_DLL_VL DtFom : public DtFedTarget
{
public:

    // Constructor
    DtFom(DtExerciseConn* exConn);

    // Destructor
    virtual ~DtFom();

    // copy constructor
    DtFom(const DtFom& orig);

    // assignment operator
    DtFom& operator=(const DtFom& orig);

    // Define a class. Used when initializing from FED file.
    virtual void addInterClass(const char* name,
        const char* baseName, DtTransportType transport,
        DtOrderType order);

    // Define a class. Used when initializing from FED file.
    virtual void addObjClass(const char* name,
```

```

    const char* baseName);

// Define an parameter of a class.
// Used when initializing from FED file.
virtual void addParameter(const char* className,
    const char* paramName);

// Define an attribute of a class.
// Used when initializing from FED file.
virtual void addAttribute(const char* className,
    const char* attrName, DtTransportType transport,
    DtOrderType order);

// Returns the class descriptor for the name
virtual DtInterClassDesc* interClassByName(const char *name)
const;

// Returns the class descriptor for the name
virtual DtObjClassDesc* objClassByName(const char *name) const;

// Returns the class descriptor for the handle
virtual DtInterClassDesc* interClassByHandle(
    RTI::InteractionClassHandle handle) const;

// Returns the class descriptor for the handle
virtual DtObjClassDesc* objClassByHandle(
    RTI::ObjectClassHandle handle) const;

// Returns the first, last and next object classes in the fom.
// Next returns NULL if current is the last class, or if current
is NULL.
// These functions facilitate looping through classes like this:
// for (desc = firstObjClass; desc; desc = nextObjClass(desc))
virtual DtObjClassDesc* firstObjClass() const;
virtual DtObjClassDesc* lastObjClass() const;
virtual DtObjClassDesc* nextObjClass(DtObjClassDesc* current)
const;

// Returns the first, last and next interaction class in the fom.
// Next returns NULL if current is the last class, or if current
is NULL.
// These functions facilitate looping through classes like this:
// for (desc = firstInterClass; desc; desc = nextInter-
Class(desc))
virtual DtInterClassDesc* firstInterClass() const;
virtual DtInterClassDesc* lastInterClass() const;
virtual DtInterClassDesc* nextInterClass(DtInterClassDesc*
current) const;

// Returns our RTI ambassador

```

```
    virtual DtVrlRtiAmbassador* rtiAmb();

    // prints the contents of the FOM
    virtual void print() const;

public:

    // Set/Get max number of object or interaction classes
    static void setMaxClasses(int num);
    static int maxClasses();

protected:

    DtExerciseConn* myExConn;
    DtInterClassDesc** myInterClasses;
    DtObjClassDesc** myObjClasses;

    int myLowestObj;
    int myHighestObj;
    int myLowestInter;
    int myHighestInter;

#if !DtRTI103
    char* myObjRootName;
    char* myInterRootName;
#endif

protected:

    static int theMaxClasses;
};
#include "exerciseConn.h"

#include "chooseDefines.h"

#endif
```


fomMapper.h

```

#if DtHLA

#include "RTI.hh"
#include "hostStructs.h"
#include "hashlist.h"

#include "chooseDefines.h"

// class DtFomMapper:
//
// Instances of DtFomMapper are used to provide FOM agility in VR-
// Link
// by mapping VR-Link classes and functionality to FOM classes,
// attributes and parameters. DtFomMapper is an abstract class.
// Most FOM
// Mappers (including DtRprFomMapper) are derived from DtEmptyFom-
// Mapper, and
// not directly from DtFomMapper.

class DT_DLL_VL DtFomMapper
{
public:

    // A function that chooses an interaction class from the FOM
    based on
    // the contents of a derived DtInteraction object.
    typedef char* (*DtInteractionClassChooser)(
        DtExerciseConn* exConn, const DtInteraction& inter);

    // A function that chooses an object class from the FOM based
    based on the
    // name of a C++ class derived from DtObjectPublisher, and arbi-
    tary user
    // data (e.g. entity type).
    typedef char* (*DtObjectClassChooser)(
        const char* vrlinkClassName, DtExerciseConn* conn, void* usr);

public:

    // Default constructor. Typically, most initialization takes
    place within
    // init(), which is called by DtExerciseConn's constructor.
    DtFomMapper();

    // destructor
    virtual ~DtFomMapper();

```

```
// Set/Get the current factory that can be used to create
// DtHlaStateDecoders
virtual DtStateDecoderFactory* setStateDecoderFactory(
    DtStateDecoderFactory* fact);
virtual DtStateDecoderFactory* stateDecoderFactory();

// Set/Get the current factory that can be used to create
// DtHlaStateEncoders
virtual DtStateEncoderFactory* setStateEncoderFactory(
    DtStateEncoderFactory* fact);
virtual DtStateEncoderFactory* stateEncoderFactory();

// Set/Get the current factory that can be used to create
// DtInteractionDecoders
virtual DtInteractionDecoderFactory* setInteractionDecoderFac-
tory(
    DtInteractionDecoderFactory* fact);
virtual DtInteractionDecoderFactory* interactionDecoderFactory();

// Set/Get the current factory that can be used to create
// DtInteractionEncoders
virtual DtInteractionEncoderFactory* setInteractionEncoderFac-
tory(
    DtInteractionEncoderFactory* fact);
virtual DtInteractionEncoderFactory* interactionEncoderFactory();

// Set/Get the current factory that can be used to create
// DtHlaObjects
virtual DtHlaObjectFactory* setHlaObjectFactory(
    DtHlaObjectFactory* fact);
virtual DtHlaObjectFactory* hlaObjectFactory();

// Set/Get the current factory that can be used to create
// DtInteractions
virtual DtInteractionFactory* setInteractionFactory(
    DtInteractionFactory* fact);
virtual DtInteractionFactory* interactionFactory();

// Tell the FOM Mapper how to choose a FOM object class to use to
for
// objects being published by a particular kind of object
publisher. You
// can either associate a single FOM class name, or a FOM class
choosing
// function with the name of a C++ class derived from DtObject-
Publisher.
virtual void setObjectClassToChoose(
    const char* vlinkClassName, const char* fomClassName);
virtual void setObjectClassChooser(
    const char* vlinkClassName, DtObjectClassChooser chooser);
```

```
// Chooses an object class to represent an object managed by the
C++ class
// named vrlinkClassName (which should be a class derived from
// DtObjectPublisher). usr is additional user data that may be
required to
// make the choice. It is passed as usr to a chooser function
that has
// been registered with the FOM Mapper for this publisher type,
if any.
virtual char* chooseObjectClass(
    const char* vrlinkClassName, void* usr = NULL) const;

// Tell the FOM Mapper how to choose a FOM interaction class to
use to for
// a particular kind of derived DtInteraction. You can either
associate a
// single FOM class name, or a FOM class choosing function with
the name of
// a C++ class derived from DtInteraction.
virtual void setInteractionClassToChoose(
    const char* vrlinkClassName, const char* fomClassName);
virtual void setInteractionClassChooser(
    const char* vrlinkClassName, DtInteractionClassChooser
chooser);

// Chooses a FOM interaction class to be by a particular derived
// DtInteraction class named vrlinkClassName.
virtual char* chooseInteractionClass(const DtInteraction& inter)
const;

// Set/Get a list of names (char*) of FOM classes that can be
decoded into
// the derived DtInteraction class whose name is name. These
classes will
// be subscribed to, when you use a derived DtInteraction's
addCallback
// function. The DtFomMapper copies the data in the list passed
to
// setInteractionClasses, so the object can be deleted at the
caller's
// leisure. The list returned by interactionClasses is owned by
the
// DtFomMapper and shouldn't be deleted by the caller.
virtual void setInteractionClasses(const char* vrlinkClassName,
    const DtList& classes);
virtual DtList* interactionClasses(const char* vrlinkClassName);

// When there is only a single FOM class to be associated with a
```

```
// VR-Link interaction class, then there is no need to create a
DtList.
// Just use this:
virtual void setInteractionClass(const char* vrlinkClassName,
    const char* fomClassName);

// Set/Get a list of names (char*) of FOM object classes that
should be
// subscribed to by a particular subclass of VR-Link's
// DtReflectedObjectList. vrlinkClassName is the name of the
// DtReflectedObjectList subclass for which you are creating a
mapping.
// The DtFomMapper copies the data in the list passed to
// setObjectClassNames, so the object can be deleted at the
caller's
// leisure. The list returned by objectClassNames is owned by
the
// DtFomMapper and shouldn't be deleted by the caller.
virtual void setObjectClassNames(const char* vrlinkClassName,
    const DtList& classes);
virtual DtList* objectClassNames(const char* vrlinkClassName);

// Similar to objectClassNames, but returns a DtList of Object-
ClassHandles
// instead. The returned list is owned by the DtFomMapper, and
should not
// be deleted. In addition, it is only valid until the next call
to this
// function.
virtual DtList* objectClassHandles(const char* vrlinkClassName);

// When there is only a single FOM class to be associated with a
VR-Link
// DtReflectedObjectList class, then there is no need to create a
DtList.
// Just use this:
virtual void setObjectClass(const char* vrlinkClassName,
    const char* fomClassName);

// Initialize the instance, once an exercise connection is avail-
able.
// Must be overridden in subclasses.
virtual void init(DtExerciseConn* conn) = 0;

// Returns the version number of the FOM, as passed to setVer-
sion.
virtual void setVersion(double vers);
virtual double version() const;

protected:
```

```

// copy constructor - not implemented
DtFomMapper(const DtFomMapper& orig);

// assignment operator - not implemented
DtFomMapper& operator=(const DtFomMapper& orig);

// Return the FOM-class-chooser function, if any, that has been
associated
// with the name of a C++ class derived from DtObjectPublisher
using
// setObjectClassChooser.
virtual DtObjectClassChooser objectClassChooser(
    const char* vrlinkClassName) const;

// Return the FOM-class-chooser function, if any, that has been
associated
// with the name of a C++ class derived from DtInteraction using
// setInteractionClassChooser.
virtual DtInteractionClassChooser interactionClassChooser(
    const char* vrlinkClassName) const;

protected:

    DtExerciseConn* myExConn;

    // Current factories and other objects
    DtStateDecoderFactory* myDecFact;
    DtStateEncoderFactory* myEncFact;
    DtInteractionDecoderFactory* myInterDecFact;
    DtInteractionEncoderFactory* myInterEncFact;
    DtHlaObjectFactory* myObjFact;
    DtInteractionFactory* myInterFact;

    // Hashlist that maps names of C++ classes derived from DtInter-
action to
    // lists of names of FOM classes. Hashkeys are DtStringHashKeys.
Data are
    // DtLists of char*'s that indicate FOM class names.
    DtHashlist myIntMapList;

    // Hashlist that maps names of C++ classes derived from
    // DtReflectedObjectList to lists of names of FOM classes. Hash-
keys are
    // DtStringHashKeys. Data are DtLists of char*'s that indicate
FOM class
    // names.
    DtHashlist myObjMapList;

```

```
// Hashlist that maps names of C++ classes derived from DtObject-
Publisher
// to DtObjectClassChoosers - functions that can choose an object
class
// name from the FOM to use for the object.
DtHashlist myObjChooserList;

// Hashlist that maps names of C++ classes derived from DtObject-
Publisher
// to single FOM object class names.
DtHashlist mySimpleObjChooserList;

// Hashlist that maps names of C++ classes derived from DtInter-
action to
// DtInteractionClassChoosers - functions that can choose an
interaction
// class name from the FOM to use for the interaction.
DtHashlist myInterChooserList;

// Hashlist that maps names of C++ classes derived from DtInter-
action to
// single FOM object class names.
DtHashlist mySimpleInterChooserList;

// The object returned by objectClassHandles. It is reused and
overwritten
// each call.
DtList myObjHandleList;

// Just an empty list to return when no class list has been regis-
tered for
// a particular VR-Link class.
DtList myEmptyList;

protected:

    double myVersion;
};

#endif
```

gedBasFixWing.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"
#include "hostStructs.h"

#include "packets/isGroupOfP.h"

// class DtBasicFixedWingAircraftRecord:
//
// Instances of DtBasicFixedWingAircraftRecord are used to contain
// the
// entity state and logistics information about an individual entity
// within
// a group of entities.

class DT_DLL_VL DtBasicFixedWingAircraftRecord
{
public:

    // Constructors
    DtBasicFixedWingAircraftRecord();
    DtBasicFixedWingAircraftRecord(
        const DtBasicFixedWingAircraftRecord& orig);

    // Destructor
    virtual ~DtBasicFixedWingAircraftRecord();

    // Assignment operators
    DtBasicFixedWingAircraftRecord& operator=(
        const DtBasicFixedWingAircraftRecord& orig);
    DtBasicFixedWingAircraftRecord& operator=(
        const DtNetBasicFixedWingAircraftRecord& orig);

    // Set/Get the entity ID
    virtual void setEntityID(DtU16 value);
    virtual DtU16 entityID() const;

    // Set/Get the entity location X offset
    virtual void setX_Offset(DtInt16 value);
    virtual DtInt16 X_Offset() const;

    // Set/Get the entity location Y offset
    virtual void setY_Offset(DtInt16 value);
    virtual DtInt16 Y_Offset() const;

```

```
// Set/Get the entity location Z offset
virtual void setZ_Offset(DtInt16 value);
virtual DtInt16 Z_Offset() const;

// Set/Get appearance (as a 32 bit integer)
virtual void setAppearance(DtU32 app);
virtual DtU32 appearance() const;

// Set/Get appearance by attribute
virtual void setAppearanceAttribute(
    const DtAppearAttribute attr, const DtAppearAttrVal val);
virtual DtAppearAttrVal appearanceAttribute(
    const DtAppearAttribute attr) const;

// Set/Get the entity orientation psi
virtual void setPsi(DtInt8 value);
virtual DtInt8 psi() const;

// Set/Get the entity orientation theta
virtual void setTheta(DtInt8 value);
virtual DtInt8 theta() const;

// Set/Get the entity orientation phi
virtual void setPhi(DtInt8 value);
virtual DtInt8 phi() const;

// Set/Get the fuel status
virtual void setFuelStatus(DtInt8 value);
virtual DtInt8 fuelStatus() const;

// Set/Get the horizontal deviation
virtual void setHorizontalDeviation(DtInt8 value);
virtual DtInt8 horizontalDeviation() const;

// Set/Get the vertical deviation
virtual void setVerticalDeviation(DtInt8 value);
virtual DtInt8 verticalDeviation() const;

// Set/Get the movement speed
virtual void setMovementSpeed(DtInt16 value);
virtual DtInt16 movementSpeed() const;

// Print this record
virtual void print() const;

// Load/Unload data from a packet record
virtual void unloadBaseNetRecord(
    const DtNetBasicFixedWingAircraftRecord& record);
virtual void loadBaseNetRecord(
```



```
        DtNetBasicFixedWingAircraftRecord& record) const;

protected:

    DtU16    myEntityID;
    DtInt16  myX_Offset;
    DtInt16  myY_Offset;
    DtInt16  myZ_Offset;
    DtU32    myAppearance;
    DtInt8   myPsi;
    DtInt8   myTheta;
    DtInt8   myPhi;
    DtU8     myFuelStatus;
    DtU8     myHorizontalDeviation;
    DtU8     myVerticalDeviation;
    DtInt16  myMovementSpeed;

};
#endif
#endif
```

gedBasRotWing.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"
#include "hostStructs.h"

#include "packets/isGroupOfP.h"

// class DtBasicRotorWingAircraftRecord:
//
// Instances of DtBasicRotorWingAircraftRecord are used to contain
// the
// entity state and logistics information about an individual entity
// within
// a group of entities.

class DT_DLL_VL DtBasicRotorWingAircraftRecord
{
public:

    // Constructors
    DtBasicRotorWingAircraftRecord();
    DtBasicRotorWingAircraftRecord(
        const DtBasicRotorWingAircraftRecord& orig);

    // Destructor
    virtual ~DtBasicRotorWingAircraftRecord();

    // Assignment operators
    DtBasicRotorWingAircraftRecord& operator=(
        const DtBasicRotorWingAircraftRecord& orig);
    DtBasicRotorWingAircraftRecord& operator=(
        const DtNetBasicRotorWingAircraftRecord& orig);

    // Set/Get the entity ID
    virtual void setEntityID(DtU16 value);
    virtual DtU16 entityID() const;

    // Set/Get the entity location X offset
    virtual void setX_Offset(DtInt16 value);
    virtual DtInt16 X_Offset() const;

    // Set/Get the entity location Y offset
    virtual void setY_Offset(DtInt16 value);
    virtual DtInt16 Y_Offset() const;
```

```
// Set/Get the entity location Z offset
virtual void setZ_Offset(DtInt16 value);
virtual DtInt16 Z_Offset() const;

// Set/Get appearance (as a 32 bit integer)
virtual void setAppearance(DtU32 app);
virtual DtU32 appearance() const;

// Set/Get appearance by attribute
virtual void setAppearanceAttribute(
    const DtAppearAttribute attr, const DtAppearAttrVal val);
virtual DtAppearAttrVal appearanceAttribute(
    const DtAppearAttribute attr) const;

// Set/Get the entity orientation psi
virtual void setPsi(DtInt8 value);
virtual DtInt8 psi() const;

// Set/Get the entity orientation theta
virtual void setTheta(DtInt8 value);
virtual DtInt8 theta() const;

// Set/Get the entity orientation phi
virtual void setPhi(DtInt8 value);
virtual DtInt8 phi() const;

// Set/Get the fuel status
virtual void setFuelStatus(DtInt8 value);
virtual DtInt8 fuelStatus() const;

// Set/Get the horizontal deviation
virtual void setHorizontalDeviation(DtInt8 value);
virtual DtInt8 horizontalDeviation() const;

// Set/Get the vertical deviation
virtual void setVerticalDeviation(DtInt8 value);
virtual DtInt8 verticalDeviation() const;

// Set/Get the movement speed
virtual void setMovementSpeed(DtInt16 value);
virtual DtInt16 movementSpeed() const;

// Set/Get the turret azimuth
virtual void setTurretAzimuth(DtInt8 value);
virtual DtInt8 turretAzimuth() const;

// Set/Get the gun elevation
virtual void setGunElevation(DtInt8 value);
virtual DtInt8 gunElevation() const;
```

```
// Set/Get the turret slew rate
virtual void setTurretSlewRate(DtInt8 value);
virtual DtInt8 turretSlewRate() const;

// Set/Get the gun elevation rate
virtual void setGunElevationRate(DtInt8 value);
virtual DtInt8 gunElevationRate() const;

// Print this record
virtual void print() const;

// Load/Unload data from a packet record
virtual void unloadBaseNetRecord(
    const DtNetBasicRotorWingAircraftRecord& record);
virtual void loadBaseNetRecord(
    DtNetBasicRotorWingAircraftRecord& record) const;
```

protected:

```
DtU16    myEntityID;
DtInt16  myX_Offset;
DtInt16  myY_Offset;
DtInt16  myZ_Offset;
DtU32    myAppearance;
DtInt8   myPsi;
DtInt8   myTheta;
DtInt8   myPhi;
DtU8     myFuelStatus;
DtU8     myHorizontalDeviation;
DtU8     myVerticalDeviation;
DtInt16  myMovementSpeed;
DtInt8   myTurretAzimuth;
DtInt8   myGunElevation;
DtInt8   myTurretSlewRate;
DtInt8   myGunElevationRate;
```

```
};
#endif
#endif
```

gedBasSoldier.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"
#include "hostStructs.h"

#include "packets/isGroupOfP.h"

// class DtBasicGroundCombatSoldierRecord:
//
// Instances of DtBasicGroundCombatSoldierRecord are used to contain
// the
// entity state and logistics information about an individual entity
// within
// a group of entities.

class DT_DLL_VL DtBasicGroundCombatSoldierRecord
{
public:

    // Constructors
    DtBasicGroundCombatSoldierRecord();
    DtBasicGroundCombatSoldierRecord(
        const DtBasicGroundCombatSoldierRecord& orig);

    // Destructor
    virtual ~DtBasicGroundCombatSoldierRecord();

    // Assignment operators
    DtBasicGroundCombatSoldierRecord& operator=(
        const DtBasicGroundCombatSoldierRecord& orig);
    DtBasicGroundCombatSoldierRecord& operator=(
        const DtNetBasicGroundCombatSoldierRecord& orig);

    // Set/Get the entity ID
    virtual void setEntityID(DtU16 value);
    virtual DtU16 entityID() const;

    // Set/Get the entity location X offset
    virtual void setX_Offset(DtInt8 value);
    virtual DtInt8 X_Offset() const;

    // Set/Get the entity location Y offset
    virtual void setY_Offset(DtInt8 value);
    virtual DtInt8 Y_Offset() const;

```

```
// Set/Get the entity location Z offset
virtual void setZ_Offset(DtInt8 value);
virtual DtInt8 Z_Offset() const;

// Set/Get appearance (as a 32 bit integer)
virtual void setAppearance(DtU32 app);
virtual DtU32 appearance() const;

// Set/Get appearance by attribute
virtual void setAppearanceAttribute(
    const DtAppearAttribute attr, const DtAppearAttrVal val);
virtual DtAppearAttrVal appearanceAttribute(
    const DtAppearAttribute attr) const;

// Set/Get the entity orientation psi
virtual void setPsi(DtInt8 value);
virtual DtInt8 psi() const;

// Set/Get the entity orientation theta
virtual void setTheta(DtInt8 value);
virtual DtInt8 theta() const;

// Set/Get the entity orientation phi
virtual void setPhi(DtInt8 value);
virtual DtInt8 phi() const;

// Set/Get the entity speed
virtual void setEntitySpeed(DtInt8 value);
virtual DtInt8 entitySpeed() const;

// Set/Get the head azimuth
virtual void setHeadAzimuth(DtInt8 value);
virtual DtInt8 headAzimuth() const;

// Set/Get the head elevation
virtual void setHeadElevation(DtInt8 value);
virtual DtInt8 headElevation() const;

// Set/Get the head scan rate
virtual void setHeadScanRate(DtInt8 value);
virtual DtInt8 headScanRate() const;

// Set/Get the head elevation rate
virtual void setHeadElevationRate(DtInt8 value);
virtual DtInt8 headElevationRate() const;

// Print this record
virtual void print() const;
```

```
// Load/Unload data from a packet record
virtual void unloadBaseNetRecord(
    const DtNetBasicGroundCombatSoldierRecord& record);
virtual void loadBaseNetRecord(
    DtNetBasicGroundCombatSoldierRecord& record) const;

protected:

    DtU16    myEntityID;
    DtInt8   myX_Offset;
    DtInt8   myY_Offset;
    DtInt8   myZ_Offset;
    DtU32    myAppearance;
    DtInt8   myPsi;
    DtInt8   myTheta;
    DtInt8   myPhi;
    DtInt8   myEntitySpeed;
    DtInt8   myHeadAzimuth;
    DtInt8   myHeadElevation;
    DtInt8   myHeadScanRate;
    DtInt8   myHeadElevationRate;

};
#endif
#endif
```

gedBasVehicle.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"
#include "hostStructs.h"

#include "packets/isGroupOfP.h"

// class DtBasicGroundCombatVehicleRecord:
//
// Instances of DtBasicGroundCombatVehicleRecord are used to contain
// the
// entity state and logistics information about an individual entity
// within
// a group of entities.

class DT_DLL_VL DtBasicGroundCombatVehicleRecord
{
public:

    // Constructors
    DtBasicGroundCombatVehicleRecord();
    DtBasicGroundCombatVehicleRecord(
        const DtBasicGroundCombatVehicleRecord& orig);

    // Destructor
    virtual ~DtBasicGroundCombatVehicleRecord();

    // Assignment operators
    DtBasicGroundCombatVehicleRecord& operator=(
        const DtBasicGroundCombatVehicleRecord& orig);
    DtBasicGroundCombatVehicleRecord& operator=(
        const DtNetBasicGroundCombatVehicleRecord& orig);

    // Set/Get the entity ID
    virtual void setEntityID(DtU16 value);
    virtual DtU16 entityID() const;

    // Set/Get the entity location X offset
    virtual void setX_Offset(DtInt8 value);
    virtual DtInt8 X_Offset() const;

    // Set/Get the entity location Y offset
    virtual void setY_Offset(DtInt8 value);
    virtual DtInt8 Y_Offset() const;
```



```
// Set/Get the entity location Z offset
virtual void setZ_Offset(DtInt8 value);
virtual DtInt8 Z_Offset() const;

// Set/Get appearance (as a 32 bit integer)
virtual void setAppearance(DtU32 app);
virtual DtU32 appearance() const;

// Set/Get appearance by attribute
virtual void setAppearanceAttribute(
    const DtAppearAttribute attr, const DtAppearAttrVal val);
virtual DtAppearAttrVal appearanceAttribute(
    const DtAppearAttribute attr) const;

// Set/Get the entity orientation psi
virtual void setPsi(DtInt8 value);
virtual DtInt8 psi() const;

// Set/Get the entity orientation theta
virtual void setTheta(DtInt8 value);
virtual DtInt8 theta() const;

// Set/Get the entity orientation phi
virtual void setPhi(DtInt8 value);
virtual DtInt8 phi() const;

// Set/Get the entity speed
virtual void setEntitySpeed(DtInt8 value);
virtual DtInt8 entitySpeed() const;

// Set/Get the turret azimuth
virtual void setTurretAzimuth(DtInt8 value);
virtual DtInt8 turretAzimuth() const;

// Set/Get the gun elevation
virtual void setGunElevation(DtInt8 value);
virtual DtInt8 gunElevation() const;

// Set/Get the turret slew rate
virtual void setTurretSlewRate(DtInt8 value);
virtual DtInt8 turretSlewRate() const;

// Set/Get the gun elevation rate
virtual void setGunElevationRate(DtInt8 value);
virtual DtInt8 gunElevationRate() const;

// Print this record
virtual void print() const;
```

```
// Load/Unload data from a packet record
virtual void unloadBaseNetRecord(
    const DtNetBasicGroundCombatVehicleRecord& record);
virtual void loadBaseNetRecord(
    DtNetBasicGroundCombatVehicleRecord& record) const;

protected:

    DtU16    myEntityID;
    DtInt8   myX_Offset;
    DtInt8   myY_Offset;
    DtInt8   myZ_Offset;
    DtU32    myAppearance;
    DtInt8   myPsi;
    DtInt8   myTheta;
    DtInt8   myPhi;
    DtInt8   myEntitySpeed;
    DtInt8   myTurretAzimuth;
    DtInt8   myGunElevation;
    DtInt8   myTurretSlewRate;
    DtInt8   myGunElevationRate;

};
#endif
#endif
```

gedEnhFixWing.h

```

#ifndef DtEnhancedFixedWingAircraftRecord_H_
#define DtEnhancedFixedWingAircraftRecord_H_

#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"
#include "hostStructs.h"

#include "gedBasFixWing.h"
#include "packets/isGroupOfP.h"

// class DtEnhancedFixedWingAircraftRecord:
//
// Instances of DtEnhancedFixedWingAircraftRecord are used to
// contain the
// entity state and logistics information about an individual entity
// within
// a group of entities.

class DT_DLL_VL DtEnhancedFixedWingAircraftRecord :
    public DtBasicFixedWingAircraftRecord
{
public:

    // Constructors
    DtEnhancedFixedWingAircraftRecord();
    DtEnhancedFixedWingAircraftRecord(
        const DtEnhancedFixedWingAircraftRecord& orig);

    // Destructor
    virtual ~DtEnhancedFixedWingAircraftRecord();

    // Assignment operators
    DtEnhancedFixedWingAircraftRecord& operator=(
        const DtEnhancedFixedWingAircraftRecord& orig);
    DtEnhancedFixedWingAircraftRecord& operator=(
        const DtNetEnhancedFixedWingAircraftRecord& orig);

    // Set/Get the fuel status
    virtual void setSupplementalFuelStatus(DtU8 value);
    virtual DtU8 supplementalFuelStatus() const;

    // Set/Get the air maintenance status
    virtual void setAirMaintenanceStatus(DtU8 value);

```

```
virtual DtU8 airMaintenanceStatus() const;

// Set/Get the primary ammunition
virtual void setPrimaryAmmunition(DtU8 value);
virtual DtU8 primaryAmmunition() const;

// Set/Get the secondary ammunition
virtual void setSecondaryAmmunition(DtU8 value);
virtual DtU8 secondaryAmmunition() const;

// Print this record
virtual void print() const;

// Load/Unload data from a packet record
virtual void unloadNetRecord(
    const DtNetEnhancedFixedWingAircraftRecord& record);
virtual void loadNetRecord(
    DtNetEnhancedFixedWingAircraftRecord& record) const;

protected:

    DtU8    mySupplementalFuelStatus;
    DtU8    myAirMaintenanceStatus;
    DtU8    myPrimaryAmmunition;
    DtU8    mySecondaryAmmunition;

};

#endif
#endif
#endif
```

gedEnhRotWing.h

```

#ifndef DtEnhancedRotorWingAircraftRecord_H_
#define DtEnhancedRotorWingAircraftRecord_H_

#ifdef !DtHLA
#ifdef DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"
#include "hostStructs.h"

#include "gedBasRotWing.h"
#include "packets/isGroupOfP.h"

// class DtEnhancedRotorWingAircraftRecord:
//
// Instances of DtEnhancedRotorWingAircraftRecord are used to
// contain the
// entity state and logistics information about an individual entity
// within
// a group of entities.

class DT_DLL_VL DtEnhancedRotorWingAircraftRecord :
    public DtBasicRotorWingAircraftRecord
{
public:

    // Constructors
    DtEnhancedRotorWingAircraftRecord();
    DtEnhancedRotorWingAircraftRecord(
        const DtEnhancedRotorWingAircraftRecord& orig);

    // Destructor
    virtual ~DtEnhancedRotorWingAircraftRecord();

    // Assignment operators
    DtEnhancedRotorWingAircraftRecord& operator=(
        const DtEnhancedRotorWingAircraftRecord& orig);
    DtEnhancedRotorWingAircraftRecord& operator=(
        const DtNetEnhancedRotorWingAircraftRecord& orig);

    // Set/Get the fuel status
    virtual void setSupplementalFuelStatus(DtU8 value);
    virtual DtU8 supplementalFuelStatus() const;

    // Set/Get the air maintenance status
    virtual void setAirMaintenanceStatus(DtU8 value);

```

```
virtual DtU8 airMaintenanceStatus() const;

// Set/Get the primary ammunition
virtual void setPrimaryAmmunition(DtU8 value);
virtual DtU8 primaryAmmunition() const;

// Set/Get the secondary ammunition
virtual void setSecondaryAmmunition(DtU8 value);
virtual DtU8 secondaryAmmunition() const;

// Print this record
virtual void print() const;

// Load/Unload data from a packet record
virtual void unloadNetRecord(
    const DtNetEnhancedRotorWingAircraftRecord& record);
virtual void loadNetRecord(
    DtNetEnhancedRotorWingAircraftRecord& record) const;

protected:

    DtU8    mySupplementalFuelStatus;
    DtU8    myAirMaintenanceStatus;
    DtU8    myPrimaryAmmunition;
    DtU8    mySecondaryAmmunition;

};

#endif
#endif
#endif
```

gedEnhSoldier.h

```

#ifndef DtEnhancedGroundCombatSoldierRecord_H_
#define DtEnhancedGroundCombatSoldierRecord_H_

#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"
#include "hostStructs.h"

#include "gedBasSoldier.h"
#include "packets/isGroupOfP.h"

// class DtEnhancedGroundCombatSoldierRecord:
//
// Instances of DtEnhancedGroundCombatSoldierRecord are used to
// contain the
// entity state and logistics information about an individual entity
// within
// a group of entities.

class DT_DLL_VL DtEnhancedGroundCombatSoldierRecord :
    public DtBasicGroundCombatSoldierRecord
{

public:

    // Constructors
    DtEnhancedGroundCombatSoldierRecord();
    DtEnhancedGroundCombatSoldierRecord(
        const DtEnhancedGroundCombatSoldierRecord& orig);

    // Destructor
    virtual ~DtEnhancedGroundCombatSoldierRecord();

    // Assignment operators
    DtEnhancedGroundCombatSoldierRecord& operator=(
        const DtEnhancedGroundCombatSoldierRecord& orig);
    DtEnhancedGroundCombatSoldierRecord& operator=(
        const DtNetEnhancedGroundCombatSoldierRecord& orig);

    // Set/Get the water status

```

```
virtual void setWaterStatus(DtU8 value);
virtual DtU8 waterStatus() const;

// Set/Get the rest status
virtual void setRestStatus(DtRestStatus value);
virtual DtRestStatus restStatus() const;

// Set/Get the primary ammunition
virtual void setPrimaryAmmunition(DtU8 value);
virtual DtU8 primaryAmmunition() const;

// Set/Get the secondary ammunition
virtual void setSecondaryAmmunition(DtU8 value);
virtual DtU8 secondaryAmmunition() const;

// Print this record
virtual void print() const;

// Load/Unload data from a packet record
virtual void unloadNetRecord(
    const DtNetEnhancedGroundCombatSoldierRecord& record);
virtual void loadNetRecord(
    DtNetEnhancedGroundCombatSoldierRecord& record) const;
```

protected:

```
DtU8    myWaterStatus;
DtU8    myRestStatus;
DtU8    myPrimaryAmmunition;
DtU8    mySecondaryAmmunition;
```

```
};
```

```
#endif
```

```
#endif
```

```
#endif
```


gedEnhVehicle.h

```

#ifndef DtEnhancedGroundCombatVehicleRecord_H_
#define DtEnhancedGroundCombatVehicleRecord_H_

#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"
#include "hostStructs.h"

#include "gedBasVehicle.h"
#include "packets/isGroupOfP.h"

// class DtEnhancedGroundCombatVehicleRecord:
//
// Instances of DtEnhancedGroundCombatVehicleRecord are used to
// contain the
// entity state and logistics information about an individual entity
// within
// a group of entities.

class DT_DLL_VL DtEnhancedGroundCombatVehicleRecord :
    public DtBasicGroundCombatVehicleRecord
{
public:

    // Constructors
    DtEnhancedGroundCombatVehicleRecord();
    DtEnhancedGroundCombatVehicleRecord(
        const DtEnhancedGroundCombatVehicleRecord& orig);

    // Destructor
    virtual ~DtEnhancedGroundCombatVehicleRecord();

    // Assignment operators
    DtEnhancedGroundCombatVehicleRecord& operator=(
        const DtEnhancedGroundCombatVehicleRecord& orig);
    DtEnhancedGroundCombatVehicleRecord& operator=(
        const DtNetEnhancedGroundCombatVehicleRecord& orig);

    // Set/Get the fuel status
    virtual void setFuelStatus(DtU8 value);
    virtual DtU8 fuelStatus() const;

    // Set/Get the ground maintenance status
    virtual void setGroundMaintenanceStatus(DtU8 value);

```

```
virtual DtU8 groundMaintenanceStatus() const;

// Set/Get the primary ammunition
virtual void setPrimaryAmmunition(DtU8 value);
virtual DtU8 primaryAmmunition() const;

// Set/Get the secondary ammunition
virtual void setSecondaryAmmunition(DtU8 value);
virtual DtU8 secondaryAmmunition() const;

// Print this record
virtual void print() const;

// Load/Unload data from a packet record
virtual void unloadNetRecord(
    const DtNetEnhancedGroundCombatVehicleRecord& record);
virtual void loadNetRecord(
    DtNetEnhancedGroundCombatVehicleRecord& record) const;

protected:

    DtU8    myFuelStatus;
    DtU8    myGroundMaintenanceStatus;
    DtU8    myPrimaryAmmunition;
    DtU8    mySecondaryAmmunition;

};

#endif
#endif
#endif
```

gedLogVehicle.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"
#include "hostStructs.h"

#include "packets/isGroupOfP.h"

// class DtGroundLogisticsVehicleRecord:
//
// Instances of DtGroundLogisticsVehicleRecord are used to contain
// the
// entity state and logistics information about an individual entity
// within
// a group of entities.

class DT_DLL_VL DtGroundLogisticsVehicleRecord
{
public:

    // Constructors
    DtGroundLogisticsVehicleRecord();
    DtGroundLogisticsVehicleRecord(
        const DtGroundLogisticsVehicleRecord& orig);

    // Destructor
    virtual ~DtGroundLogisticsVehicleRecord();

    // Assignment operators
    DtGroundLogisticsVehicleRecord& operator=(
        const DtGroundLogisticsVehicleRecord& orig);
    DtGroundLogisticsVehicleRecord& operator=(
        const DtNetGroundLogisticsVehicleRecord& orig);

    // Set/Get the entity ID
    virtual void setEntityID(DtU16 value);
    virtual DtU16 entityID() const;

    // Set/Get the entity location X offset
    virtual void setX_Offset(DtInt8 value);
    virtual DtInt8 X_Offset() const;

    // Set/Get the entity location Y offset
    virtual void setY_Offset(DtInt8 value);
    virtual DtInt8 Y_Offset() const;

```

```
// Set/Get the entity location Z offset
virtual void setZ_Offset(DtInt8 value);
virtual DtInt8 Z_Offset() const;

// Set/Get appearance (as a 32 bit integer)
virtual void setAppearance(DtU32 app);
virtual DtU32 appearance() const;

// Set/Get appearance by attribute
virtual void setAppearanceAttribute(
    const DtAppearAttribute attr, const DtAppearAttrVal val);
virtual DtAppearAttrVal appearanceAttribute(
    const DtAppearAttribute attr) const;

// Set/Get the entity orientation psi
virtual void setPsi(DtInt8 value);
virtual DtInt8 psi() const;

// Set/Get the entity orientation theta
virtual void setTheta(DtInt8 value);
virtual DtInt8 theta() const;

// Set/Get the entity orientation phi
virtual void setPhi(DtInt8 value);
virtual DtInt8 phi() const;

// Set/Get the entity speed
virtual void setEntitySpeed(DtInt8 value);
virtual DtInt8 entitySpeed() const;

// Print this record
virtual void print() const;

// Load/Unload data from a packet record
virtual void unloadNetRecord(
    const DtNetGroundLogisticsVehicleRecord& record);
virtual void loadNetRecord(
    DtNetGroundLogisticsVehicleRecord& record) const;
```

protected:

```
DtU16    myEntityID;
DtInt8   myX_Offset;
DtInt8   myY_Offset;
DtInt8   myZ_Offset;
DtU32    myAppearance;
DtInt8   myPsi;
DtInt8   myTheta;
DtInt8   myPhi;
```

```
DtInt8    myEntitySpeed;  
  
};  
#endif  
#endif
```

genericDec.h

```
#if DtHLA

#ifndef DtGenericObjectDecoder_H_
#define DtGenericObjectDecoder_H_

#include "chooseDefines.h"
#include "hStateDecoder.h"

// Forward declaration of DtGenericStateRepository class.
// The DtGenericObjectDecoder class:

class DT_DLL_VL DtGenericObjectDecoder : public DtHlaStateDecoder
{
public:

    // default constructor
    DtGenericObjectDecoder(DtExerciseConn* exConn, DtObjClassDesc*
classDesc);

    // destructor
    virtual ~DtGenericObjectDecoder();

    // Virtual function override.
    virtual void decodeAttribute(
        DtStateRepository* stateRep,
        const RTI::AttributeHandleValuePairSet& attrs,
        int pairSetIndex) const;
};
#endif
```

genericEnc.h

```

#if DtHLA

#ifndef DtGenericObjectEncoder_H_
#define DtGenericObjectEncoder_H_

#include "hStateEncoder.h"

// Forward declaration of DtGenericStateRepository class.
// The DtGenericObjectEncoder class - the encoder used by
// DtGenericObjectPublisher.

class DT_DLL_VL DtGenericObjectEncoder : public DtHlaStateEncoder
{
public:

    // default constructor
    DtGenericObjectEncoder(DtExerciseConn* exConn, DtObjClassDesc*
classDesc);

    // destructor
    virtual ~DtGenericObjectEncoder();

    // Encode necessary values from the state repository. The second
state
    // repository represents what is seen by remote simulators and is
used to
    // determine when things need to be sent in the state message.
    // includeThese is a set of attributes that should be included
even if
    // their thresholds have not been exceeded. This DtGenericOb-
jectEncoder
    // assumes an update condition of "OnChange" for all attributes.
    virtual void encode(const DtStateRepository& stateRep,
        const DtStateRepository& asSeenByRemote,
        const RTI::AttributeHandleSet& neededAttrs,
        const RTI::AttributeHandleSet& includeThese,
        DtStateMsg* msg);

protected:

    virtual DtBoolean valueDiffers(
        const DtGenericStateRepository& stateRep,
        const DtGenericStateRepository& asSeenByRemote,
        RTI::AttributeHandle handle);
};
#endif

```

genericPub.h

```
#if DtHLA

#include "hObjPub.h"
#include "genericSR.h"

// class DtGenericObjectPublisher:
//
// Instances of DtGenericObjectPublisher are used for local or
// source
// simulations of objects for which a more specific publisher does
// not exist.
// The publisher sends data to the network when necessary. This
// class is
// derived from DtObjectPublisher, which does most of the work. The
// derived
// class mostly exists to provide typesafety, and to configure the
// base
// object.

class DtGenericObjectPublisher : public DtObjectPublisher
{
public:

    // Constructor - configure base class with proper FOM class,
    state
    // repositories, encoder and decoder.
    DtGenericObjectPublisher(DtExerciseConn* conn,
        const char* className,
        const char* objName = NULL);

    // destructor
    virtual ~DtGenericObjectPublisher();

    // Returns a pointer to the DtGenericStateRepository - through
    // which you can set current state information.
    virtual DtGenericStateRepository* genericStateRep();

    // Synonym for genericStateRep()
    virtual DtGenericStateRepository* gsr();
};
#endif
```


genericSR.h

```

#if DtHLA

#include "stateRep.h"

// class DtGenericStateRepository:
//
// Instances of DtGenericStateRepository are repositories for the
// state of
// an HLA object of unknown type. State is represented exactly the
// same way
// as in the FOM.

class DtGenericStateRepository : public DtStateRepository
{
public:

    // default constructor
    DtGenericStateRepository();

    // destructor
    virtual ~DtGenericStateRepository();

    // copy constructor
    DtGenericStateRepository(const DtGenericStateRepository& orig);

    // assignment operator
    DtGenericStateRepository& operator=(
        const DtGenericStateRepository& orig);

    // Sets the value of an attribute by handle.
    virtual void setValue(RTI::AttributeHandle handle,
        void* buffer, int size);

    // Returns the value of the attribute associated with handle,
    // or NULL if none exists.
    virtual void* value(RTI::AttributeHandle handle) const;

    // Returns the length of the value of the attribute associated
    // with handle,
    // or - if none exists.
    virtual int valueLength(RTI::AttributeHandle handle) const;

    // return an empty object of the same type as this
    virtual DtStateRepository* clone() const;

    // Print the state data.
    virtual void printData() const;

```

protected:

```
    RTI::AttributeHandleValuePairSet* myAttrs;  
};  
#endif
```

geodCoord.h

```

#include "vlVector.h"
#include "mathTypes.h"
#include "chooseDefines.h"

// class DtGeodeticCoord:
//
// Instances of DtGeodeticCoord are used to explicitly represent
// Geodetic coordinates.

class DT_DLL_VL DtGeodeticCoord : public DtVector
{
    friend DT_DLL_VL void DtGeocToGeod(DtConstVector gc, DtGeodetic-
Coord& gd);

public:

    // default constructor
    DtGeodeticCoord();

    // constructor
    DtGeodeticCoord(double lat, double lon, double alt);

    // destructor
    virtual ~DtGeodeticCoord();

    // copy constructor
    DtGeodeticCoord(const DtGeodeticCoord& orig);

    // assignment operator
    DtGeodeticCoord& operator=(const DtGeodeticCoord& orig);

    // Set from geocentric coordinate
    // (convert geocentric to geodetic).
    // Can also use assignment operator.
    virtual DtGeodeticCoord& setGeocentric(DtConstVector geocentric);

    // Get geocentric coordinate equivalent
    // (convert geodetic to geocentric).
    virtual void getGeocentric(DtVectorRef geocentric) const;

    // Get geocentric coordinate equivalent
    // (convert geodetic to geocentric).
    // Note that this is not returning a reference.
    virtual DtVector geocentric() const;

    // Set/Get the latitude.
    virtual void setLat(DtFloat64 lat);

```

```
virtual DtFloat64 lat() const;

// Set/Get the longitude.
virtual void setLon(DtFloat64 lon);
virtual DtFloat64 lon() const;

// Set/Get the altitude.
virtual void setAlt(DtFloat64 alt);
virtual DtFloat64 alt() const;
};
// Conversions between geocentric and geodetic:
// Use of these functions is discouraged in favor of DtGeodeticCoord
// member functions which perform conversions.
DT_DLL_VL void DtGeocToGeod(DtConstVector in, DtGeodeticCoord& out);
DT_DLL_VL void DtGeodToGeoc(const DtGeodeticCoord& in, DtVectorRef
out);

// Conversions between geocentric and geodetic coordinates depend
// on the particular MapDatum used. The default is DtWGS84, but you
// can set the current MapDatum using DtUseMapDatum declared below.

// Spheroid
typedef struct DtSpheroid
{
    DtFloat64 semiMajor;    // semimajor axis of the ellipsoid, in
meters
    DtFloat64 semiMinor;    // semiminor axis of the ellipsoid, in
meters
} DtSpheroid;

typedef struct DtMapDatum
{
    DtSpheroid spheroid;
    DtFloat64 datumShift[3]; // add this to wgs84 to get GCC in this
datum
} DtMapDatum;

// Set and Get current map datum
// NOTE: map datum memory passed to DtUseMapDatum() must remain
valid
// until next change of datum.

DT_DLL_VL void DtUseMapDatum(const DtMapDatum* md);
DT_DLL_VL const DtMapDatum* DtCurrentMapDatum();

// Here are a few of the most important spheroids; we should put all
13 in.
#define DtWGS84Spheroid          {6378137.0, 6356752.3142}
#define DtInternationalSpheroid {6378388.0, 6356911.946127}
#define DtClark1866Spheroid      {6378206.4, 6356583.8}
```

```
// ...and Mapping Datums

const DtMapDatum DtWGS84      = {DtWGS84Spheroid,      0.,  0.,
0.};
const DtMapDatum DtED50       = {DtInternationalSpheroid, -87., -
98., -121.};
const DtMapDatum DtNASANAD27  = {DtClark1866Spheroid,   -8.,
161., 179.};
const DtMapDatum DtNASBNAD27  = {DtClark1866Spheroid,   -8.,
159., 175.};
const DtMapDatum DtCONUSNAD27 = {DtClark1866Spheroid,   -8.,
160., 176.};

// inline functions
```

globalObjDes.h

```
// A DtGlobalObjectDesignator is an identifier that can be used to
// describe an object when communicating among simulations in DIS
// PDUs or HLA objects or interactions. In DIS, the Global Object
// Designator is the entity ID. In HLA, it is a DtString denoting
// an
// object name when using RTI 1.3, or an HLA ObjectID when using
// RTI 1.0.3.

#include "str.h"
#include "chooseDefines.h"

#if DtHLA
#else
typedef DtEntityIdentifier DtObjectId;
#endif

#if DtHLA
#if DtRTI103
typedef DtObjectId DtGlobalObjectDesignator;
#else
typedef DtString DtGlobalObjectDesignator;
#endif
#else
typedef DtObjectId DtGlobalObjectDesignator;
#endif
```

globldHLst.h

```

#include "hashlist.h"
#include "globalObjDes.h"

// class DtGlobalIdHashlist:
//
// Instances of DtGlobalIdHashlist are hashlists, where you can
// store associations between DtGlobalObjectDesignators and arbitrary
// pieces of data.

class DT_DLL_VL DtGlobalIdHashlist
{
public:
    // default constructor. Pass DtTrue only if the data that you are
    // listing
    // are instances of a class derived from DtListedObject.
    DtGlobalIdHashlist(DtBoolean intrusive = DtFalse);

    // destructor
    virtual ~DtGlobalIdHashlist();

    // In the event that an item with key already exists, then add
    // will replace the data and move it to the location in list.
    // If your data was allocated, do not add unless you know key is
    // unique, or a memory leak will result.

    // Add an association between key and data to the list
    virtual DtListItem* add(const DtGlobalObjectDesignator& key,
void* data);

    // Remove the current association involving key from the list.
    // Returns the data that was associated with key.
    virtual void* remove(const DtGlobalObjectDesignator& key);

    // Lookup the data associated with key.
    virtual void* lookup(const DtGlobalObjectDesignator& key) const;
    virtual void* operator[](const DtGlobalObjectDesignator& key)
const;

    // Get a DtList containing all of the void* data that you
    // have listed, regardless of keys.
    virtual const DtList* list() const;

protected:
    // copy constructor

```

```
DtGlobalIdHashlist(const DtGlobalIdHashlist& orig);

// assignment operator
DtGlobalIdHashlist& operator=(const DtGlobalIdHashlist& orig);

protected:

    DtList* myList;
    DtHashlist* myHashlist;
};
typedef DtGlobalIdHashlist DtGlobalIdHashList;
```


gridAxisRecFix.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"

#include "packets/gridDataP.h"

// class DtGridAxisRecordFixed:
//
// DtGridAxisRecordFixed is the fixed grid axis record used by the
// gridded
// data PDU class (DtGridDataPdu). It is also the base class for
// the
// variable grid axis record.

class DT_DLL_VL DtGridAxisRecordFixed
{
public:

    // Constructors
    DtGridAxisRecordFixed();
    DtGridAxisRecordFixed(const DtGridAxisRecordFixed& orig);

    // Destructor
    virtual ~DtGridAxisRecordFixed();

    // Assignment operator
    DtGridAxisRecordFixed& operator=(const DtGridAxisRecordFixed&
orig);
    DtGridAxisRecordFixed& operator=(const DtNetGridAxisRecordFixed&
orig);

    // Set/Get the domain initial Xi
    virtual void setDomainInitialXi(const DtFloat64& initial);
    virtual const DtFloat64& domainInitialXi() const;

    // Set/Get the domain final Xi
    virtual void setDomainFinalXi(const DtFloat64& final);
    virtual const DtFloat64& domainFinalXi() const;

    // Set/Get the domain number of points
    virtual void setDomainNumPointsXi(DtU16 num);
    virtual DtU16 domainNumPointsXi() const;

    // Set/Get the interleaf factor

```

```
virtual void setInterleafFactor(DtU8 factor);
virtual DtU8 interleafFactor() const;

// Get the axis type
virtual DtGridAxisType axisType() const;

// Set/Get the number of points. May return false if PDU size
exceeds
// the maximum size for a PDU.
virtual DtBoolean setNumPointsXi(DtU16 num);
virtual DtU16 numPointsXi() const;

// Set/Get the initial index
virtual void setInitialIndexXi(DtU16 index);
virtual DtU16 initialIndexXi() const;

// Set the total number of bytes required for the network struc-
ture.
virtual int recordLength() const;

// Print the sample type and data representation
virtual void print() const;

// Load/unload data from a packet record
virtual void baseUnloadNetRecord(const DtNetGridAxisRecordFixed&
record);
virtual void baseLoadNetRecord(DtNetGridAxisRecordFixed& record)
const;

public:

// Get the grid axis record type from a network structure
static DtGridAxisType axisType(const DtNetGridAxisRecordFixed*
rec);

// Get the grid axis record length from a network structure
static int recordLength(const DtNetGridAxisRecordFixed* rec);

protected:

// Constructor for derived classes
DtGridAxisRecordFixed(DtGridAxisType axType);

protected:

// Data members:
DtFloat64          myDomainInitialXi;
DtFloat64          myDomainFinalXi;
DtU16              myDomainNumPointsXi;
DtU8               myInterleafFactor;
```

```
DtGridAxisType      myAxisType;  
DtU16               myNumPointsXi;  
DtU16               myInitialIndexXi;  
  
};  
#endif  
#endif
```

gridAxiRecVar.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"
#include "hostStructs.h"
#include "exerciseConn.h"

#include "gridAxiRecFix.h"

// class DtGridAxisRecordVariable:
//
// Instances of DtGridAxisRecordVariable are used to contain the
// information
// in the grid axis variable spacing records used by the gridded
// data PDU.

class DT_DLL_VL DtGridAxisRecordVariable : public DtGridAxisRecord-
Fixed
{
public:

    // Constructors
    DtGridAxisRecordVariable();
    DtGridAxisRecordVariable(const DtGridAxisRecordVariable& orig);

    // Destructor
    virtual ~DtGridAxisRecordVariable();

    // Assignment operators
    DtGridAxisRecordVariable& operator=(
        const DtGridAxisRecordVariable& orig);
    DtGridAxisRecordVariable& operator=(
        const DtNetGridAxisRecordVariable& orig);

    // Get the total number of bytes required for the network struc-
    // ture.
    virtual int recordLength() const;

    // Set/Get the coordinate scale
    virtual void setCoordinateScaleXi(const DtFloat64& num);
    virtual const DtFloat64& coordinateScaleXi() const;

    // Set/Get the coordinate offset
    virtual void setCoordinateOffsetXi(const DtFloat64& num);
    virtual const DtFloat64& coordinateOffsetXi() const;
```

```
// Set/Get the number of points.
// If the number of points is changed, previous points are not
preserved.
virtual DtBoolean setNumPointsXi(DtU16 num);
virtual DtU16 numPointsXi() const;

// Set/Get an Xi value
virtual void setXi(int index, DtU16 x);
virtual DtU16 xi(int index) const;

// Print this record
virtual void print() const;

// Load/unload data from a packet record
virtual void unloadNetRecord(const DtNetGridAxisRecordVariable&
record);
virtual void loadNetRecord(DtNetGridAxisRecordVariable& record)
const;

public:

    // Get the grid axis record length from a network structure
    static int recordLength(DtNetGridAxisRecordVariable* rec);

protected:

    DtFloat64                myCoordinateScaleXi;
    DtFloat64                myCoordinateOffsetXi;
    DtU16*                   myXi;

};
#endif
#endif
```

gridDataPdu.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "hostStructs.h"
#include "pdu.h"
#include "mathTypes.h"

#include "packets/gridDataP.h"

#include "gridAxiRecFix.h"
#include "gridAxiRecVar.h"

#include "gridDataRec.h"
#include "gridDataRecT0.h"
#include "gridDataRecT1.h"
#include "gridDataRecT2.h"

#include "enviroTypeRec.h"

// class DtGriddedDataPdu:
//
// Instances of DtGriddedDataPdu are used to communicate environ-
// mental data
// defined on a grid.

// DtGriddedDataPdu callback function signature.
typedef void (*DtGriddedDataPduCb)(DtGriddedDataPdu* pdu, void*
usr);

class DT_DLL_VL DtGriddedDataPdu : public DtPdu
{
public:

    // Constructors
    DtGriddedDataPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtGriddedDataPdu(const DtNetGriddedDataPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtGriddedDataPdu(const DtGriddedDataPdu& orig);

    // destructor
    virtual ~DtGriddedDataPdu();

    // Create a DtGriddedDataPdu.
    // Caller is responsible for deletion.
    static DtPdu* create(const DtNetPacket* initial,
```

```
DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

// Add (register)/Remove (deregister) function to be called when
// pdu is received.
static void addCallback(DtExerciseConn* conn,
    DtGriddedDataPduCb cb, void* usr);
static void removeCallback(DtExerciseConn* conn,
    DtGriddedDataPduCb cb, void* usr);

// Assignment operator
DtGriddedDataPdu& operator=(const DtGriddedDataPdu& orig);

// Set/Get environmental simulation application identifier
virtual void setEnvSimAppID(const DtEntityIdentifier& app);
virtual const DtEntityIdentifier& envSimAppID() const;

// Set/Get field number
virtual void setFieldNumber(DtU16 field);
virtual DtU16 fieldNumber() const;

// Set/Get the PDU index number
virtual void setPduNumber(DtU16 num);
virtual DtU16 pduNumber() const;

// Set/Get total number of PDUs used to transmit the environ-
mental data
virtual void setPduTotal(DtU16 total);
virtual DtU16 pduTotal() const;

// Set/Get coordinate system
virtual void setCoordSystem(DtU16 sys);
virtual DtU16 coordSystem() const;

// Set/Get number of grid axes
virtual DtBoolean setNumGridAxes(DtU8 axes);
virtual DtU8 numGridAxes() const;

// Set/Get constant grid indicator
virtual void setConstantGrid(DtU8 constant);
virtual DtU8 constantGrid() const;

// Set/Get environment type
virtual void setEnvironmentType(const DtEnvironmentTypeRecord&
type);
virtual const DtEnvironmentTypeRecord& environmentType() const;

// Set/Get orientation
virtual void setOrientation(const DtTaitBryan& orientation);
virtual const DtTaitBryan& orientation() const;
```

```
// Set/Get sample time. This is a 64-bit quantity. The argument
// index selects between the most significant 32-bit word (0) and
least
// significant 32-bit word (1) of the complete 64-bit value.
virtual void setSampleTime(int index, DtU32 time);
virtual DtU32 sampleTime(int index) const;

// Set/Get total values.
virtual void setTotalValues(DtU32 total);
virtual DtU32 totalValues() const;

// Set/Get vector dimension. setVectorDim may return false if
the PDU
// size exceeds the maximum allowable size for a PDU.
virtual DtBoolean setVectorDim(DtU8 dim);
virtual DtU8 vectorDim() const;

// Get the grid axes record type for a particular record
virtual DtGridAxisType gridAxisRecordType(int index) const;

// Set/Get the grid axis record type and values of record index to
// values specified by the initializer in the second argument.
If index
// is out of bounds, or if the record type at the specified index
does
// not match the parameter record type (for a get), then DtFalse
is
// returned. If the space would exceed the maximum space
permitted for
// a PDU (for a set) then DtFalse may be returned.
virtual DtBoolean setGridAxisRecord(int index,
    const DtGridAxisRecordFixed& init);
virtual DtBoolean getGridAxisRecord(int index,
    DtGridAxisRecordFixed& record) const;
virtual DtBoolean setGridAxisRecord(int index,
    const DtGridAxisRecordVariable& init);
virtual DtBoolean getGridAxisRecord(int index,
    DtGridAxisRecordVariable& record) const;

// Get the grid data record type for a particular record
virtual DtGridDataRecType gridDataRecordType(int index) const;

// Set/Get the grid data record type and values of record index to
// values specified by the initializer in the second argument.
If index
// is out of bounds, or if the record type at the specified index
does
// not match the parameter record type (for a get), then DtFalse
is
```



```
// returned. If the space would exceed the maximum space
permitted for
// a PDU (for a set) then DtFalse may be returned.
virtual DtBoolean setGridDataRecord(int index,
    const DtGridDataRecordType0& init);
virtual DtBoolean getGridDataRecord(int index,
    DtGridDataRecordType0& record) const;
virtual DtBoolean setGridDataRecord(int index,
    const DtGridDataRecordType1& init);
virtual DtBoolean getGridDataRecord(int index,
    DtGridDataRecordType1& record) const;
virtual DtBoolean setGridDataRecord(int index,
    const DtGridDataRecordType2& init);
virtual DtBoolean getGridDataRecord(int index,
    DtGridDataRecordType2& record) const;

// Print the data contained by this PDU
virtual void printData() const;

protected:

// Returns the internal PDU kind
virtual DtPduKind internalGetPduKind() const;

// Casts the netPdu element to our network struct
virtual DtNetGriddedDataPdu* netPduStruct() const;

// Returns a pointer to the first byte of variable length data
virtual char* var() const;

// Casts the 'this' pointer for changing elements in a const
function
// Non-virtual since you can't override based on return type
DtGriddedDataPdu* self() const;

// Returns the length of a given grid axis record
virtual int gridAxisRecordLength(DtNetGridAxisRecordFixed* rec)
const;

// Returns the length of a given grid data record
virtual int gridDataRecordLength(DtNetGridDataRecord* rec) const;

// Returns a buffer offset to the given grid axis record
virtual int gridAxisRecordOffset(int recordIndex) const;

// Returns a buffer offset to the given grid data record
virtual int gridDataRecordOffset(int recordIndex) const;

// Resizes a grid axis record
```

```
    virtual DtBoolean resizeGridAxisRecord(DtNetGridAxisRecordFixed*
rec,
    int newSize);

    // Resizes a grid data record
    virtual DtBoolean resizeGridDataRecord(DtNetGridDataRecord* rec,
    int newSize);

protected:

    // Size of PDU with zero parameter records.
    static int theStaticSize;

protected:

    // Temporary objects for reference returns
    DtEntityIdentifier myEntityID;
    DtEnvironmentTypeRecord myEnvironmentType;
    DtTaitBryan myEulerAngles;

};
#endif
#endif
```

gridDataRec.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"

#include "packets/gridDataP.h"

// class DtGridDataRecord:
//
// DtGridDataRecord is an abstract base class for the grid data
// record
// classes which are used by the gridded data PDU class (DtGrid-
// DataPdu).

class DT_DLL_VL DtGridDataRecord
{
public:

    // Destructor
    virtual ~DtGridDataRecord();

    // Set/Get the grid data record sample type
    virtual void setSampleType(DtU16 type);
    virtual DtU16 sampleType() const;

    // Get the grid data record representation type
    virtual DtGridDataRecType recordType() const;

    // Get the total number of bytes required for the network struc-
    // ture.
    virtual int recordLength() const = 0;

    // Print the sample type and data representation
    virtual void print() const;

    // Load/unload data from a packet record
    virtual void baseUnloadNetRecord(const DtNetGridDataRecord&
record);
    virtual void baseLoadNetRecord(DtNetGridDataRecord& record)
const;

public:

    // Get the grid data record representation type from a network
    // structure

```

```
    static DtGridDataRecType recordType(DtNetGridDataRecord* rec);

protected:

    // Constructors (protected because this is an abstract base
class)
    DtGridDataRecord();
    DtGridDataRecord(const DtGridDataRecord& orig);
    DtGridDataRecord(DtGridDataRecType recType);

    // Assignment operator
    DtGridDataRecord& operator=(const DtGridDataRecord& orig);

protected:

    // Data members:
    DtU16                mySampleType;
    DtGridDataRecType    myRecordType;

};
#endif
#endif
```

gridDataRecT0.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"
#include "hostStructs.h"

#include "packets/gridDataP.h"
#include "gridDataRec.h"

// class DtGridDataRecordType0:
//
// Instances of DtGridDataRecordType0 are used to contain the infor-
// mation
// in the grid data records (of representation type 0) used by the
// gridded
// data PDU class.

class DT_DLL_VL DtGridDataRecordType0 : public DtGridDataRecord
{
public:

    // Constructors
    DtGridDataRecordType0();
    DtGridDataRecordType0(const DtGridDataRecordType0& orig);

    // Destructor
    virtual ~DtGridDataRecordType0();

    // Assignment operators
    DtGridDataRecordType0& operator=(
        const DtGridDataRecordType0& orig);
    DtGridDataRecordType0& operator=(
        const DtNetGridDataRecordType0& orig);

    // Get the total number of bytes required for the network struc-
    // ture.
    virtual int recordLength() const;

    // Set/Get the number of data bytes
    // When number of data bytes is changed, old values are not
    // preserved
    virtual DtBoolean setNumDataBytes(DtU16 bytes);
    virtual DtU16 numDataBytes() const;

    // Set/Get a data byte

```

```
virtual void setDataByte(int byteNum, DtU8 data);
virtual DtU8 dataByte(int byteNum) const;

// Set/Get multiple data bytes. Provide a pointer and data is
// copied to and from that memory. Will copy numDataBytes() on
get.
virtual DtBoolean setData(const DtU8* data, int numBytes);
virtual void getData(DtU8* buff) const;

// Print this record
virtual void print() const;

// Load/unload data from a packet record
virtual void unloadNetRecord(const DtNetGridDataRecordType0&
record);
virtual void loadNetRecord(DtNetGridDataRecordType0& record)
const;

public:

// Get the grid data record representation type from a network
structure
static int recordLength(DtNetGridDataRecordType0* rec);

protected:

DtU16                myNumDataBytes;
DtU8*                myData;

};
#endif
#endif
```

gridDataRecT1.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"
#include "hostStructs.h"

#include "packets/gridDataP.h"
#include "gridDataRec.h"

// class DtGridDataRecordType1:
//
// Instances of DtGridDataRecordType1 are used to contain the infor-
// mation
// in the grid data records (of representation type 1) used by the
// gridded
// data PDU class.

class DT_DLL_VL DtGridDataRecordType1 : public DtGridDataRecord
{
public:

    // Constructors
    DtGridDataRecordType1();
    DtGridDataRecordType1(const DtGridDataRecordType1& orig);

    // Destructor
    virtual ~DtGridDataRecordType1();

    // Assignment operators
    DtGridDataRecordType1& operator=(
        const DtGridDataRecordType1& orig);
    DtGridDataRecordType1& operator=(
        const DtNetGridDataRecordType1& orig);

    // Get the total number of bytes required for the network struc-
    // ture.
    virtual int recordLength() const;

    // Set/Get the field scale
    virtual void setFieldScale(DtFloat32 scale);
    virtual DtFloat32 fieldScale() const;

    // Set/Get the field offset
    virtual void setFieldOffset(DtFloat32 offset);
    virtual DtFloat32 fieldOffset() const;

```

```
// Set/Get the number of data values
// Old values are not preserved when number of values is changed
virtual DtBoolean setNumDataValues(DtU16 values);
virtual DtU16 numDataValues() const;

// Set/Get a data value
virtual void setDataValue(int index, DtInt16 data);
virtual DtInt16 dataValue(int index) const;

// Print this record
virtual void print() const;

// Load/unload data from a packet record
virtual void unloadNetRecord(const DtNetGridDataRecordType1&
record);
virtual void loadNetRecord(DtNetGridDataRecordType1& record)
const;

public:

    // Get the grid data record representation type from a network
    structure
    static int recordLength(DtNetGridDataRecordType1* rec);

protected:

    DtFloat32                myFieldScale;
    DtFloat32                myFieldOffset;
    DtU16                    myNumDataValues;
    DtInt16*                 myData;

};
#endif
#endif
```


gridDataRecT2.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"
#include "hostStructs.h"

#include "packets/gridDataP.h"

#include "gridDataRec.h"

// class DtGridDataRecordType2:
//
// Instances of DtGridDataRecordType2 are used to contain the infor-
// mation
// in the grid data records (of representation type 2) used by the
// gridded
// data PDU class.

class DT_DLL_VL DtGridDataRecordType2 : public DtGridDataRecord
{
public:

    // Constructors
    DtGridDataRecordType2();
    DtGridDataRecordType2(const DtGridDataRecordType2& orig);

    // Destructor
    virtual ~DtGridDataRecordType2();

    // Assignment operators
    DtGridDataRecordType2& operator=(
        const DtGridDataRecordType2& orig);
    DtGridDataRecordType2& operator=(
        const DtNetGridDataRecordType2& orig);

    // Get the total number of bytes required for the network struc-
    // ture.
    virtual int recordLength() const;

    // Set/Get the number of data values
    // Old values are not preserved when the number of values is
    // changed
    virtual DtBoolean setNumDataValues(DtU16 values);
    virtual DtU16 numDataValues() const;

```

```
// Set/Get a data value
virtual void setDataValue(int index, DtFloat32 data);
virtual DtFloat32 dataValue(int index) const;

// Print this record
virtual void print() const;

// Load/unload data from a packet record
virtual void unloadNetRecord(const DtNetGridDataRecordType2&
record);
virtual void loadNetRecord(DtNetGridDataRecordType2& record)
const;

public:

    // Get the grid data record representation type from a network
    structure
    static int recordLength(DtNetGridDataRecordType2* rec);

protected:

    DtU16                myNumDataValues;
    DtFloat32*           myData;

};
#endif
#endif
```

hAckDec.h

```

#if DtHLA

#include "interDecoder.h"

#include "chooseDefines.h"

// This can be used to declare parameter processor functions
// which are static members of a class.

#define DtDECLARE_ACK_PARAM_DECODER(paramName) \
    DtDECLARE_PARAM_DECODER(DtAcknowledgeInteraction, paramName)

#define DtDEFINE_SIMPLE_ACK_PARAM_DECODER( \
    paramName, netType, mutator) \
    DtDEFINE_SIMPLE_PARAM_DECODER( \
        DtAcknowledgeDecoder, DtAcknowledgeInteraction, \
        paramName, netType, mutator)

#define DtDEFINE_SIMPLE_ACK_PARAM_DECODER_WITH_CAST( \
    paramName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_PARAM_DECODER_WITH_CAST( \
        DtAcknowledgeDecoder, DtAcknowledgeInteraction, \
        paramName, netType, mutator, castExpr)

// class DtAcknowledgeDecoder:
//
// Instances of DtAcknowledgeDecoder are used to decode Acknowledge
// Interactions from the network.

class DT_DLL_VL DtAcknowledgeDecoder : public DtInteractionDecoder
{
public:

    // constructor
    DtAcknowledgeDecoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtAcknowledgeDecoder();

    // copy constructor
    DtAcknowledgeDecoder(const DtAcknowledgeDecoder& orig);

    // assignment operator
    DtAcknowledgeDecoder& operator=(const DtAcknowledgeDecoder&
orig);

```

protected:

```
    // Individual parameter decoders

    DtDECLARE_ACK_PARAM_DECODER(OriginatingEntity);
    DtDECLARE_ACK_PARAM_DECODER(ReceivingEntity);
    DtDECLARE_ACK_PARAM_DECODER(RequestID);
    DtDECLARE_ACK_PARAM_DECODER(AcknowledgeFlag);
    DtDECLARE_ACK_PARAM_DECODER(ResponseFlag);
};
#endif
```

hAckEnc.h

```

#if DtHLA

#include "interEncoder.h"

#define DtDECLARE_ACK_PARAM_ENCODER(paramName) \
    DtDECLARE_PARAM_ENCODER(DtAcknowledgeInteraction, paramName)

#define DtDEFINE_SIMPLE_ACK_PARAM_ENCODER(paramName, netType, \
    inspector) \
    DtDEFINE_SIMPLE_PARAM_ENCODER(DtAcknowledgeEncoder, \
    DtAcknowledgeInteraction, paramName, netType, inspector)

// class DtAcknowledgeEncoder:
//
// DtAcknowledgeEncoder is a DtInteractionEncoder that registers
// parameter encoders for all of the ACK interaction parameters.

class DT_DLL_VL DtAcknowledgeEncoder : public DtInteractionEncoder
{
public:

    DtAcknowledgeEncoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtAcknowledgeEncoder();

    // copy constructor
    DtAcknowledgeEncoder(const DtAcknowledgeEncoder& orig);

    // assignment operator
    DtAcknowledgeEncoder& operator=(const DtAcknowledgeEncoder&
orig);

protected:

    // Individual parameter encoders

    DtDECLARE_ACK_PARAM_ENCODER(OriginatingEntity);
    DtDECLARE_ACK_PARAM_ENCODER(ReceivingEntity);
    DtDECLARE_ACK_PARAM_ENCODER(RequestID);
    DtDECLARE_ACK_PARAM_ENCODER(AcknowledgeFlag);
    DtDECLARE_ACK_PARAM_ENCODER(ResponseFlag);
};
#endif

```

hAckInter.h

```
#if DtHLA

#include "hIntWEncDec.h"
#include "hostStructs.h"
#include "mathTypes.h"

#include "chooseDefines.h"

// class DtAcknowledgeInteraction:
//
// Instances of DtAcknowledgeInteraction are used to represent
// Acknowledge Interactions in the RPR ROM.

// Acknowledge interaction callback function signature.
typedef void (*DtAcknowledgeInteractionCb) (DtAcknowledgeInteraction* inter,
      void* usr);

class DT_DLL_VL DtAcknowledgeInteraction : public DtInteractionWithEncDec
{
public:

    // default constructor
    DtAcknowledgeInteraction();

    // destructor
    virtual ~DtAcknowledgeInteraction();

    // copy constructor
    DtAcknowledgeInteraction(const DtAcknowledgeInteraction& orig);

    // assignment operator
    DtAcknowledgeInteraction& operator=(const DtAcknowledgeInteraction& orig);

    // Print the interaction's data.
    virtual void printData() const;

    // Set/Get sender ID
    virtual void setSenderId(const DtEntityIdentifier& id);
    virtual const DtEntityIdentifier& senderId() const;

    // Set/Get receiver ID
    virtual void setReceiverId(const DtEntityIdentifier& id);
    virtual const DtEntityIdentifier& receiverId() const;
```

```

// Set/Get acknowledge flag
virtual void setAcknowledgeFlag(DtAcknowledgeFlag flag);
virtual DtAcknowledgeFlag acknowledgeFlag() const;

// Set/Get response flag
virtual void setResponseFlag(DtResponseFlag flag);
virtual DtResponseFlag responseFlag() const;

// Set/Get request ID
virtual void setRequestId(int id);
virtual int requestId() const;

// Name of the interaction (Not nec. the ClassName from the FOM).
virtual const char* name() const;

public:

    // Create a DtAcknowledgeInteraction.
    // Caller is responsible for deletion.
    static DtInteraction* create();

    // Add (register)/Remove (deregister) function to be called when
    // interaction occurs.
    static void addCallback(DtExerciseConn* conn,
        DtAcknowledgeInteractionCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtAcknowledgeInteractionCb cb, void* usr);

protected:

    DtEntityIdentifier myOrigEnt;
    DtEntityIdentifier myReceiveEnt;
    int myRequestId;
    DtAcknowledgeFlag myAckFlag;
    DtResponseFlag myRespFlag;
};
#endif

```

hActReqDec.h

```
#if DtHLA

#include "interDecoder.h"

#include "chooseDefines.h"

// This can be used to declare parameter processor functions
// which are static members of a class.

#define DtDECLARE_ACTREQ_PARAM_DECODER(paramName) \
    DtDECLARE_PARAM_DECODER(DtActionRequestInteraction, paramName)

#define DtDEFINE_SIMPLE_ACTREQ_PARAM_DECODER( \
    paramName, netType, mutator) \
    DtDEFINE_SIMPLE_PARAM_DECODER( \
        DtActionRequestDecoder, DtActionRequestInteraction, \
        paramName, netType, mutator)

#define DtDEFINE_SIMPLE_ACTREQ_PARAM_DECODER_WITH_CAST( \
    paramName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_PARAM_DECODER_WITH_CAST( \
        DtActionRequestDecoder, DtActionRequestInteraction, \
        paramName, netType, mutator, castExpr)

// class DtActionRequestDecoder:
//
// Instances of DtActionRequestDecoder are used to decode ActionRe-
// quest
// Interactions from the network.

class DT_DLL_VL DtActionRequestDecoder : public DtInteractionDecoder
{
public:

    // constructor
    DtActionRequestDecoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtActionRequestDecoder();

    // copy constructor
    DtActionRequestDecoder(const DtActionRequestDecoder& orig);

    // assignment operator
    DtActionRequestDecoder& operator=(const DtActionRequestDecoder&
orig);
```


protected:

```
    // Individual parameter decoders

    DtDECLARE_ACTREQ_PARAM_DECODER(OriginatingEntity);
    DtDECLARE_ACTREQ_PARAM_DECODER(ReceivingEntity);
    DtDECLARE_ACTREQ_PARAM_DECODER(RequestID);
    DtDECLARE_ACTREQ_PARAM_DECODER(Action);
    DtDECLARE_ACTREQ_PARAM_DECODER(FixedDatums);
    DtDECLARE_ACTREQ_PARAM_DECODER(VariableDatumSet);
};
#endif
```

hActReqEnc.h

```
#if DtHLA

#include "interEncoder.h"

#define DtDECLARE_ACTREQ_PARAM_ENCODER(paramName) \
    DtDECLARE_PARAM_ENCODER(DtActionRequestInteraction, paramName)

#define DtDEFINE_SIMPLE_ACTREQ_PARAM_ENCODER(paramName, netType, \
    inspector) \
    DtDEFINE_SIMPLE_PARAM_ENCODER(DtActionRequestEncoder, \
        DtActionRequestInteraction, paramName, netType, inspector)

// class DtActionRequestEncoder:
//
// DtActionRequestEncoder is a DtInteractionEncoder that registers
// parameter encoders for all of the ActionRequest interaction
// parameters.

class DT_DLL_VL DtActionRequestEncoder : public DtInteractionEncoder
{
public:

    DtActionRequestEncoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtActionRequestEncoder();

    // copy constructor
    DtActionRequestEncoder(const DtActionRequestEncoder& orig);

    // assignment operator
    DtActionRequestEncoder& operator=(const DtActionRequestEncoder&
orig);

protected:

    // Individual parameter encoders

    DtDECLARE_ACTREQ_PARAM_ENCODER(OriginatingEntity);
    DtDECLARE_ACTREQ_PARAM_ENCODER(ReceivingEntity);
    DtDECLARE_ACTREQ_PARAM_ENCODER(RequestID);
    DtDECLARE_ACTREQ_PARAM_ENCODER(Action);
    DtDECLARE_ACTREQ_PARAM_ENCODER(FixedDatums);
    DtDECLARE_ACTREQ_PARAM_ENCODER(VariableDatumSet);
};
#endif
```

hActReqInter.h

```

#if DtHLA

#include "hInterWData.h"
#include "hostStructs.h"
#include "mathTypes.h"

#include "chooseDefines.h"

// class DtActionRequestInteraction:
//
// Instances of DtActionRequestInteraction are used to represent
// ActionRequest Interactions in the RPR FOM. The variable and
// fixed
// datum records are accessed through DtInteractionWithData member
// functions.

// ActionRequest interaction callback function signature.
typedef void (*DtActionRequestInteractionCb)
    (DtActionRequestInteraction* inter, void* usr);

class DT_DLL_VL DtActionRequestInteraction : public DtInteraction-
    WithData
{
public:

    // default constructor
    DtActionRequestInteraction();

    // destructor
    virtual ~DtActionRequestInteraction();

    // copy constructor
    DtActionRequestInteraction(const DtActionRequestInteraction&
    orig);

    // assignment operator
    DtActionRequestInteraction& operator=(
        const DtActionRequestInteraction& orig);

    // Print the interaction's data.
    virtual void printData() const;

    // 'Set/Get sender ID
    virtual void setSenderId(const DtEntityIdentifier& id);
    virtual const DtEntityIdentifier& senderId() const;

    // Set/Get receiver ID

```

```
virtual void setReceiverId(const DtEntityIdentifier& id);
virtual const DtEntityIdentifier& receiverId() const;

// Set/Get request ID
virtual void setRequestId(int id);
virtual int requestId() const;

// Set/Get action ID
virtual void setActionId(DtActionId id);
virtual DtActionId actionId() const;

// Name of the interaction (Not nec. the ClassName from the FOM).
virtual const char* name() const;

public:

    // Create a DtActionRequestInteraction.
    // Caller is responsible for deletion.
    static DtInteraction* create();

    // Add (register)/Remove (deregister) function to be called when
    // interaction occurs.
    static void addCallback(DtExerciseConn* conn,
        DtActionRequestInteractionCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtActionRequestInteractionCb cb, void* usr);

protected:

    DtEntityIdentifier myOrigEnt;
    DtEntityIdentifier myReceiveEnt;
    int myRequestId;
    DtActionId myActionId;
};
#endif
```

hActReqObjDec.h

```

#if DtHLA

#include "chooseDefines.h"
#include "interDecoder.h"

// This can be used to declare parameter processor functions
// which are static members of a class.

#define DtDECLARE_ACTREQOBJ_PARAM_DECODER(paramName) \
    DtDECLARE_PARAM_DECODER(DtActionRequestToObjInter, paramName)

#define DtDEFINE_SIMPLE_ACTREQOBJ_PARAM_DECODER( \
    paramName, netType, mutator) \
    DtDEFINE_SIMPLE_PARAM_DECODER( \
        DtActionRequestToObjDecoder, DtActionRequestToObjInter, \
        paramName, netType, mutator)

#define DtDEFINE_SIMPLE_ACTREQOBJ_PARAM_DECODER_WITH_CAST( \
    paramName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_PARAM_DECODER_WITH_CAST( \
        DtActionRequestToObjDecoder, DtActionRequestToObjInter, \
        paramName, netType, mutator, castExpr)

// class DtActionRequestToObjDecoder:
//
// Instances of DtActionRequestToObjDecoder are used to decode
// ActionRequestToObject Interactions from the network.

class DT_DLL_VL DtActionRequestToObjDecoder : public DtInteraction-
Decoder
{
public:

    // constructor
    DtActionRequestToObjDecoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtActionRequestToObjDecoder();

    // copy constructor
    DtActionRequestToObjDecoder(const DtActionRequestToObjDecoder&
orig);

    // assignment operator
    DtActionRequestToObjDecoder& operator=(
        const DtActionRequestToObjDecoder& orig);

```

protected:

```
    // Individual parameter decoders

    DtDECLARE_ACTREQOBJ_PARAM_DECODER(ObjectIDs);
    DtDECLARE_ACTREQOBJ_PARAM_DECODER(Action);
};
#endif
```

hActReqObjEnc.h

```

#if DtHLA

#include "chooseDefines.h"
#include "interEncoder.h"

#define DtDECLARE_ACTREQOBJ_PARAM_ENCODER(paramName) \
    DtDECLARE_PARAM_ENCODER(DtActionRequestToObjInter, paramName)

#define DtDEFINE_SIMPLE_ACTREQOBJ_PARAM_ENCODER(paramName, netType, \
    inspector) \
    DtDEFINE_SIMPLE_PARAM_ENCODER(DtActionRequestToObjEncoder, \
    DtActionRequestToObjInter, paramName, netType, inspector)

// class DtActionRequestToObjEncoder:
//
// DtActionRequestToObjEncoder is a DtInteractionEncoder that regis-
// ters
// parameter encoders for all of the ActionRequestToObject interac-
// tion
// parameters.

class DT_DLL_VL DtActionRequestToObjEncoder : public DtInteraction-
Encoder
{
public:

    DtActionRequestToObjEncoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtActionRequestToObjEncoder();

    // copy constructor
    DtActionRequestToObjEncoder(const DtActionRequestToObjEncoder&
orig);

    // assignment operator
    DtActionRequestToObjEncoder& operator=(
        const DtActionRequestToObjEncoder& orig);

protected:

    // Individual parameter encoders

    DtDECLARE_ACTREQOBJ_PARAM_ENCODER(ObjectIDs);
    DtDECLARE_ACTREQOBJ_PARAM_ENCODER(Action);
};
#endif

```

hActReqObjInt.h

```
#if DtHLA

#include "hIntWEncDec.h"
#include "hostStructs.h"
#include "mathTypes.h"
#include "objDesList.h"
#include "chooseDefines.h"

// class DtActionRequestToObjInter:
//
// Instances of DtActionRequestToObjInter are used to represent
// ActionRequestToObject Interactions in the RPR FOM.

// ActionRequestToObject interaction callback function signature.
typedef void (*DtActionRequestToObjInterCb)
    (DtActionRequestToObjInter* inter, void* usr);

class DT_DLL_VL DtActionRequestToObjInter : public DtInteraction-
WithEncDec
{
public:

    // default constructor
    DtActionRequestToObjInter();

    // destructor
    virtual ~DtActionRequestToObjInter();

    // copy constructor
    DtActionRequestToObjInter(
        const DtActionRequestToObjInter& orig);

    // assignment operator
    DtActionRequestToObjInter& operator=(
        const DtActionRequestToObjInter& orig);

    // Print the interaction's data.
    virtual void printData() const;

    // Return the Object ID list
    virtual const DtGlobalObjectDesignatorList& objectIDs() const;
    virtual DtGlobalObjectDesignatorList* objectIDs();

    // Set/Get number of IDs.
    virtual void setNumIds(DtU8 numIds);
    virtual DtU8 numIds() const;

}
```



```
// Set/Get the nth ObjectId.
virtual void setNthObjectId(
    DtInt32 idIndex, const DtGlobalObjectDesignator& id);
virtual const DtGlobalObjectDesignator& nthObjectId(DtInt32
idIndex) const;

// Set/Get action enum
virtual void setAction(DtActionId action);
virtual DtActionId action() const;

// Name of the interaction (Not nec. the ClassName from the FOM).
virtual const char* name() const;

public:

    // Create a DtActionRequestToObjInter.
    // Caller is responsible for deletion.
    static DtInteraction* create();

    // Add (register)/Remove (deregister) function to be called when
    // interaction occurs.
    static void addCallback(DtExerciseConn* conn,
        DtActionRequestToObjInterCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtActionRequestToObjInterCb cb, void* usr);

protected:

    DtGlobalObjectDesignatorList myObjectList;
    DtActionId myAction;

protected:

    static char* theClassName;
};
#endif
```

hActRespDec.h

```
#if DtHLA

#include "chooseDefines.h"
#include "interDecoder.h"

// This can be used to declare parameter processor functions
// which are static members of a class.

#define DtDECLARE_ACTRESP_PARAM_DECODER(paramName) \
    DtDECLARE_PARAM_DECODER(DtActionResponseInteraction, paramName)

#define DtDEFINE_SIMPLE_ACTRESP_PARAM_DECODER( \
    paramName, netType, mutator) \
    DtDEFINE_SIMPLE_PARAM_DECODER( \
        DtActionResponseDecoder, DtActionResponseInteraction, \
        paramName, netType, mutator)

#define DtDEFINE_SIMPLE_ACTRESP_PARAM_DECODER_WITH_CAST( \
    paramName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_PARAM_DECODER_WITH_CAST( \
        DtActionResponseDecoder, DtActionResponseInteraction, \
        paramName, netType, mutator, castExpr)

// class DtActionResponseDecoder:
//
// Instances of DtActionResponseDecoder are used to decode ActionRe-
// sponse
// Interactions from the network.

class DT_DLL_VL DtActionResponseDecoder : public DtInteractionDe-
coder
{
public:

    // constructor
    DtActionResponseDecoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtActionResponseDecoder();

    // copy constructor
    DtActionResponseDecoder(const DtActionResponseDecoder& orig);

    // assignment operator
    DtActionResponseDecoder& operator=(const DtActionResponseDecoder&
orig);
```

protected:

```
    // Individual parameter decoders

    DtDECLARE_ACTRESP_PARAM_DECODER(OriginatingEntity);
    DtDECLARE_ACTRESP_PARAM_DECODER(ReceivingEntity);
    DtDECLARE_ACTRESP_PARAM_DECODER(RequestID);
    DtDECLARE_ACTRESP_PARAM_DECODER(RequestStatus);
    DtDECLARE_ACTRESP_PARAM_DECODER(FixedDatums);
    DtDECLARE_ACTRESP_PARAM_DECODER(VariableDatumSet);
};
#endif
```

hActRespEnc.h

```
#if DtHLA

#include "chooseDefines.h"
#include "interEncoder.h"

#define DtDECLARE_ACTRESP_PARAM_ENCODER(paramName) \
    DtDECLARE_PARAM_ENCODER(DtActionResponseInteraction, paramName)

#define DtDEFINE_SIMPLE_ACTRESP_PARAM_ENCODER(paramName, netType, \
    inspector) \
    DtDEFINE_SIMPLE_PARAM_ENCODER(DtActionResponseEncoder, \
        DtActionResponseInteraction, paramName, netType, inspector)

// class DtActionResponseEncoder:
//
// DtActionResponseEncoder is a DtInteractionEncoder that registers
// parameter encoders for all of the ActionResponse interaction
// parameters.

class DT_DLL_VL DtActionResponseEncoder : public DtInteractionEn-
coder
{
public:

    DtActionResponseEncoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtActionResponseEncoder();

    // copy constructor
    DtActionResponseEncoder(const DtActionResponseEncoder& orig);

    // assignment operator
    DtActionResponseEncoder& operator=(const DtActionResponseEncoder&
orig);

protected:

    // Individual parameter encoders

    DtDECLARE_ACTRESP_PARAM_ENCODER(OriginatingEntity);
    DtDECLARE_ACTRESP_PARAM_ENCODER(ReceivingEntity);
    DtDECLARE_ACTRESP_PARAM_ENCODER(RequestID);
    DtDECLARE_ACTRESP_PARAM_ENCODER(RequestStatus);
    DtDECLARE_ACTRESP_PARAM_ENCODER(FixedDatums);
    DtDECLARE_ACTRESP_PARAM_ENCODER(VariableDatumSet);
```

```
};  
#endif
```

hActRespInter.h

```
#if DtHLA

#include "hInterWData.h"
#include "hostStructs.h"
#include "mathTypes.h"
#include "chooseDefines.h"

// class DtActionResponseInteraction:
//
// Instances of DtActionResponseInteraction are used to represent
// ActionResponse Interactions in the RPR FOM. The variable and
// fixed
// datum records are accessed through DtInteractionWithData member
// functions.

// ActionResponse interaction callback function signature.
typedef void (*DtActionResponseInteractionCb)
    (DtActionResponseInteraction* inter, void* usr);

class DT_DLL_VL DtActionResponseInteraction : public DtInteraction-
    WithData
{
public:

    // default constructor
    DtActionResponseInteraction();

    // destructor
    virtual ~DtActionResponseInteraction();

    // copy constructor
    DtActionResponseInteraction(const DtActionResponseInteraction&
    orig);

    // assignment operator
    DtActionResponseInteraction& operator=(
        const DtActionResponseInteraction& orig);

    // Print the interaction's data.
    virtual void printData() const;

    // 'Set/Get sender ID
    virtual void setSenderId(const DtEntityIdentifier& id);
    virtual const DtEntityIdentifier& senderId() const;

    // Set/Get receiver ID
    virtual void setReceiverId(const DtEntityIdentifier& id);
```

```
virtual const DtEntityIdentifier& receiverId() const;

// Set/Get request ID
virtual void setRequestId(DtU32 id);
virtual DtU32 requestId() const;

// Set/Get the request status.
virtual void setRequestStatus(DtRequestStatus rStatus);
virtual DtRequestStatus requestStatus() const;

// Name of the interaction (Not nec. the ClassName from the FOM).
virtual const char* name() const;

public:

    // Create a DtActionResponseInteraction.
    // Caller is responsible for deletion.
    static DtInteraction* create();

    // Add (register)/Remove (deregister) function to be called when
    // interaction occurs.
    static void addCallback(DtExerciseConn* conn,
        DtActionResponseInteractionCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtActionResponseInteractionCb cb, void* usr);

protected:

    DtEntityIdentifier myOrigEnt;
    DtEntityIdentifier myReceiveEnt;
    DtU32 myRequestId;
    DtRequestStatus myRequestStatus;
};
#endif
```

hActRspObjDec.h

```
#if DtHLA

#include "interDecoder.h"

#include "chooseDefines.h"

// This can be used to declare parameter processor functions
// which are static members of a class.

#define DtDECLARE_ACTRESPOBJ_PARAM_DECODER(paramName) \
    DtDECLARE_PARAM_DECODER(DtActionResponseFromObjInter, paramName)

#define DtDEFINE_SIMPLE_ACTRESPOBJ_PARAM_DECODER( \
    paramName, netType, mutator) \
    DtDEFINE_SIMPLE_PARAM_DECODER( \
        DtActionResponseFromObjDecoder, DtActionResponseFromObjInter, \
        paramName, netType, mutator)

#define DtDEFINE_SIMPLE_ACTRESPOBJ_PARAM_DECODER_WITH_CAST( \
    paramName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_PARAM_DECODER_WITH_CAST( \
        DtActionResponseFromObjDecoder, DtActionResponseFromObjInter, \
        paramName, netType, mutator, castExpr)

// class DtActionResponseFromObjDecoder:
//
// Instances of DtActionResponseFromObjDecoder are used to decode
// ActionResponseFromObject Interactions from the network.

class DT_DLL_VL DtActionResponseFromObjDecoder : public DtInteractionDecoder
{
public:

    // constructor
    DtActionResponseFromObjDecoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtActionResponseFromObjDecoder();

    // copy constructor
    DtActionResponseFromObjDecoder(const DtActionResponseFromObjDecoder& orig);
```



```
// assignment operator
DtActionResponseFromObjDecoder& operator=(
    const DtActionResponseFromObjDecoder& orig);

protected:

    // Individual parameter decoders

    DtDECLARE_ACTRESPOBJ_PARAM_DECODER(ActionResult);
};
#endif
```

hActRspObjEnc.h

```
#if DtHLA

#include "hIntWEncDec.h"
#include "hostStructs.h"
#include "mathTypes.h"

#include "chooseDefines.h"

// class DtActionResponseFromObjInter:
//
// Instances of DtActionResponseFromObjInter are used to represent
// ActionResponseFromObject Interactions in the RPR FOM.

// ActionResponseFromObject interaction callback function signature.
typedef void (*DtActionResponseFromObjInterCb)
    (DtActionResponseFromObjInter* inter, void* usr);

class DT_DLL_VL DtActionResponseFromObjInter : public DtInteraction-
WithEncDec
{
public:

    // default constructor
    DtActionResponseFromObjInter();

    // destructor
    virtual ~DtActionResponseFromObjInter();

    // copy constructor
    DtActionResponseFromObjInter(const DtActionResponseFromObjInter&
orig);

    // assignment operator
    DtActionResponseFromObjInter& operator=(
        const DtActionResponseFromObjInter& orig);

    // Print the interaction's data.
    virtual void printData() const;

    // Set/Get action enum
    virtual void setActionResult(DtResponseFlag response);
    virtual DtResponseFlag actionResult() const;

    // Name of the interaction (Not nec. the ClassName from the FOM).
    virtual const char* name() const;

public:
```

```
// Create a DtActionResponseFromObjInter.  
// Caller is responsible for deletion.  
static DtInteraction* create();  
  
// Add (register)/Remove (deregister) function to be called when  
// interaction occurs.  
static void addCallback(DtExerciseConn* conn,  
    DtActionResponseFromObjInterCb cb, void* usr);  
static void removeCallback(DtExerciseConn* conn,  
    DtActionResponseFromObjInterCb cb, void* usr);  
  
protected:  
  
    DtResponseFlag myActionResult;  
  
protected:  
  
    static char* theClassName;  
};  
#endif
```

hActRspObjInt.h

```
#if DtHLA

#include "hIntWEncDec.h"
#include "hostStructs.h"
#include "mathTypes.h"

#include "chooseDefines.h"

// class DtActionResponseFromObjInter:
//
// Instances of DtActionResponseFromObjInter are used to represent
// ActionResponseFromObject Interactions in the RPR FOM.

// ActionResponseFromObject interaction callback function signature.
typedef void (*DtActionResponseFromObjInterCb)
    (DtActionResponseFromObjInter* inter, void* usr);

class DT_DLL_VL DtActionResponseFromObjInter : public DtInteraction-
WithEncDec
{
public:

    // default constructor
    DtActionResponseFromObjInter();

    // destructor
    virtual ~DtActionResponseFromObjInter();

    // copy constructor
    DtActionResponseFromObjInter(const DtActionResponseFromObjInter&
orig);

    // assignment operator
    DtActionResponseFromObjInter& operator=(
        const DtActionResponseFromObjInter& orig);

    // Print the interaction's data.
    virtual void printData() const;

    // Set/Get action enum
    virtual void setActionResult(DtResponseFlag response);
    virtual DtResponseFlag actionResult() const;

    // Name of the interaction (Not nec. the ClassName from the FOM).
    virtual const char* name() const;

public:
```

```
// Create a DtActionResponseFromObjInter.  
// Caller is responsible for deletion.  
static DtInteraction* create();  
  
// Add (register)/Remove (deregister) function to be called when  
// interaction occurs.  
static void addCallback(DtExerciseConn* conn,  
    DtActionResponseFromObjInterCb cb, void* usr);  
static void removeCallback(DtExerciseConn* conn,  
    DtActionResponseFromObjInterCb cb, void* usr);  
  
protected:  
  
    DtResponseFlag myActionResult;  
  
protected:  
  
    static char* theClassName;  
};  
#endif
```

hAggPub.h

```
#if DtHLA

#include "hObjPub.h"

// class DtAggregatePublisher:
//
// Instances of DtAggregatePublisher are used for local or source
// simulations
// of aggregate entities, (i.e. these entities are not dependent on
// data being
// produced elsewhere on the network for their existence). These
// entities send data to the network.

class DT_DLL_VL DtAggregatePublisher : public DtObjectPublisher
{
public:

    // Function prototype for functions that create
    // DtAggregateStateRepositories
    typedef DtAggregateStateRepository* (*DtStateRepCreator)();

public:

    // constructors
    DtAggregatePublisher(RTI::ObjectClassHandle h,
        DtExerciseConn* conn, const char* name = NULL);

    DtAggregatePublisher(const DtEntityType& type,
        DtExerciseConn* conn, const char* name = NULL);

    DtAggregatePublisher(const DtEntityType& type,
        DtExerciseConn *conn,
        DtDeadReckonTypes algorithm,
        DtForceType forceId,
        const char* name = NULL);

    // destructor
    virtual ~DtAggregatePublisher();

    // Returns a pointer to the AggregateStateRepository - through
    // which you can set or inspect current state information.
    virtual DtAggregateStateRepository* aggregateStateRep();

    // Synonym for aggregateStateRep()
    virtual DtAggregateStateRepository* asr();

public:
```

```
// Give us a function to be used for creating SRs.
static DtStateRepCreator setStateRepCreator(DtStateRepCreator
creator);

protected:

    // copy constructor - not implemented
    DtAggregatePublisher(const DtAggregatePublisher& orig);

    // assignment operator - not implemented
    DtAggregatePublisher& operator=(const DtAggregatePublisher&
orig);

    // Non-virtual since called from constructor. If you don't like
    // the default mapping between entity type and object class,
    // just pass the object class you want to the constructor.
    const char* objectClassNameFromType(const DtEntityType &type);

protected:

    static DtStateRepCreator theStateRepCreator;
};
#endif
```

hAppRadSigDec.h

```
#if DtHLA

#include "chooseDefines.h"
#include "interDecoder.h"

// This can be used to declare parameter processor functions
// which are static members of a class.

#define DtDECLARE_APP_SPECIFIC_RADIO_SIGNAL_PARAM_DECODER(param-
Name) \
    DtDECLARE_PARAM_DECODER(DtSignalInteraction, paramName)

#define
DtDEFINE_SIMPLE_APP_SPECIFIC_RADIO_SIGNAL_PARAM_DECODER(paramName,
\
    netType, mutator) \
    DtDEFINE_SIMPLE_PARAM_DECODER(DtAppSpecificRadioSignalDecoder, \
        DtSignalInteraction, paramName, netType, mutator)

#define
DtDEFINE_SIMPLE_APP_SPECIFIC_RADIO_SIGNAL_PARAM_DECODER_WITH_CAST(
\
    paramName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_PARAM_DECODER_WITH_CAST( \
        DtAppSpecificRadioSignalDecoder, DtSignalInteraction, \
        paramName, netType, mutator, castExpr)

// class DtAppSpecificRadioSignalDecoder:
//
// Instances of DtAppSpecificRadioSignalDecoder are

class DT_DLL_VL DtAppSpecificRadioSignalDecoder : public DtInterac-
tionDecoder
{
public:

    // default constructor
    DtAppSpecificRadioSignalDecoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtAppSpecificRadioSignalDecoder();

    // copy constructor
    DtAppSpecificRadioSignalDecoder(
        const DtAppSpecificRadioSignalDecoder& orig);
```



```
// assignment operator
DtAppSpecificRadioSignalDecoder& operator=(
    const DtAppSpecificRadioSignalDecoder& orig);

protected:

    DtDECLARE_APP_SPECIFIC_RADIO_SIGNAL_PARAM_DECODER(HostRadioID);
    DtDECLARE_APP_SPECIFIC_RADIO_SIGNAL_PARAM_DECODER(DataRate);
    DtDECLARE_APP_SPECIFIC_RADIO_SIGNAL_PARAM_DECODER(SignalDataLength);
    DtDECLARE_APP_SPECIFIC_RADIO_SIGNAL_PARAM_DECODER(SignalData);
    DtDECLARE_APP_SPECIFIC_RADIO_SIGNAL_PARAM_DECODER(TacticalDataLinkType);
    DtDECLARE_APP_SPECIFIC_RADIO_SIGNAL_PARAM_DECODER(TDLMessageCount);
    DtDECLARE_APP_SPECIFIC_RADIO_SIGNAL_PARAM_DECODER(UserProtocolID);
};
#endif    // DtHLA
```

hAppRadSigEnc.h

```
#if DtHLA

#include "chooseDefines.h"
#include "interEncoder.h"

#define DtDECLARE_APP_SPECIFIC_RADIO_SIGNAL_PARAM_ENCODER(param-
Name) \
    DtDECLARE_PARAM_ENCODER(DtSignalInteraction, paramName)

#define
DtDEFINE_SIMPLE_APP_SPECIFIC_RADIO_SIGNAL_PARAM_ENCODER(paramName,
\
    netType, inspector) \
    DtDEFINE_SIMPLE_PARAM_ENCODER(DtAppSpecificRadioSignalEncoder, \
    DtSignalInteraction, paramName, netType, inspector)

// class DtAppSpecificRadioSignalEncoder:
//
// Instances of DtAppSpecificRadioSignalEncoder are

class DT_DLL_VL DtAppSpecificRadioSignalEncoder : public DtInterac-
tionEncoder
{
public:

    // default constructor
    DtAppSpecificRadioSignalEncoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtAppSpecificRadioSignalEncoder();

    // copy constructor
    DtAppSpecificRadioSignalEncoder(
        const DtAppSpecificRadioSignalEncoder& orig);

    // assignment operator
    DtAppSpecificRadioSignalEncoder& operator=(
        const DtAppSpecificRadioSignalEncoder& orig);

protected:

    DtDECLARE_APP_SPECIFIC_RADIO_SIGNAL_PARAM_ENCODER(HostRadioID);
    DtDECLARE_APP_SPECIFIC_RADIO_SIGNAL_PARAM_ENCODER(DataRate);
    DtDECLARE_APP_SPECIFIC_RADIO_SIGNAL_PARAM_ENCODER(SignalDataL-
ength);
    DtDECLARE_APP_SPECIFIC_RADIO_SIGNAL_PARAM_ENCODER(SignalData);
```

```
    DtDECLARE_APP_SPECIFIC_RADIO_SIGNAL_PARAM_ENCODER(Tactical-  
DataLinkType);  
    DtDECLARE_APP_SPECIFIC_RADIO_SIGNAL_PARAM_ENCODER(TDLMessage-  
Count);  
    DtDECLARE_APP_SPECIFIC_RADIO_SIGNAL_PARAM_ENCODER(UserProto-  
colID);  
};  
#endif    //DtHLA
```

hashItem.h

```
#include "baseHashKey.h"
#include "vList.h"

// class DtHashItem:
//
// Instances of DtHashItem are used internally to a hashlist to
// associate
// a pointer to an item in a list with a key.

class DT_DLL_VLUTIL DtHashItem
{
public:

    // default constructor
    DtHashItem(const DtBaseHashKey& key, DtListItem* item);

    // destructor
    virtual ~DtHashItem();

    // copy constructor
    DtHashItem(const DtHashItem& orig);

    // assignment operator
    DtHashItem& operator=(const DtHashItem& orig);

    // get the key for the hashed item
    virtual const DtBaseHashKey& hashKey() const;

    // get the list item
    virtual DtListItem* listItem() const;

protected:

    DtBaseHashKey* myKey;
    DtListItem* myItem;
};
```

hashlist.h

```

#include "vList.h"
#include "baseHashKey.h"
#include "hashItem.h"

// Include obsolete old version for backwards compatibility.
#include "oldHashList.h"

// class DtHashlist:
//
// A DtHashlist is used to store associations between keys and arbitrary
// pieces of data passed as void*. Keys must be instances of a
// class derived
// from DtBaseHashKey (defined in baseHashKey.h). A subclass of
// DtHashKey
// defines a hashing function that hashes to an unsigned long, as
// well as a
// stringRep function that converts your key to a string representation. This
// allows DtHashlist to compare two DtHashKeys without knowing what
// type of
// DtHashKey subclass they actually are. You can add, remove and
// look up data
// by its DtHashKey.

// Not static member since g++ doesn't like to use static
// members as default arguments.
static const int dfltNumBuckets = 119;

class DT_DLL_VLUTIL DtHashlist
{
public:
    // Default constructor. Creates and uses a regular DtList to
    store the
    // list of data items that you add.
    DtHashlist(int numBuckets = dfltNumBuckets);

    // Constructor for a DtHashlist that uses an external DtList
    constructed by
    // the user.
    DtHashlist(DtList* list, int numBuckets = dfltNumBuckets);

    // Destructor
    virtual ~DtHashlist();

```

```
// Copy constructor - always uses an internal list, so if you want
your
// copy to use an external list, do not use the copy constructor.
Instead,
// use the standard with-external-list constructor to create an
empty hash
// list, then use setFromOther or the assignment operator to copy
the
// contents.
DtHashlist(const DtHashlist& orig);

// Assignment operator - Does not replace the DtHashlist's list.
Just
// empties the DtHashlist, and then sets its contents to those of
the
// original using setFromOther.
DtHashlist& operator=(const DtHashlist& orig);

// Add something to the list. The actual pointer is stored, so
you should
// make sure the pointer remains valid until after you remove the
data from
// the hash list. At any given time, only one data item may be
associated
// with a particular key. If add is called with a key with which
an item
// is already associated, the old item is replaced with the new,
and the
// old item is returned in the optional oldData argument for
deletion by
// the user. Otherwise, oldData will be set to NULL.
virtual DtListItem* add(const DtBaseHashKey& key,
    void* data, void** oldData = NULL);

// Remove something from the list. Returns the data that was
associated
// with the key, so that you can cast it to the right type and
delete it.
virtual void* remove(const DtBaseHashKey& key);

// Lookup something in the list, return data pointer.
virtual void* lookup(const DtBaseHashKey& key) const;
virtual void* operator[] (const DtBaseHashKey& key) const;

// Returns a DtList of void*'s which are the data that you have
// added to the hashlist (independent of hash keys).
virtual const DtList* list() const;

// Prints the data in the hashlist.
virtual void print() const;
```

```
// Returns a DtList of DtHashKey*'s representing the set of keys
with which
// data is currently associated.
virtual const DtList* keys() const;

// Remove all keys and data from the list.
virtual void empty();

// Sets the contents of this hashlist to match those of other.
Data
// pointers that have been added to orig are copied, but the items
that
// they point to are not.
virtual void setFromOther(const DtHashlist& orig);

protected:

// Lookup something in the list, return DtListItem
virtual DtListItem* itemLookup(const DtBaseHashKey& key) const;

// Determine the bucket from the hashkey
virtual DtList& bucketFromHashKey(const DtBaseHashKey& key)
const;

// Lookup an item in a bucket
virtual DtListItem* bucketItemLookup(const DtBaseHashKey& key)
const;

protected:

int myNumBuckets;
const DtBoolean myListIsMine;

// Regular DtList containing all of the items added to the
hashlist,
// without regard to key information.
DtList* myList;

// Array of buckets (DtLists of DtHashItems).
DtList* myBuckets;

// DtList of keys (DtBaseHashKeys).
DtList myKeys;

protected:

static int theNumBucketsDefault;

};
```

```
typedef DtHashlist DtHashList;
```


hAttChgReqDec.h

```

#if DtHLA

#include "interDecoder.h"

// This can be used to declare parameter processor functions
// which are static members of a class.

#define DtDECLARE_ATTCHGREQ_PARAM_DECODER(paramName) \
    DtDECLARE_PARAM_DECODER(DtAttributeChangeReqInter, paramName)

#define DtDEFINE_SIMPLE_ATTCHGREQ_PARAM_DECODER( \
    paramName, netType, mutator) \
    DtDEFINE_SIMPLE_PARAM_DECODER( \
        DtAttributeChangeReqDec, DtAttributeChangeReqInter, \
        paramName, netType, mutator)

// class DtAttributeChangeReqDec:
//
// Instances of DtAttributeChangeReqDec are used to decode
//CreateObjectRequest Interactions from the network.

class DT_DLL_VL DtAttributeChangeReqDec : public DtInteractionDe-
coder
{
public:

    // constructor
    DtAttributeChangeReqDec(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtAttributeChangeReqDec();

    // copy constructor
    DtAttributeChangeReqDec(const DtAttributeChangeReqDec& orig);

    // assignment operator
    DtAttributeChangeReqDec& operator=(const DtAttributeChangeReqDec&
orig);

protected:

    // Individual parameter decoders

    DtDECLARE_ATTCHGREQ_PARAM_DECODER(ObjectIDs);
    DtDECLARE_ATTCHGREQ_PARAM_DECODER(AttributeValueSet);
};
#endif

```

hAttChgReqEnc.h

```
#if DtHLA

#include "interEncoder.h"

#define DtDECLARE_ATTCHGREQ_PARAM_ENCODER(paramName) \
    DtDECLARE_PARAM_ENCODER(DtAttributeChangeReqInter, paramName)

#define DtDEFINE_SIMPLE_ATTCHGREQ_PARAM_ENCODER(paramName, netType, \
    inspector) \
    DtDEFINE_SIMPLE_PARAM_ENCODER(DtAttributeChangeReqEnc, \
        DtAttributeChangeReqInter, paramName, netType, inspector)

// class DtAttributeChangeReqEnc:
//
// DtAttributeChangeReqEnc is a DtInteractionEncoder that registers
// parameter encoders for all of the AttributeChangeRequest interac-
// tion
// parameters.

class DT_DLL_VL DtAttributeChangeReqEnc : public DtInteractionEn-
coder
{
public:

    DtAttributeChangeReqEnc(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtAttributeChangeReqEnc();

    // copy constructor
    DtAttributeChangeReqEnc(const DtAttributeChangeReqEnc& orig);

    // assignment operator
    DtAttributeChangeReqEnc& operator=(const DtAttributeChangeReqEnc&
orig);

protected:

    // Individual parameter encoders

    DtDECLARE_ATTCHGREQ_PARAM_ENCODER(ObjectIDs);
    DtDECLARE_ATTCHGREQ_PARAM_ENCODER(AttributeValueSet);
};
#endif
```

hAttChgReqInt.h

```

#if DtHLA

#include "hIntWEncDec.h"
#include "hostStructs.h"
#include "mathTypes.h"
#include "objDesList.h"
#include "datumRecSet.h"

#include "chooseDefines.h"

// class DtAttributeChangeReqInter:
//
// Instances of DtAttributeChangeReqInter are used to represent
// AttributeChangeRequest Interactions in the RPR FOM.

// AttributeChangeRequest interaction callback function signature.
typedef void (*DtAttributeChangeReqInterCb)(DtAttributeChangeRe-
qInter* inter,
    void* usr);

class DT_DLL_VL DtAttributeChangeReqInter : public DtInteraction-
WithEncDec
{
public:

    // default constructor
    DtAttributeChangeReqInter();

    // destructor
    virtual ~DtAttributeChangeReqInter();

    // copy constructor
    DtAttributeChangeReqInter(const DtAttributeChangeReqInter& orig);

    // assignment operator
    DtAttributeChangeReqInter& operator=(const DtAttributeChangeRe-
qInter& orig);

    // Print the interaction's data.
    virtual void printData() const;

    // Return the Object ID list
    virtual const DtGlobalObjectDesignatorList& objectIDs() const;
    virtual DtGlobalObjectDesignatorList* objectIDs();

    // Set/Get number of IDs.
    virtual void setNumIds(DtU8 numIds);

```

```
virtual DtU8 numIds() const;

// Set/Get the nth ObjectId.
virtual void setNthObjectId(DtInt32 idIndex,
    const DtGlobalObjectDesignator& id);
virtual const DtGlobalObjectDesignator& nthObjectId(DtInt32
idIndex) const;

// Set/Get the number of attribute/value pairs
virtual void setNumPairs(DtU32 pairs);
virtual DtU32 numPairs() const;

virtual void setAvPairSet(
    const RTI::AttributeHandleValuePairSet& atts);
virtual const RTI::AttributeHandleValuePairSet& avPairSet()
const;

// Set/Get the attribute handle given an index
virtual void setAttributeHandle(DtU32 index, DtU32 handle);
virtual DtU32 attributeHandle(DtU32 index) const;

// Set/Get the attribute value length given and index
virtual void setAttributeValueLength(DtU32 index, DtU32 length);
virtual DtU32 attributeValueLength(DtU32 index) const;

virtual void setAttributeValue(DtU32 index, const char* usr);
virtual const char* attributeValue(DtU32 index) const;

// Return the DtDatumRecordSet which an InteractionWithData
// uses to store its fixed and var records. In general,
// the wrapper functions above should be used rather than going
directly
// through the DtDatumRecordSet, which is necessary only when
doing
// encoding and decoding of network representations.
virtual DtDatumRecordSet* datumRecSet();
virtual const DtDatumRecordSet* datumRecSet() const;

// Name of the interaction (Not nec. the ClassName from the FOM).
virtual const char* name() const;

public:

// Create a DtAttributeChangeReqInter.
// Caller is responsible for deletion.
static DtInteraction* create();

// Add (register)/Remove (deregister) function to be called when
// interaction occurs.
static void addCallback(DtExerciseConn* conn,
```

```
        DtAttributeChangeReqInterCb cb, void* usr);
static void removeCallback(DtExerciseConn* conn,
        DtAttributeChangeReqInterCb cb, void* usr);

protected:
    DtGlobalObjectDesignatorList myObjectList;
    DtDatumRecordSet myRecordSet;

protected:

    static char* theClassName;
};
#endif
```

hAttChgResDec.h

```
#if DtHLA

#include "interDecoder.h"

// This can be used to declare parameter processor functions
// which are static members of a class.

#define DtDECLARE_ATTCHGRES_PARAM_DECODER(paramName) \
    DtDECLARE_PARAM_DECODER(DtAttributeChangeResInter, paramName)

#define DtDEFINE_SIMPLE_ATTCHGRES_PARAM_DECODER( \
    paramName, netType, mutator) \
    DtDEFINE_SIMPLE_PARAM_DECODER( \
        DtAttributeChangeResDec, DtAttributeChangeResInter, \
        paramName, netType, mutator)

#define DtDEFINE_SIMPLE_ATTCHGRES_PARAM_DECODER_WITH_CAST( \
    paramName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_PARAM_DECODER_WITH_CAST( \
        DtAttributeChangeResDec, DtAttributeChangeResInter, \
        paramName, netType, mutator, castExpr)

// class DtAttributeChangeResDec:
//
// Instances of DtAttributeChangeResDec are used to decode
//AttributeChangeResult Interactions from the network.

class DT_DLL_VL DtAttributeChangeResDec : public DtInteractionDe-
coder
{
public:

    // constructor
    DtAttributeChangeResDec(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtAttributeChangeResDec();

    // copy constructor
    DtAttributeChangeResDec(const DtAttributeChangeResDec& orig);

    // assignment operator
    DtAttributeChangeResDec& operator=(
        const DtAttributeChangeResDec& orig);

protected:
```

```
// Individual parameter decoders

DtDECLARE_ATTCHGRES_PARAM_DECODER(ObjectID);
DtDECLARE_ATTCHGRES_PARAM_DECODER(AttributeChangeResult);
DtDECLARE_ATTCHGRES_PARAM_DECODER(AttributeValueSet);

// RPR FOM 0.7 compatibility
DtDECLARE_ATTCHGRES_PARAM_DECODER(AttributeChangeResultPre08);
};
#endif
```

hAttChgResEnc.h

```
#if DtHLA

#include "interEncoder.h"

#define DtDECLARE_ATTCHGRES_PARAM_ENCODER(paramName) \
    DtDECLARE_PARAM_ENCODER(DtAttributeChangeResInter, paramName)

#define DtDEFINE_SIMPLE_ATTCHGRES_PARAM_ENCODER(paramName, netType, \
    inspector) \
    DtDEFINE_SIMPLE_PARAM_ENCODER(DtAttributeChangeResEnc, \
        DtAttributeChangeResInter, paramName, netType, inspector)

// class DtAttributeChangeResEnc:
//
// DtAttributeChangeResEnc is a DtInteractionEncoder that registers
// parameter encoders for all of the AttributeChangeResult interac-
// tion
// parameters.

class DT_DLL_VL DtAttributeChangeResEnc : public DtInteractionEn-
coder
{
public:

    DtAttributeChangeResEnc(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtAttributeChangeResEnc();

    // copy constructor
    DtAttributeChangeResEnc(const DtAttributeChangeResEnc& orig);

    // assignment operator
    DtAttributeChangeResEnc& operator=(const DtAttributeChangeResEnc&
orig);

protected:

    // Individual parameter encoders

    DtDECLARE_ATTCHGRES_PARAM_ENCODER(ObjectID);
    DtDECLARE_ATTCHGRES_PARAM_ENCODER(AttributeChangeResult);
    DtDECLARE_ATTCHGRES_PARAM_ENCODER(AttributeValueSet);

    // RPR FOM 0.7 compatibility
    DtDECLARE_ATTCHGRES_PARAM_ENCODER(AttributeChangeResultPre08);

}
```



```
};  
#endif
```

hAttChgResInt.h

```
#if DtHLA

#include "hIntWEncDec.h"
#include "hostStructs.h"
#include "mathTypes.h"
#include "datumRecSet.h"

#include "chooseDefines.h"

// class DtAttributeChangeResInter:
//
// Instances of DtAttributeChangeResInter are used to represent
// AttributeChangeResult Interactions in the RPR FOM.

// AttributeChangeResult interaction callback function signature.
typedef void (*DtAttributeChangeResInterCb)(DtAttributeChangeRes-
Inter* inter,
void* usr);

class DT_DLL_VL DtAttributeChangeResInter : public DtInteraction-
WithEncDec
{
public:

    // default constructor
    DtAttributeChangeResInter();

    // destructor
    virtual ~DtAttributeChangeResInter();

    // copy constructor
    DtAttributeChangeResInter(const DtAttributeChangeResInter& orig);

    // assignment operator
    DtAttributeChangeResInter& operator=(const DtAttributeChangeRes-
Inter& orig);

    // Print the interaction's data.
    virtual void printData() const;

    // Set/Get the objectID.
    virtual void setObjectId(const DtGlobalObjectDesignator &id);
    virtual const DtGlobalObjectDesignator &objectId() const;

    // Set/Get the number of attribute/value pairs
    virtual void setNumPairs(DtU32 pairs);
    virtual DtU32 numPairs() const;
```

```

virtual void setAvPairSet(
    const RTI::AttributeHandleValuePairSet& atts);
virtual const RTI::AttributeHandleValuePairSet& avPairSet()
const;

// Set/Get the attribute handle given an index
virtual void setAttributeHandle(DtU32 index, DtU32 handle);
virtual DtU32 attributeHandle(DtU32 index) const;

// Set/Get the attribute value length given and index
virtual void setAttributeValueLength(DtU32 index, DtU32 length);
virtual DtU32 attributeValueLength(DtU32 index) const;

virtual void setAttributeValue(DtU32 index, const char* usr);
virtual const char* attributeValue(DtU32 index) const;

// Return the DtDatumRecordSet which an InteractionWithData
// uses to store its fixed and var records. In general,
// the wrapper functions above should be used rather than going
directly
// through the DtDatumRecordSet, which is necessary only when
doing
// encoding and decoding of network representations.
virtual DtDatumRecordSet* datumRecSet();
virtual const DtDatumRecordSet* datumRecSet() const;

// Set/Get the AttributeChangeResult type.
virtual void setAttributeChangeResult(DtAttributeChangeResult
result);
virtual DtAttributeChangeResult attributeChangeResult() const;

// Name of the interaction (Not nec. the ClassName from the FOM).
virtual const char* name() const;

public:

    // Create a DtAttributeChangeResInter.
    // Caller is responsible for deletion.
    static DtInteraction* create();

    // Add (register)/Remove (deregister) function to be called when
    // interaction occurs.
    static void addCallback(DtExerciseConn* conn,
        DtAttributeChangeResInterCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtAttributeChangeResInterCb cb, void* usr);

protected:
    DtGlobalObjectDesignator myObjectId;

```

```
DtDatumRecordSet myRecordSet;  
DtAttributeChangeResult myAttributeChangeResult;  
  
protected:  
  
    static char* theClassName;  
};  
#endif
```

hCollideDec.h

```

#if DtHLA

#include "interDecoder.h"

#include "chooseDefines.h"

// This can be used to declare parameter processor functions
// which are static members of a class.

#define DtDECLARE_COLLISION_PARAM_DECODER(paramName) \
    DtDECLARE_PARAM_DECODER(DtCollisionInteraction, paramName)

#define DtDEFINE_SIMPLE_COLLISION_PARAM_DECODER(paramName, netType, \
mutator) \
    DtDEFINE_SIMPLE_PARAM_DECODER(DtCollisionDecoder, DtCollisionIn- \
teraction, \
                                paramName, netType, mutator)

#define DtDEFINE_SIMPLE_COLLISION_PARAM_DECODER_WITH_CAST( \
    paramName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_PARAM_DECODER_WITH_CAST( \
        DtCollisionDecoder, DtCollisionInteraction, \
        paramName, netType, mutator, castExpr)

// class DtCollisionDecoder:
//
// Instances of DtCollisionDecoder are used to decode Collision
// Interactions from the network.

class DT_DLL_VL DtCollisionDecoder : public DtInteractionDecoder
{
public:

    // constructor
    // Adds collision parameter decoders.
    DtCollisionDecoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtCollisionDecoder();

    // copy constructor
    DtCollisionDecoder(const DtCollisionDecoder& orig);

    // assignment operator
    DtCollisionDecoder& operator=(const DtCollisionDecoder& orig);

```

protected:

```
DtDECLARE_COLLISION_PARAM_DECODER(CollidingObjectID);
#if !DtrPR016
    DtDECLARE_COLLISION_PARAM_DECODER(IssuingObjectMass);
    DtDECLARE_COLLISION_PARAM_DECODER(IssuingObjectVelocity);
#endif
    DtDECLARE_COLLISION_PARAM_DECODER(CollisionType);
    DtDECLARE_COLLISION_PARAM_DECODER(CollisionLocation);
    DtDECLARE_COLLISION_PARAM_DECODER(EventID);
    DtDECLARE_COLLISION_PARAM_DECODER(IssuingObjectID);
#if DtrPR016
    DtDECLARE_COLLISION_PARAM_DECODER(CollidingObjectMass);
    DtDECLARE_COLLISION_PARAM_DECODER(CollidingObjectVelocity);
#endif
};
#endif
```

hCollideEnc.h

```

#if DtHLA

#include "interEncoder.h"

#define DtDECLARE_COLLISION_PARAM_ENCODER(paramName) \
    DtDECLARE_PARAM_ENCODER(DtCollisionInteraction, paramName)

#define DtDEFINE_SIMPLE_COLLISION_PARAM_ENCODER(paramName, netType, \
    inspector) \
    DtDEFINE_SIMPLE_PARAM_ENCODER(DtCollisionEncoder, DtCollisionIn- \
        teraction, \
                                paramName, netType, \
                                inspector)

// class DtCollisionEncoder:
//
// DtCollisionEncoder is a DtInteractionEncoder that registers
// parameter encoders for all of the COLLISION interaction param-
// eters.

class DT_DLL_VL DtCollisionEncoder : public DtInteractionEncoder
{
public:

    DtCollisionEncoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtCollisionEncoder();

    // copy constructor
    DtCollisionEncoder(const DtCollisionEncoder& orig);

    // assignment operator
    DtCollisionEncoder& operator=(const DtCollisionEncoder& orig);

protected:

    DtDECLARE_COLLISION_PARAM_ENCODER(CollidingObjectID);
#if !DtRPR016
    DtDECLARE_COLLISION_PARAM_ENCODER(IssuingObjectMass);
    DtDECLARE_COLLISION_PARAM_ENCODER(IssuingObjectVelocity);
#endif
    DtDECLARE_COLLISION_PARAM_ENCODER(CollisionType);
    DtDECLARE_COLLISION_PARAM_ENCODER(CollisionLocation);
    DtDECLARE_COLLISION_PARAM_ENCODER(EventID);
    DtDECLARE_COLLISION_PARAM_ENCODER(IssuingObjectID);

```

```
#if DtrPR016
    DtDECLARE_COLLISION_PARAM_ENCODER(CollidingObjectMass);
    DtDECLARE_COLLISION_PARAM_ENCODER(CollidingObjectVelocity);
#endif
};
#endif
```


hCollideInter.h

```

#if DtHLA

#include "hIntWEncDec.h"
#include "hostStructs.h"
#include "mathTypes.h"

#include "chooseDefines.h"

// class DtCollisionInteraction:
//
// Instances of DtCollisionInteraction are used to represent
// Collision Interactions in the RPR FOM.

// Collision interaction callback function signature.
typedef void (*DtCollisionInteractionCb)
    (DtCollisionInteraction* inter, void* usr);

class DT_DLL_VL DtCollisionInteraction : public DtInteractionWith-
EncDec
{
public:

    // default constructor
    DtCollisionInteraction();

    // destructor
    virtual ~DtCollisionInteraction();

    // copy constructor
    DtCollisionInteraction(const DtCollisionInteraction& orig);

    // assignment operator
    DtCollisionInteraction& operator=(const DtCollisionInteraction&
orig);

    // Print the interaction's data.
    virtual void printData() const;

    // Set/Get the issuer identifier.
    virtual void setIssueId(const DtGlobalObjectDesignator& id);
    virtual const DtGlobalObjectDesignator& issueId() const;

    // Set/Get the collide identifier.
    virtual void setCollideId(const DtGlobalObjectDesignator& id);
    virtual const DtGlobalObjectDesignator& collideId() const;

    // Set/Get the collision event id.

```

```
virtual void setEventId(const DtEventID& id);
virtual const DtEventID& eventId() const;

// Set/Get the collision type.
virtual void setCollisionType(DtCollisionType t);
virtual DtCollisionType collisionType() const;

// Set/Get the velocity.
virtual void setVelocity(DtConstVector veloc);
virtual DtConstVector velocity() const;

// Set/Get the mass involved.
virtual void setMass(DtFloat32 mass);
virtual DtFloat32 mass() const;

// Set/Get the entity coordinates.
virtual void setEntityCoords(DtConstVector coords);
virtual DtConstVector entityCoords() const;

// Name of the interaction (Not nec. the ClassName from the FOM).
virtual const char* name() const;

public:

    // Create a DtCollisionInteraction.
    // Caller is responsible for deletion.
    static DtInteraction* create();

    // Add (register)/Remove (deregister) function to be called when
    // interaction occurs.
    static void addCallback(DtExerciseConn* conn,
        DtCollisionInteractionCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtCollisionInteractionCb cb, void* usr);

protected:

    DtGlobalObjectDesignator myIssueId;
    DtGlobalObjectDesignator myCollideId;
    DtEventID myEventId;
    DtCollisionType myCollisionType;
    DtVector myVelocity;
    DtFloat32 myMass;
    DtVector myEntityCoords;
};
#endif
```

hCommentDec.h

```

#ifdef DtHLA

#include "interDecoder.h"

// This can be used to declare parameter processor functions
// which are static members of a class.

#define DtDECLARE_COMMENT_PARAM_DECODER(paramName) \
    DtDECLARE_PARAM_DECODER(DtCommentInteraction, paramName)

#define DtDEFINE_SIMPLE_COMMENT_PARAM_DECODER( \
    paramName, netType, mutator) \
    DtDEFINE_SIMPLE_PARAM_DECODER( \
        DtCommentDecoder, DtCommentInteraction, \
        paramName, netType, mutator)

// class DtCommentDecoder:
//
// Instances of DtCommentDecoder are used to decode Comment
// Interactions from the network.

class DT_DLL_VL DtCommentDecoder : public DtInteractionDecoder
{
public:

    // constructor
    DtCommentDecoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtCommentDecoder();

    // copy constructor
    DtCommentDecoder(const DtCommentDecoder& orig);

    // assignment operator
    DtCommentDecoder& operator=(const DtCommentDecoder& orig);

protected:

    // Individual parameter decoders

    DtDECLARE_COMMENT_PARAM_DECODER(OriginatingEntity);
    DtDECLARE_COMMENT_PARAM_DECODER(ReceivingEntity);
    DtDECLARE_COMMENT_PARAM_DECODER(VariableDatumSet);
};
#endif

```

hCommentEnc.h

```
#if DtHLA

#include "interEncoder.h"

#define DtDECLARE_COMMENT_PARAM_ENCODER(paramName) \
    DtDECLARE_PARAM_ENCODER(DtCommentInteraction, paramName)

#define DtDEFINE_SIMPLE_COMMENT_PARAM_ENCODER(paramName, netType, \
    inspector) \
    DtDEFINE_SIMPLE_PARAM_ENCODER(DtCommentEncoder, \
        DtCommentInteraction, paramName, netType, inspector)

// class DtCommentEncoder:
//
// DtCommentEncoder is a DtInteractionEncoder that registers
// parameter encoders for all of the Comment interaction parameters.

class DT_DLL_VL DtCommentEncoder : public DtInteractionEncoder
{
public:

    DtCommentEncoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtCommentEncoder();

    // copy constructor
    DtCommentEncoder(const DtCommentEncoder& orig);

    // assignment operator
    DtCommentEncoder& operator=(const DtCommentEncoder& orig);

protected:

    // Individual parameter encoders

    DtDECLARE_COMMENT_PARAM_ENCODER(OriginatingEntity);
    DtDECLARE_COMMENT_PARAM_ENCODER(ReceivingEntity);
    DtDECLARE_COMMENT_PARAM_ENCODER(VariableDatumSet);
};
#endif
```

hCommentInter.h

```

#if DtHLA

#include "hInterWData.h"
#include "hostStructs.h"
#include "mathTypes.h"

#include "chooseDefines.h"

// class DtCommentInteraction:
//
// Instances of DtCommentInteraction are used to represent
// Comment Interactions in the RPR FOM. The variable
// datum record is accessed through DtInteractionWithData member
// functions.

// Comment interaction callback function signature.
typedef void (*DtCommentInteractionCb)
    (DtCommentInteraction* inter, void* usr);

class DT_DLL_VL DtCommentInteraction : public DtInteractionWithData
{
public:

    // default constructor
    DtCommentInteraction();

    // destructor
    virtual ~DtCommentInteraction();

    // copy constructor
    DtCommentInteraction(const DtCommentInteraction& orig);

    // assignment operator
    DtCommentInteraction& operator=(
        const DtCommentInteraction& orig);

    // Print the interaction's data.
    virtual void printData() const;

    // 'Set/Get sender ID
    virtual void setSenderId(const DtEntityIdentifier& id);
    virtual const DtEntityIdentifier& senderId() const;

    // Set/Get receiver ID
    virtual void setReceiverId(const DtEntityIdentifier& id);
    virtual const DtEntityIdentifier& receiverId() const;

```

```
// Set/Get the comment; warning: inspector returns a readable
text string.
// Use DtInteractionWithData member functions to get non-string
comments.
virtual void setComment(const char *message, int size);
virtual const char* comment() const;

// Name of the interaction (Not nec. the ClassName from the FOM).
virtual const char* name() const;

public:

    // Create a DtCommentInteraction.
    // Caller is responsible for deletion.
    static DtInteraction* create();

    // Add (register)/Remove (deregister) function to be called when
    // interaction occurs.
    static void addCallback(DtExerciseConn* conn,
        DtCommentInteractionCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtCommentInteractionCb cb, void* usr);

protected:

    DtEntityIdentifier myOrigEnt;
    DtEntityIdentifier myReceiveEnt;
};
#endif
```

hCreatEntDec.h

```

#ifdef DtHLA

#include "interDecoder.h"

// This can be used to declare parameter processor functions
// which are static members of a class.

#define DtDECLARE_CREATENT_PARAM_DECODER(paramName) \
    DtDECLARE_PARAM_DECODER(DtCreateEntityInteraction, paramName)

#define DtDEFINE_SIMPLE_CREATENT_PARAM_DECODER( \
    paramName, netType, mutator) \
    DtDEFINE_SIMPLE_PARAM_DECODER( \
        DtCreateEntityDecoder, DtCreateEntityInteraction, \
        paramName, netType, mutator)

#define DtDEFINE_SIMPLE_CREATENT_PARAM_DECODER_WITH_CAST( \
    paramName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_PARAM_DECODER_WITH_CAST( \
        DtCreateEntityDecoder, DtCreateEntityInteraction, \
        paramName, netType, mutator, castExpr)

// class DtCreateEntityDecoder:
//
// Instances of DtCreateEntityDecoder are used to decodeCreateEn-
// tity
// Interactions from the network.

class DT_DLL_VL DtCreateEntityDecoder : public DtInteractionDecoder
{
public:

    // constructor
    DtCreateEntityDecoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtCreateEntityDecoder();

    // copy constructor
    DtCreateEntityDecoder(const DtCreateEntityDecoder& orig);

    // assignment operator
    DtCreateEntityDecoder& operator=(const DtCreateEntityDecoder&
orig);

protected:

```

```
// Individual parameter decoders

DtDECLARE_CREATENT_PARAM_DECODER(OriginatingEntity);
DtDECLARE_CREATENT_PARAM_DECODER(ReceivingEntity);
DtDECLARE_CREATENT_PARAM_DECODER(RequestID);
};
#endif
```


hCreatEntEnc.h

```

#if DtHLA

#include "interEncoder.h"

#define DtDECLARE_CREATENT_PARAM_ENCODER(paramName) \
    DtDECLARE_PARAM_ENCODER(DtCreateEntityInteraction, paramName)

#define DtDEFINE_SIMPLE_CREATENT_PARAM_ENCODER(paramName, netType, \
    inspector) \
    DtDEFINE_SIMPLE_PARAM_ENCODER(DtCreateEntityEncoder, \
        DtCreateEntityInteraction, paramName, netType, inspector)

// class DtCreateEntityEncoder:
//
// DtCreateEntityEncoder is a DtInteractionEncoder that registers
// parameter encoders for all of the CreateEntity interaction param-
// eters.

class DT_DLL_VL DtCreateEntityEncoder : public DtInteractionEncoder
{
public:

    DtCreateEntityEncoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtCreateEntityEncoder();

    // copy constructor
    DtCreateEntityEncoder(const DtCreateEntityEncoder& orig);

    // assignment operator
    DtCreateEntityEncoder& operator=(const DtCreateEntityEncoder&
orig);

protected:

    // Individual parameter encoders

    DtDECLARE_CREATENT_PARAM_ENCODER(OriginatingEntity);
    DtDECLARE_CREATENT_PARAM_ENCODER(ReceivingEntity);
    DtDECLARE_CREATENT_PARAM_ENCODER(RequestID);
};
#endif

```

hCreatEntInt.h

```
#if DtHLA

#include "hIntWEncDec.h"
#include "hostStructs.h"
#include "mathTypes.h"

#include "chooseDefines.h"

// class DtCreateEntityInteraction:
//
// Instances of DtCreateEntityInteraction are used to represent
// CreateEntity Interactions in the RPR FOM.

// CreateEntity interaction callback function signature.
typedef void (*DtCreateEntityInteractionCb)(
    DtCreateEntityInteraction* inter, void* usr);

class DT_DLL_VL DtCreateEntityInteraction : public DtInteraction-
WithEncDec
{
public:

    // default constructor
    DtCreateEntityInteraction();

    // destructor
    virtual ~DtCreateEntityInteraction();

    // copy constructor
    DtCreateEntityInteraction(const DtCreateEntityInteraction& orig);

    // assignment operator
    DtCreateEntityInteraction& operator=(const DtCreateEntityInterac-
tion& orig);

    // Print the interaction's data.
    virtual void printData() const;

    // Set/Get sender ID
    virtual void setSenderId(const DtEntityIdentifier& id);
    virtual const DtEntityIdentifier& senderId() const;

    // Set/Get receiver ID
    virtual void setReceiverId(const DtEntityIdentifier& id);
    virtual const DtEntityIdentifier& receiverId() const;

    // Set/Get request ID
```

```
virtual void setRequestId(int id);
virtual int requestId() const;

// Name of the interaction (Not nec. the ClassName from the FOM).
virtual const char* name() const;

public:

    // Create a DtCreateEntityInteraction.
    // Caller is responsible for deletion.
    static DtInteraction* create();

    // Add (register)/Remove (deregister) function to be called when
    // interaction occurs.
    static void addCallback(DtExerciseConn* conn,
        DtCreateEntityInteractionCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtCreateEntityInteractionCb cb, void* usr);

protected:

    DtEntityIdentifier myOrigEnt;
    DtEntityIdentifier myReceiveEnt;
    int myRequestId;

protected:

    static char* theClassName;
};
#endif
```

hCrtObjReqDec.h

```
#if DtHLA

#include "interDecoder.h"

// This can be used to declare parameter processor functions
// which are static members of a class.

#define DtDECLARE_CRTOBJREQ_PARAM_DECODER(paramName) \
    DtDECLARE_PARAM_DECODER(DtCreateObjReqInter, paramName)

#define DtDEFINE_SIMPLE_CRTOBJREQ_PARAM_DECODER( \
    paramName, netType, mutator) \
    DtDEFINE_SIMPLE_PARAM_DECODER( \
        DtCreateObjReqDecoder, DtCreateObjReqInter, \
        paramName, netType, mutator)

// class DtCreateObjReqDecoder:
//
// Instances of DtCreateObjReqDecoder are used to decode
// CreateObjectRequest Interactions from the network.

class DT_DLL_VL DtCreateObjReqDecoder : public DtInteractionDecoder
{
public:

    // constructor
    DtCreateObjReqDecoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtCreateObjReqDecoder();

    // copy constructor
    DtCreateObjReqDecoder(const DtCreateObjReqDecoder& orig);

    // assignment operator
    DtCreateObjReqDecoder& operator=(const DtCreateObjReqDecoder&
orig);

protected:

    // Individual parameter decoders

    DtDECLARE_CRTOBJREQ_PARAM_DECODER(ObjectClass);
    DtDECLARE_CRTOBJREQ_PARAM_DECODER(AttributeValueSet);
    DtDECLARE_CRTOBJREQ_PARAM_DECODER(RequestID);
};
#endif
```

hCrtObjReqEnc.h

```

#if DtHLA

#include "interEncoder.h"

#define DtDECLARE_CRTOBJREQ_PARAM_ENCODER(paramName) \
    DtDECLARE_PARAM_ENCODER(DtCreateObjReqInter, paramName)

#define DtDEFINE_SIMPLE_CRTOBJREQ_PARAM_ENCODER(paramName, netType, \
    inspector) \
    DtDEFINE_SIMPLE_PARAM_ENCODER(DtCreateObjReqEncoder, \
        DtCreateObjReqInter, paramName, netType, inspector)

// class DtCreateObjReqEncoder:
//
// DtCreateObjReqEncoder is a DtInteractionEncoder that registers
// parameter encoders for all of the CreateObjectRequest interaction
// parameters.

class DT_DLL_VL DtCreateObjReqEncoder : public DtInteractionEncoder
{
public:

    DtCreateObjReqEncoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtCreateObjReqEncoder();

    // copy constructor
    DtCreateObjReqEncoder(const DtCreateObjReqEncoder& orig);

    // assignment operator
    DtCreateObjReqEncoder& operator=(const DtCreateObjReqEncoder&
orig);

protected:

    // Individual parameter encoders

    DtDECLARE_CRTOBJREQ_PARAM_ENCODER(ObjectClass);
    DtDECLARE_CRTOBJREQ_PARAM_ENCODER(AttributeValueSet);
    DtDECLARE_CRTOBJREQ_PARAM_ENCODER(RequestID);
};
#endif

```

hCrtObjReqInt.h

```
#if DtHLA

#include "hInterWData.h"
#include "hostStructs.h"
#include "mathTypes.h"

#include "chooseDefines.h"

// class DtCreateObjReqInter:
//
// Instances of DtCreateObjReqInter are used to represent
// CreateObjectRequest Interactions in the RPR FOM.

// CreateObjectRequest interaction callback function signature.
typedef void (*DtCreateObjReqInterCb)(DtCreateObjReqInter* inter,
void* usr);

class DT_DLL_VL DtCreateObjReqInter : public DtInteractionWithEncDec
{
public:

    // default constructor
    DtCreateObjReqInter();

    // destructor
    virtual ~DtCreateObjReqInter();

    // copy constructor
    DtCreateObjReqInter(const DtCreateObjReqInter& orig);

    // assignment operator
    DtCreateObjReqInter& operator=(const DtCreateObjReqInter& orig);

    // Print the interaction's data.
    virtual void printData() const;

    // Set/Get object class
    virtual void setObjectClass(DtU32 id);
    virtual DtU32 objectClass() const;

    // Set/Get the number of attribute/value pairs
    virtual void setNumPairs(DtU32 pairs);
    virtual DtU32 numPairs() const;

    virtual void setAvPairSet(
        const RTI::AttributeHandleValuePairSet& atts);
}
```

```

    virtual const RTI::AttributeHandleValuePairSet& avPairSet()
const;
    // Set/Get the attribute handle given an index
    virtual void setAttributeHandle(DtU32 index, DtU32 handle);
    virtual DtU32 attributeHandle(DtU32 index) const;

    // Set/Get the attribute value length given an index
    virtual void setAttributeValueLength(DtU32 index, DtU32 length);
    virtual DtU32 attributeValueLength(DtU32 index) const;

    virtual void setAttributeValue(DtU32 index, const char* usr);
    virtual const char* attributeValue(DtU32 index) const;

    // Return the DtDatumRecordSet which an InteractionWithData
    // uses to store its fixed and var records. In general,
    // the wrapper functions above should be used rather than going
directly
    // through the DtDatumRecordSet, which is necessary only when
doing
    // encoding and decoding of network representations.
    virtual DtDatumRecordSet* datumRecSet();
    virtual const DtDatumRecordSet* datumRecSet() const;

    // Set/Get request ID
    virtual void setRequestId(DtU32 id);
    virtual DtU32 requestId() const;

    // Name of the interaction (Not nec. the ClassName from the FOM).
    virtual const char* name() const;

public:

    // Create a DtCreateObjReqInter.
    // Caller is responsible for deletion.
    static DtInteraction* create();

    // Add (register)/Remove (deregister) function to be called when
    // interaction occurs.
    static void addCallback(DtExerciseConn* conn,
        DtCreateObjReqInterCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtCreateObjReqInterCb cb, void* usr);

protected:

    DtU32 myObjectClass;
    DtDatumRecordSet myRecordSet;
    DtU32 myRequestId;

protected:

```

```
    static char* theClassName;  
};  
#endif
```


hCrtObjResDec.h

```

#if DtHLA

#include "interDecoder.h"

// This can be used to declare parameter processor functions
// which are static members of a class.

#define DtDECLARE_CREATOBJRES_PARAM_DECODER(paramName) \
    DtDECLARE_PARAM_DECODER(DtCreateObjectResultInter, paramName)

#define DtDEFINE_SIMPLE_CREATOBJRES_PARAM_DECODER( \
    paramName, netType, mutator) \
    DtDEFINE_SIMPLE_PARAM_DECODER( \
        DtCreateObjectResultDecoder, DtCreateObjectResultInter, \
        paramName, netType, mutator)

#define DtDEFINE_SIMPLE_CREATOBJRES_PARAM_DECODER_WITH_CAST( \
    paramName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_PARAM_DECODER_WITH_CAST( \
        DtCreateObjectResultDecoder, DtCreateObjectResultInter, \
        paramName, netType, mutator, castExpr)

// class DtCreateObjectResultDecoder:
//
// Instances of DtCreateObjectResultDecoder are used to decode
// CreateObjectResult Interactions from the network.

class DT_DLL_VL DtCreateObjectResultDecoder : public DtInteraction-
Decoder
{
public:

    // constructor
    DtCreateObjectResultDecoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtCreateObjectResultDecoder();

    // copy constructor
    DtCreateObjectResultDecoder(const DtCreateObjectResultDecoder&
orig);

    // assignment operator
    DtCreateObjectResultDecoder& operator=(
        const DtCreateObjectResultDecoder& orig);

```

protected:

```
    // Individual parameter decoders

    DtDECLARE_CREATOBJRES_PARAM_DECODER(CreateObjectResult);
    DtDECLARE_CREATOBJRES_PARAM_DECODER(RequestID);

    // RPR FOM 0.7 compatibility
    DtDECLARE_CREATOBJRES_PARAM_DECODER(CreateObjectResultPre08);
};
#endif
```

hCrtObjResEnc.h

```

#if DtHLA

#include "interEncoder.h"

#define DtDECLARE_CRTOBJRES_PARAM_ENCODER(paramName) \
    DtDECLARE_PARAM_ENCODER(DtCreateObjectResultInter, paramName)

#define DtDEFINE_SIMPLE_CRTOBJRES_PARAM_ENCODER(paramName, netType, \
    inspector) \
    DtDEFINE_SIMPLE_PARAM_ENCODER(DtCreateObjectResultEncoder, \
    DtCreateObjectResultInter, paramName, netType, inspector)

// class DtCreateObjectResultEncoder:
//
// DtCreateObjectResultEncoder is a DtInteractionEncoder that regis-
// ters
// parameter encoders for all of the CreateObjectResult interaction
// parameters.

class DT_DLL_VL DtCreateObjectResultEncoder : public DtInteraction-
Encoder
{
public:

    DtCreateObjectResultEncoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtCreateObjectResultEncoder();

    // copy constructor
    DtCreateObjectResultEncoder(const DtCreateObjectResultEncoder&
orig);

    // assignment operator
    DtCreateObjectResultEncoder& operator=(
        const DtCreateObjectResultEncoder& orig);

protected:

    // Individual parameter encoders

    DtDECLARE_CRTOBJRES_PARAM_ENCODER(CreateObjectResult);
    DtDECLARE_CRTOBJRES_PARAM_ENCODER(RequestID);

    // RPR FOM 0.7 compatibility
    DtDECLARE_CRTOBJRES_PARAM_ENCODER(CreateObjectResultPre08);

```

```
};  
#endif
```

hCrtObjResInt.h

```

#if DtHLA

#include "hIntWEncDec.h"
#include "hostStructs.h"
#include "mathTypes.h"

#include "chooseDefines.h"

// class DtCreateObjectResultInter:
//
// Instances of DtCreateObjectResultInter are used to represent
// CreateObjectResult Interactions in the RPR FOM.

// CreateObjectResult interaction callback function signature.
typedef void (*DtCreateObjectResultInterCb)
    (DtCreateObjectResultInter* inter, void* usr);

class DT_DLL_VL DtCreateObjectResultInter : public DtInteraction-
WithEncDec
{
public:

    // default constructor
    DtCreateObjectResultInter();

    // destructor
    virtual ~DtCreateObjectResultInter();

    // copy constructor
    DtCreateObjectResultInter(const DtCreateObjectResultInter& orig);

    // assignment operator
    DtCreateObjectResultInter& operator=(
        const DtCreateObjectResultInter& orig);

    // Print the interaction's data.
    virtual void printData() const;

    // Set/Get create object result
    virtual void setCreateObjectResult(DtCreateObjectResult result);
    virtual DtCreateObjectResult createObjectResult() const;

    // Set/Get request ID
    virtual void setRequestId(DtU32 id);
    virtual DtU32 requestId() const;

    // Name of the interaction (Not nec. the ClassName from the FOM).

```

```
    virtual const char* name() const;

public:

    // Create a DtCreateObjectResultInter.
    // Caller is responsible for deletion.
    static DtInteraction* create();

    // Add (register)/Remove (deregister) function to be called when
    // interaction occurs.
    static void addCallback(DtExerciseConn* conn,
        DtCreateObjectResultInterCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtCreateObjectResultInterCb cb, void* usr);

protected:

    DtCreateObjectResult myCreateObjectResult;
    DtU32 myRequestId;

protected:

    static char* theClassName;
};
#endif
```

hDataDec.h

```

#if DtHLA

#include "interDecoder.h"

// This can be used to declare parameter processor functions
// which are static members of a class.

#define DtDECLARE_DATA_PARAM_DECODER(paramName) \
    DtDECLARE_PARAM_DECODER(DtDataInteraction, paramName)

#define DtDEFINE_SIMPLE_DATA_PARAM_DECODER( \
    paramName, netType, mutator) \
    DtDEFINE_SIMPLE_PARAM_DECODER( \
        DtDataDecoder, DtDataInteraction, \
        paramName, netType, mutator)

#define DtDEFINE_SIMPLE_DATA_PARAM_DECODER_WITH_CAST( \
    paramName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_PARAM_DECODER_WITH_CAST( \
        DtDataDecoder, DtDataInteraction, \
        paramName, netType, mutator, castExpr)

// class DtDataDecoder:
//
// Instances of DtDataDecoder are used to decode Data
// Interactions from the network.

class DT_DLL_VL DtDataDecoder : public DtInteractionDecoder
{
public:

    // constructor
    DtDataDecoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtDataDecoder();

    // copy constructor
    DtDataDecoder(const DtDataDecoder& orig);

    // assignment operator
    DtDataDecoder& operator=(const DtDataDecoder& orig);

protected:

    // Individual parameter decoders

```

```
DtDECLARE_DATA_PARAM_DECODER(OriginatingEntity);  
DtDECLARE_DATA_PARAM_DECODER(ReceivingEntity);  
DtDECLARE_DATA_PARAM_DECODER(RequestID);  
DtDECLARE_DATA_PARAM_DECODER(FixedDatums);  
DtDECLARE_DATA_PARAM_DECODER(VariableDatumSet);  
};  
#endif
```


hDataEnc.h

```

#if DtHLA

#include "interEncoder.h"

#define DtDECLARE_DATA_PARAM_ENCODER(paramName) \
    DtDECLARE_PARAM_ENCODER(DtDataInteraction, paramName)

#define DtDEFINE_SIMPLE_DATA_PARAM_ENCODER(paramName, netType, \
    inspector) \
    DtDEFINE_SIMPLE_PARAM_ENCODER(DtDataEncoder, \
    DtDataInteraction, paramName, netType, inspector)

// class DtDataEncoder:
//
// DtDataEncoder is a DtInteractionEncoder that registers
// parameter encoders for all of the Data interaction parameters.

class DT_DLL_VL DtDataEncoder : public DtInteractionEncoder
{
public:

    DtDataEncoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtDataEncoder();

    // copy constructor
    DtDataEncoder(const DtDataEncoder& orig);

    // assignment operator
    DtDataEncoder& operator=(const DtDataEncoder& orig);

protected:

    // Individual parameter encoders

    DtDECLARE_DATA_PARAM_ENCODER(OriginatingEntity);
    DtDECLARE_DATA_PARAM_ENCODER(ReceivingEntity);
    DtDECLARE_DATA_PARAM_ENCODER(RequestID);
    DtDECLARE_DATA_PARAM_ENCODER(FixedDatums);
    DtDECLARE_DATA_PARAM_ENCODER(VariableDatumSet);
};
#endif

```

hDataInter.h

```
#if DtHLA

#include "hInterWData.h"
#include "hostStructs.h"
#include "mathTypes.h"

#include "chooseDefines.h"

// class DtDataInteraction:
//
// Instances of DtDataInteraction are used to represent
// Data Interactions in the RPR FOM. The variable and fixed
// datum records are accessed through DtInteractionWithData member
// functions.

// Data interaction callback function signature.
typedef void (*DtDataInteractionCb)
    (DtDataInteraction* inter, void* usr);

class DT_DLL_VL DtDataInteraction : public DtInteractionWithData
{
public:

    // default constructor
    DtDataInteraction();

    // destructor
    virtual ~DtDataInteraction();

    // copy constructor
    DtDataInteraction(const DtDataInteraction& orig);

    // assignment operator
    DtDataInteraction& operator=(
        const DtDataInteraction& orig);

    // Print the interaction's data.
    virtual void printData() const;

    // 'Set/Get sender ID
    virtual void setSenderId(const DtEntityIdentifier& id);
    virtual const DtEntityIdentifier& senderId() const;

    // Set/Get receiver ID
    virtual void setReceiverId(const DtEntityIdentifier& id);
    virtual const DtEntityIdentifier& receiverId() const;
```

```
// Set/Get request ID
virtual void setRequestId(unsigned long id);
virtual unsigned long requestId() const;

// Name of the interaction (Not nec. the ClassName from the FOM).
virtual const char* name() const;

public:

    // Create a DtDataInteraction.
    // Caller is responsible for deletion.
    static DtInteraction* create();

    // Add (register)/Remove (deregister) function to be called when
    // interaction occurs.
    static void addCallback(DtExerciseConn* conn,
        DtDataInteractionCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtDataInteractionCb cb, void* usr);

protected:

    DtEntityIdentifier myOrigEnt;
    DtEntityIdentifier myReceiveEnt;
    int myRequestId;
};
#endif
```

hDataQDec.h

```
#if DtHLA

#include "interDecoder.h"

// This can be used to declare parameter processor functions
// which are static members of a class.

#define DtDECLARE_DATAQ_PARAM_DECODER(paramName) \
    DtDECLARE_PARAM_DECODER(DtDataQueryInteraction, paramName)

#define DtDEFINE_SIMPLE_DATAQ_PARAM_DECODER( \
    paramName, netType, mutator) \
    DtDEFINE_SIMPLE_PARAM_DECODER( \
        DtDataQueryDecoder, DtDataQueryInteraction, \
        paramName, netType, mutator)

#define DtDEFINE_SIMPLE_DATAQ_PARAM_DECODER_WITH_CAST( \
    paramName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_PARAM_DECODER_WITH_CAST( \
        DtDataQueryDecoder, DtDataQueryInteraction, \
        paramName, netType, mutator, castExpr)

// class DtDataQueryDecoder:
//
// Instances of DtDataQueryDecoder are used to decode DataQuery
// Interactions from the network.

class DT_DLL_VL DtDataQueryDecoder : public DtInteractionDecoder
{
public:

    // constructor
    DtDataQueryDecoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtDataQueryDecoder();

    // copy constructor
    DtDataQueryDecoder(const DtDataQueryDecoder& orig);

    // assignment operator
    DtDataQueryDecoder& operator=(const DtDataQueryDecoder& orig);

protected:

    // Individual parameter decoders
```

```
DtDECLARE_DATAQ_PARAM_DECODER(OriginatingEntity);  
DtDECLARE_DATAQ_PARAM_DECODER(ReceivingEntity);  
DtDECLARE_DATAQ_PARAM_DECODER(RequestID);  
DtDECLARE_DATAQ_PARAM_DECODER(TimeInterval);  
DtDECLARE_DATAQ_PARAM_DECODER(FixedDatumIDs);  
DtDECLARE_DATAQ_PARAM_DECODER(VariableDatumIDs);  
};  
#endif
```

hDataQEnc.h

```
#if DtHLA

#include "interEncoder.h"

#define DtDECLARE_DATAQ_PARAM_ENCODER(paramName) \
    DtDECLARE_PARAM_ENCODER(DtDataQueryInteraction, paramName)

#define DtDEFINE_SIMPLE_DATAQ_PARAM_ENCODER(paramName, netType, \
    inspector) \
    DtDEFINE_SIMPLE_PARAM_ENCODER(DtDataQueryEncoder, \
        DtDataQueryInteraction, paramName, netType, inspector)

// class DtDataQueryEncoder:
//
// DtDataQueryEncoder is a DtInteractionEncoder that registers
// parameter encoders for all of the DataQuery interaction param-
// eters.

class DT_DLL_VL DtDataQueryEncoder : public DtInteractionEncoder
{
public:

    DtDataQueryEncoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtDataQueryEncoder();

    // copy constructor
    DtDataQueryEncoder(const DtDataQueryEncoder& orig);

    // assignment operator
    DtDataQueryEncoder& operator=(const DtDataQueryEncoder& orig);

protected:

    // Individual parameter encoders

    DtDECLARE_DATAQ_PARAM_ENCODER(OriginatingEntity);
    DtDECLARE_DATAQ_PARAM_ENCODER(ReceivingEntity);
    DtDECLARE_DATAQ_PARAM_ENCODER(RequestID);
    DtDECLARE_DATAQ_PARAM_ENCODER(TimeInterval);
    DtDECLARE_DATAQ_PARAM_ENCODER(FixedDatumIDs);
    DtDECLARE_DATAQ_PARAM_ENCODER(VariableDatumIDs);
};
#endif
```

hDataQInter.h

```

#if DtHLA

#include "hostStructs.h"
#include "mathTypes.h"
#include "hIntWEncDec.h"

#include "chooseDefines.h"

// class DtDataQueryInteraction:
//
// Instances of DtDataQueryInteraction are used to represent
// DataQuery Interactions in the RPR FOM.

// DataQuery interaction callback function signature.
typedef void (*DtDataQueryInteractionCb)
    (DtDataQueryInteraction* inter, void* usr);

class DT_DLL_VL DtDataQueryInteraction: public DtInteractionWith-
EncDec
{
public:

    // default constructor
    DtDataQueryInteraction();

    // destructor
    virtual ~DtDataQueryInteraction();

    // copy constructor
    DtDataQueryInteraction(const DtDataQueryInteraction& orig);

    // assignment operator
    DtDataQueryInteraction& operator=(
        const DtDataQueryInteraction& orig);

    // Print the interaction's data.
    virtual void printData() const;

    // 'Set/Get sender ID
    virtual void setSenderId(const DtEntityIdentifier& id);
    virtual const DtEntityIdentifier& senderId() const;

    // Set/Get receiver ID
    virtual void setReceiverId(const DtEntityIdentifier& id);
    virtual const DtEntityIdentifier& receiverId() const;

    // Set/Get request ID

```

```
virtual void setRequestId(unsigned long id);
virtual unsigned long requestId() const;

// Set/Get the time interval.
virtual void setTimeInterval(DtTime seconds);
virtual DtTime timeInterval() const;

// Initialize fixed and variable datum ids.
virtual void initDatumIds(unsigned long numFixedFields,
                          unsigned long numVarFields);

// Set/Get fixed fields.
virtual void setNumFixedFields(unsigned long numFixedFields);
virtual unsigned long numFixedFields() const;
virtual int setFixedDatumId(DtU32 index, DtDatumParam id);
virtual DtDatumParam fixedDatumId(DtU32 index) const;

// Set/Get variable fields.
virtual void setNumVarFields(unsigned long numVarFields);
virtual unsigned long numVarFields() const;
virtual int setVarDatumId(DtU32 index, DtDatumParam id);
virtual DtDatumParam varDatumId(DtU32 index) const;

// Set/Get the fixed fields memory block all at once
virtual void setFixedDatumIDs(const void* data, DtInt32
numBytes);
virtual const void* fixedDatumIDs() const;

// Set/Get the variable fields memory block all at once
virtual void setVariableDatumIDs(const void* data, DtInt32
numBytes);
virtual const void* variableDatumIDs() const;

// Change size of fixed or variable fields
virtual void changeFixedSize(DtU32 newSize);
virtual void changeVariableSize(DtU32 newSize);

// Name of the interaction (Not nec. the ClassName from the FOM).
virtual const char* name() const;

public:

// Create a DtDataQueryInteraction.
// Caller is responsible for deletion.
static DtInteraction* create();

// Add (register)/Remove (deregister) function to be called when
// interaction occurs.
static void addCallback(DtExerciseConn* conn,
DtDataQueryInteractionCb cb, void* usr);
```



```
static void removeCallback(DtExerciseConn* conn,
                          DtDataQueryInteractionCb cb, void* usr);

protected:

    DtEntityIdentifier myOrigEnt;
    DtEntityIdentifier myReceiveEnt;
    DtU32 myRequestId;
    DtNetTimeStamp myTimeInterval;
    DtNetU32* myFixedDatumIDs;
    DtNetU32* myVariableDatumIDs;
    DtU32 myNumFixed;
    DtU32 myNumVar;
};
#endif
```

hDbiRadSigDec.h

```
#if DtHLA

#include "chooseDefines.h"
#include "interDecoder.h"

// This can be used to declare parameter processor functions
// which are static members of a class.

#define DtDECLARE_DATABASE_RADIO_SIGNAL_PARAM_DECODER(paramName) \
    DtDECLARE_PARAM_DECODER(DtSignalInteraction, paramName)

#define DtDEFINE_SIMPLE_DATABASE_RADIO_SIGNAL_PARAM_DECODER(param- \
    Name, \
    netType, mutator) \
    DtDEFINE_SIMPLE_PARAM_DECODER(DtDatabaseIndexRadioSignalDe- \
    coder, \
    DtSignalInteraction, paramName, netType, mutator)

#define
DtDEFINE_SIMPLE_DATABASE_RADIO_SIGNAL_PARAM_DECODER_WITH_CAST( \
    paramName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_PARAM_DECODER_WITH_CAST( \
    DtDatabaseIndexRadioSignalDecoder, DtSignalInteraction, \
    paramName, netType, mutator, castExpr)

// class DtDatabaseIndexRadioSignalDecoder:
//
// Instances of DtDatabaseIndexRadioSignalDecoder are

class DT_DLL_VL DtDatabaseIndexRadioSignalDecoder : public DtInter- \
actionDecoder
{
public:

    // default constructor
    DtDatabaseIndexRadioSignalDecoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtDatabaseIndexRadioSignalDecoder();

    // copy constructor
    DtDatabaseIndexRadioSignalDecoder(
        const DtDatabaseIndexRadioSignalDecoder& orig);

    // assignment operator
    DtDatabaseIndexRadioSignalDecoder& operator=(
```

```
    const DtDatabaseIndexRadioSignalDecoder& orig);  
  
protected:  
  
    DtDECLARE_DATABASE_RADIO_SIGNAL_PARAM_DECODER(HostRadioID);  
    DtDECLARE_DATABASE_RADIO_SIGNAL_PARAM_DECODER(DatabaseIndex);  
    DtDECLARE_DATABASE_RADIO_SIGNAL_PARAM_DECODER(Duration);  
    DtDECLARE_DATABASE_RADIO_SIGNAL_PARAM_DECODER(StartOffset);  
    DtDECLARE_DATABASE_RADIO_SIGNAL_PARAM_DECODER(TacticalDataLink-  
Type);  
    DtDECLARE_DATABASE_RADIO_SIGNAL_PARAM_DECODER(TDLMessageCount);  
};  
#endif    // DtHLA
```

hDbiRadSigEnc.h

```
#if DtHLA

#include "chooseDefines.h"
#include "interEncoder.h"

#define DtDECLARE_DATABASE_RADIO_SIGNAL_PARAM_ENCODER(paramName) \
    DtDECLARE_PARAM_ENCODER(DtSignalInteraction, paramName)

#define DtDEFINE_SIMPLE_DATABASE_RADIO_SIGNAL_PARAM_ENCODER(param- \
    Name, \
    netType, inspector) \
    DtDEFINE_SIMPLE_PARAM_ENCODER(DtDatabaseIndexRadioSignalEn- \
    coder, \
    DtSignalInteraction, paramName, netType, inspector)

// class DtDatabaseIndexRadioSignalEncoder:
//
// Instances of DtDatabaseIndexRadioSignalEncoder are

class DT_DLL_VL DtDatabaseIndexRadioSignalEncoder : public DtInter- \
    actionEncoder
{
public:

    // default constructor
    DtDatabaseIndexRadioSignalEncoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtDatabaseIndexRadioSignalEncoder();

    // copy constructor
    DtDatabaseIndexRadioSignalEncoder(
        const DtDatabaseIndexRadioSignalEncoder& orig);

    // assignment operator
    DtDatabaseIndexRadioSignalEncoder& operator=(
        const DtDatabaseIndexRadioSignalEncoder& orig);

protected:

    DtDECLARE_DATABASE_RADIO_SIGNAL_PARAM_ENCODER(HostRadioID);
    DtDECLARE_DATABASE_RADIO_SIGNAL_PARAM_ENCODER(DatabaseIndex);
    DtDECLARE_DATABASE_RADIO_SIGNAL_PARAM_ENCODER(Duration);
    DtDECLARE_DATABASE_RADIO_SIGNAL_PARAM_ENCODER(StartOffset);
    DtDECLARE_DATABASE_RADIO_SIGNAL_PARAM_ENCODER(TacticalDataLink- \
    Type);
```

```
    DtDECLARE_DATABASE_RADIO_SIGNAL_PARAM_ENCODER(TDLMessageCount);  
};  
#endif    //DtHLA
```

hDesignatrPub.h

```
#if DtHLA

#include "hObjPub.h"

// class DtDesignatorPublisher:
//
// Instances of DtDesignatorPublisher are used for local or source
// simulations
// of designators, (i.e. these entities are not dependent on data
// being
// produced elsewhere on the network for their existence). These
// designators send data to the network.

class DT_DLL_VL DtDesignatorPublisher : public DtObjectPublisher
{
public:

    // Function prototype for functions that create DtDesignatorRe-
    // positories
    typedef DtDesignatorRepository* (*DtStateRepCreator)();

public:

    // constructor
    DtDesignatorPublisher(RTI::ObjectClassHandle h,
        DtExerciseConn* conn, const char* name = NULL);

    // constructor
    DtDesignatorPublisher(DtExerciseConn* conn, const char* name =
    NULL);

    // destructor
    virtual ~DtDesignatorPublisher();

    // Returns a pointer to the DtDesignatorRepository - through
    // which you can set or inspect current state information.
    virtual DtDesignatorRepository* designatorStateRep();

    // Synonym for designatorStateRep()
    virtual DtDesignatorRepository* dsr();

    // Returns the name of the FOM object class to use. Default
    // implementation
    // is to ask the FOM Mapper.
    virtual const char* classToUse() const;

public:
```

```
    // Give us a function to be used for creating SRs.
    static DtStateRepCreator setStateRepCreator(DtStateRepCreator
creator);

protected:

    // copy constructor
    DtDesignatorPublisher(const DtDesignatorPublisher& orig);

    // copy constructor
    DtDesignatorPublisher& operator=
        (const DtDesignatorPublisher& orig);

protected:

    static DtStateRepCreator theStateRepCreator;
};
#endif    //DtDesignatorPublisher_H_
```

hDetDec.h

```
#if DtHLA

#include "interDecoder.h"

#include "chooseDefines.h"

// This can be used to declare parameter processor functions
// which are static members of a class.

#define DtDECLARE_DETONATION_PARAM_DECODER(paramName) \
    DtDECLARE_PARAM_DECODER(DtDetonationInteraction, paramName)

#define DtDEFINE_SIMPLE_DETONATION_PARAM_DECODER( \
    paramName, netType, mutator) \
    DtDEFINE_SIMPLE_PARAM_DECODER( \
        DtDetonationDecoder, DtDetonationInteraction, \
        paramName, netType, mutator)

#define DtDEFINE_SIMPLE_DETONATION_PARAM_DECODER_WITH_CAST( \
    paramName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_PARAM_DECODER_WITH_CAST( \
        DtDetonationDecoder, DtDetonationInteraction, \
        paramName, netType, mutator, castExpr)

// class DtDetonationDecoder:
//
// Instances of DtDetonationDecoder are used to decode Detonation
// Interactions from the network.

class DT_DLL_VL DtDetonationDecoder : public DtInteractionDecoder
{
public:

    // constructor
    DtDetonationDecoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtDetonationDecoder();

    // copy constructor
    DtDetonationDecoder(const DtDetonationDecoder& orig);

    // assignment operator
    DtDetonationDecoder& operator=(const DtDetonationDecoder& orig);

protected:
```



```
// Individual parameter decoders

#if !DtrPR016
    DtDECLARE_DETONATION_PARAM_DECODER(ArticulatedPartsArray);
#else
    DtDECLARE_DETONATION_PARAM_DECODER(ArticulatedParametersArray);
    DtDECLARE_DETONATION_PARAM_DECODER(ArticulatedParametersCount);
#endif
    DtDECLARE_DETONATION_PARAM_DECODER(DetonationLocation);
    DtDECLARE_DETONATION_PARAM_DECODER(DetonationResult);
    DtDECLARE_DETONATION_PARAM_DECODER(EventID);
    DtDECLARE_DETONATION_PARAM_DECODER(FiringObjectID);
    DtDECLARE_DETONATION_PARAM_DECODER(FinalVelocityVector);
    DtDECLARE_DETONATION_PARAM_DECODER(FuseType);
    DtDECLARE_DETONATION_PARAM_DECODER(MunitionObjectID);
    DtDECLARE_DETONATION_PARAM_DECODER(MunitionType);
    DtDECLARE_DETONATION_PARAM_DECODER(QuantityFired);
    DtDECLARE_DETONATION_PARAM_DECODER(RateOfFire);
    DtDECLARE_DETONATION_PARAM_DECODER(RelativeDetonationLocation);
    DtDECLARE_DETONATION_PARAM_DECODER(TargetObjectID);
    DtDECLARE_DETONATION_PARAM_DECODER(WarheadType);
};
#endif
```

hDetEnc.h

```
#if DtHLA

#include "interEncoder.h"

#define DtDECLARE_DETONATION_PARAM_ENCODER(paramName) \
    DtDECLARE_PARAM_ENCODER(DtDetonationInteraction, paramName)

#define DtDEFINE_SIMPLE_DETONATION_PARAM_ENCODER(paramName, \
    netType, inspector) \
    DtDEFINE_SIMPLE_PARAM_ENCODER(DtDetonationEncoder, DtDetonation- \
    Interaction, \
                                   paramName, netType, \
                                   inspector)

// class DtDetonationEncoder:
//
// DtDetonationEncoder is a DtInteractionEncoder that registers
// parameter encoders for all of the DETONATION interaction param-
// eters.

class DT_DLL_VL DtDetonationEncoder : public DtInteractionEncoder
{
public:

    DtDetonationEncoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtDetonationEncoder();

    // copy constructor
    DtDetonationEncoder(const DtDetonationEncoder& orig);

    // assignment operator
    DtDetonationEncoder& operator=(const DtDetonationEncoder& orig);

protected:

#if !DtrPR016
    DtDECLARE_DETONATION_PARAM_ENCODER(ArticulatedPartsArray);
#else
    DtDECLARE_DETONATION_PARAM_ENCODER(ArticulatedParametersArray);
    DtDECLARE_DETONATION_PARAM_ENCODER(ArticulatedParametersCount);
#endif
    DtDECLARE_DETONATION_PARAM_ENCODER(DetonationLocation);
    DtDECLARE_DETONATION_PARAM_ENCODER(DetonationResult);
    DtDECLARE_DETONATION_PARAM_ENCODER(EventID);
```

```
DtDECLARE_DETONATION_PARAM_ENCODER(FiringObjectID);
DtDECLARE_DETONATION_PARAM_ENCODER(FinalVelocityVector);
DtDECLARE_DETONATION_PARAM_ENCODER(FuseType);
DtDECLARE_DETONATION_PARAM_ENCODER(MunitionObjectID);
DtDECLARE_DETONATION_PARAM_ENCODER(MunitionType);
DtDECLARE_DETONATION_PARAM_ENCODER(QuantityFired);
DtDECLARE_DETONATION_PARAM_ENCODER(RateOfFire);
DtDECLARE_DETONATION_PARAM_ENCODER(RelativeDetonationLocation);
DtDECLARE_DETONATION_PARAM_ENCODER(TargetObjectID);
DtDECLARE_DETONATION_PARAM_ENCODER(WarheadType);
};
#endif
```

hDetInter.h

```
#if DtHLA

#include "hIntWEncDec.h"
#include "hostStructs.h"
#include "mathTypes.h"
#include "artPartList.h"

#include "chooseDefines.h"

// class DtDetonationInteraction:
//
// Instances of DtDetonationInteraction are used to represent
// Detonation Interactions in the RPR ROM.

// Detonation interaction callback function signature.
typedef void (*DtDetonationInteractionCb) (DtDetonationInteraction* inter,
      void* usr);

class DT_DLL_VL DtDetonationInteraction : public DtInteractionWithEncDec
{
public:

    // default constructor
    DtDetonationInteraction();

    // Constructor
    // Fills in common fields with data from Fire Interaction
    DtDetonationInteraction(const DtFireInteraction& fire);

    // destructor
    virtual ~DtDetonationInteraction();

    // copy constructor
    DtDetonationInteraction(const DtDetonationInteraction& orig);

    // assignment operator
    DtDetonationInteraction& operator=(const DtDetonationInteraction& orig);

    // Print the interaction's data.
    virtual void printData() const;

    // Set/Get the attacker identifier.
    virtual void setAttackerId(const DtGlobalObjectDesignator& id);
    virtual const DtGlobalObjectDesignator& attackerId() const;
```

```
// Set/Get the target identifier.
virtual void setTargetId(const DtGlobalObjectDesignator& id);
virtual const DtGlobalObjectDesignator& targetId() const;

// Set/Get the munition identifier.
virtual void setMunitionId(const DtGlobalObjectDesignator& id);
virtual const DtGlobalObjectDesignator& munitionId() const;

// Set/Get the detonation event id.
virtual void setEventId(const DtEventID& id);
virtual const DtEventID& eventId() const;

// Set/Get the velocity.
virtual void setVelocity(DtConstVector velocity);
virtual DtConstVector velocity() const;

// Set/Get the location of the detonation in world coordinates.
virtual void setWorldLocation(DtConstVector location);
virtual DtConstVector worldLocation() const;

// Set/Get the location of the detonation in entity coordinates.
virtual void setEntityLocation(DtConstVector location);
virtual DtConstVector entityLocation() const;

// Set/Get the detonation result.
virtual void setResult(DtDetonationResult result);
virtual DtDetonationResult result() const;

// Set/Get the detonator fuze type.
virtual void setFuseType(DtDetonatorFuze fuse);
virtual DtDetonatorFuze fuseType() const;

// Set/Get the munition type.
virtual void setMunitionType(const DtEntityType& munType);
virtual const DtEntityType& munitionType() const;

// Set/Get the quantity.
virtual void setQuantity(int quant);
virtual int quantity() const;

// Set/Get the rate.
virtual void setRate(int rate);
virtual int rate() const;

// Set/Get the warhead type.
virtual void setWarheadType(DtWarheadType type);
virtual DtWarheadType warheadType() const;

// Set/Get ArticulatedParts
```

```
virtual void initArtParts(const DtArtPartSpec* specList);
virtual const DtArtPartSpec* artPartSpecs() const;

// Get number of articulation parameters
virtual int numArtParams() const;

// Set/Get articulation parameter values by part type
// and parameter type.
virtual int setArtParamVal(int partType, int param, DtFloat32
value);
virtual int getArtParamVal(
    int partType, int param, DtFloat32* value) const;

// Get a pointer to an DtArtPartList containing the interaction's
// articulated part information.
virtual DtArtPartList* artPartList();
virtual const DtArtPartList* artPartList() const;

// Name of the interaction (Not nec. the ClassName from the FOM).
virtual const char* name() const;

public:

    // Create a DtDetonationInteraction.
    // Caller is responsible for deletion.
    static DtInteraction* create();

    // Add (register)/Remove (deregister) function to be called when
    // interaction occurs.
    static void addCallback(DtExerciseConn* conn,
        DtDetonationInteractionCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtDetonationInteractionCb cb, void* usr);

protected:

    DtVector myWorldLocation;
    DtDetonationResult myResult;
    DtEventID myEventId;
    DtGlobalObjectDesignator myAttackerId;
    DtVector myVelocity;
    DtDetonatorFuze myFuseType;
    DtGlobalObjectDesignator myMunitionId;
    DtEntityType myMunitionType;
    int myQuantity;
    int myRate;
    DtVector myEntityLocation;
    DtGlobalObjectDesignator myTargetId;
    DtWarheadType myWarheadType;
    DtArtPartList myPartList;
```

```
};  
#endif
```

hEmitrBeamPub.h

```
#if DtHLA

#ifndef DtEmitterBeamPublisher_H_
#define DtEmitterBeamPublisher_H_

#include "hObjPub.h"

// class DtEmitterBeamPublisher:
//
// Instances of DtEmitterBeamPublisher are used for local or source
// simulations
// of emitter beams, (i.e. these entities are not dependent on data
// being
// produced elsewhere on the network for their existence). These
// emitter beams send data to the network.

class DT_DLL_VL DtEmitterBeamPublisher : public DtObjectPublisher
{
public:

    // Function prototype for functions that create DtBeamRepositories
    typedef DtEmitterBeamRepository* (*DtStateRepCreator)();

public:

    // constructor. NULL bsr means create your own.
    DtEmitterBeamPublisher(
        RTI::ObjectClassHandle h,
        DtExerciseConn* conn,
        DtEmitterBeamRepository* bsr = NULL,
        const char* name = NULL);

    // constructor. NULL bsr means create your own.
    DtEmitterBeamPublisher(
        DtExerciseConn* conn,
        DtEmitterBeamRepository* bsr = NULL,
        const char* name = NULL);

    // destructor
    virtual ~DtEmitterBeamPublisher();

    // Returns a pointer to the DtEmitterBeamRepository - through
    // which you can set or inspect current state information.
    virtual DtEmitterBeamRepository* emitrBeamRep();

    // Synonym for emitrBeamRep()
```



```
virtual DtEmitterBeamRepository* bsr();

// Returns the name of the FOM object class handle to use.
virtual const char* classToUse(
    DtEmitterBeamRepository* bsr) const;

public:

    // Give us a function to be used for creating SRs.
    static DtStateRepCreator setStateRepCreator(DtStateRepCreator
creator);

protected:

    // copy constructor
    DtEmitterBeamPublisher(const DtEmitterBeamPublisher& orig);

    // assignment operator
    DtEmitterBeamPublisher& operator=
        (const DtEmitterBeamPublisher& orig);

protected:

    DtBoolean myOwnBsr;

protected:

    static DtStateRepCreator theStateRepCreator;
};
#endif    //DtEmitterBeamPublisher_H_
```

hEmittrSysPub.h

```
#if DtHLA

#ifndef DtEmitterSystemPublisher_H_
#define DtEmitterSystemPublisher_H_

#include "hObjPub.h"

// class DtEmitterSystemPublisher:
//
// Instances of DtEmitterSystemPublisher are used for local or
// source
// simulations of emitter systems, (i.e. these entities are not
// dependent
// on data being produced elsewhere on the network for their exist-
// ence).
// These emitter systems send data to the network.

class DT_DLL_VL DtEmitterSystemPublisher : public DtObjectPublisher
{
public:

    // Function prototype for functions that create DtEmitterSystem-
    // Repositories
    typedef DtEmitterSystemRepository* (*DtStateRepCreator)();

public:

    // constructor
    DtEmitterSystemPublisher(
        RTI::ObjectClassHandle h, DtExerciseConn* conn,
        const char* name = NULL);

    // constructor
    DtEmitterSystemPublisher(DtExerciseConn* conn,
        const char* name = NULL);

    // destructor
    virtual ~DtEmitterSystemPublisher();

    // Returns a pointer to the DtEmitterSystemRepository - through
    // which you can set or inspect current state information.
    virtual DtEmitterSystemRepository* emitterSystemRep();

    // Synonym for emitterSystemRep()
    virtual DtEmitterSystemRepository* esr();
};

#endif
```

```

    // Returns the name of the FOM object class to use.  Default
implementation
    // is to ask the FOM Mapper.
    virtual const char* classToUse() const;

    // syncBeamList and update beam objects as well as main system
object.
    virtual void tick();

public:

    // Give us a function to be used for creating SRs.
    static DtStateRepCreator setStateRepCreator(DtStateRepCreator
creator);

protected:

    // copy constructor
    DtEmitterSystemPublisher(const DtEmitterSystemPublisher& orig);

    // assignment operator
    DtEmitterSystemPublisher& operator=
        (const DtEmitterSystemPublisher& orig);

    // Update myBeamPubs so that it has a publisher for all beamSRs
    // in my systemSR, but no others.  Called by tick() so that
    // all beamPubs can be ticked after systemSR has been updated.
    virtual void syncBeamList();

    // Returns the publisher responsible for publishing the indicated
BSR,
    // NULL if there is none.
    virtual DtEmitterBeamPublisher* publisherForBsr(
        DtEmitterBeamRepository* bsr);

    // Return number of beam publishers
    virtual int numBeams() const;

protected:

    // List of DtEmitterBeamPubs
    DtList myBeamPubs;

protected:

    static DtStateRepCreator theStateRepCreator;
};
#endif    //DtEmitterSystemPublisher_H_

```

hEncARSigDec.h

```
#if DtHLA

#include "chooseDefines.h"
#include "interDecoder.h"

// This can be used to declare parameter processor functions
// which are static members of a class.

#define DtDECLARE_ENC_AUDIO_RADIO_SIGNAL_PARAM_DECODER(paramName) \
    DtDECLARE_PARAM_DECODER(DtSignalInteraction, paramName)

#define DtDEFINE_SIMPLE_ENC_AUDIO_RADIO_SIGNAL_PARAM_DECODER(param-
Name, \
    netType, mutator) \
    DtDEFINE_SIMPLE_PARAM_DECODER(DtEncodedAudioRadioSignalDecoder,
\
    DtSignalInteraction, paramName, netType, mutator)

#define
DtDEFINE_SIMPLE_ENC_AUDIO_RADIO_SIGNAL_PARAM_DECODER_WITH_CAST( \
    paramName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_PARAM_DECODER_WITH_CAST( \
        DtEncodedAudioRadioSignalDecoder, DtSignalInteraction, \
        paramName, netType, mutator, castExpr)

// class DtEncodedAudioRadioSignalDecoder:
//
// Instances of DtEncodedAudioRadioSignalDecoder are

class DT_DLL_VL DtEncodedAudioRadioSignalDecoder : public DtInterac-
tionDecoder
{
public:

    // default constructor
    DtEncodedAudioRadioSignalDecoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtEncodedAudioRadioSignalDecoder();

    // copy constructor
    DtEncodedAudioRadioSignalDecoder(
        const DtEncodedAudioRadioSignalDecoder& orig);

    // assignment operator
    DtEncodedAudioRadioSignalDecoder& operator=(
```

```
    const DtEncodedAudioRadioSignalDecoder& orig);  
  
protected:  
  
    DtDECLARE_ENC_AUDIO_RADIO_SIGNAL_PARAM_DECODER(HostRadioID);  
    DtDECLARE_ENC_AUDIO_RADIO_SIGNAL_PARAM_DECODER(EncodingType);  
    DtDECLARE_ENC_AUDIO_RADIO_SIGNAL_PARAM_DECODER(SampleRate);  
    DtDECLARE_ENC_AUDIO_RADIO_SIGNAL_PARAM_DECODER(SampleCount);  
    DtDECLARE_ENC_AUDIO_RADIO_SIGNAL_PARAM_DECODER(SignalDataLength);  
    DtDECLARE_ENC_AUDIO_RADIO_SIGNAL_PARAM_DECODER(SignalData);  
};  
#endif    // DtHLA
```

hEncARSigEnc.h

```
#if DtHLA

#include "chooseDefines.h"
#include "interEncoder.h"

#define DtDECLARE_ENC_AUDIO_RADIO_SIGNAL_PARAM_ENCODER(paramName) \
    DtDECLARE_PARAM_ENCODER(DtSignalInteraction, paramName)

#define DtDEFINE_SIMPLE_ENC_AUDIO_RADIO_SIGNAL_PARAM_ENCODER(param- \
    Name, \
    netType, inspector) \
    DtDEFINE_SIMPLE_PARAM_ENCODER(DtEncodedAudioRadioSignalEncoder, \
    \
    DtSignalInteraction, paramName, netType, inspector)

// class DtEncodedAudioRadioSignalEncoder:
//
// Instances of DtEncodedAudioRadioSignalEncoder are

class DT_DLL_VL DtEncodedAudioRadioSignalEncoder : public DtInterac-
tionEncoder
{
public:

    // default constructor
    DtEncodedAudioRadioSignalEncoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtEncodedAudioRadioSignalEncoder();

    // copy constructor
    DtEncodedAudioRadioSignalEncoder(
        const DtEncodedAudioRadioSignalEncoder& orig);

    // assignment operator
    DtEncodedAudioRadioSignalEncoder& operator=(
        const DtEncodedAudioRadioSignalEncoder& orig);

protected:

    DtDECLARE_ENC_AUDIO_RADIO_SIGNAL_PARAM_ENCODER(HostRadioID);
    DtDECLARE_ENC_AUDIO_RADIO_SIGNAL_PARAM_ENCODER(EncodingType);
    DtDECLARE_ENC_AUDIO_RADIO_SIGNAL_PARAM_ENCODER(SampleRate);
    DtDECLARE_ENC_AUDIO_RADIO_SIGNAL_PARAM_ENCODER(SampleCount);
    DtDECLARE_ENC_AUDIO_RADIO_SIGNAL_PARAM_ENCODER(SignalDataL-
    ength);
```

```
    DtDECLARE_ENC_AUDIO_RADIO_SIGNAL_PARAM_ENCODER(SignalData);  
};  
#endif    // DtHLA
```

hEntityPub.h

```
#if DtHLA

#include "hObjPub.h"
#include "threshold.h"

#ifdef __GNUC__ // g++
// Note: Here due to bug in g++ when using class static member func-
tion
// as a default argument. Don't call these otherwise.
#ifdef __GNUC__ // g++
#else
#undef gGuiseSameAsType
#endif
#endif

#endif
```


hEventRepDec.h

```
#if DtHLA

#include "interDecoder.h"

// This can be used to declare parameter processor functions
// which are static members of a class.

#define DtDECLARE_EVENTREP_PARAM_DECODER(paramName) \
    DtDECLARE_PARAM_DECODER(DtEventReportInteraction, paramName)

#define DtDEFINE_SIMPLE_EVENTREP_PARAM_DECODER( \
    paramName, netType, mutator) \
    DtDEFINE_SIMPLE_PARAM_DECODER( \
        DtEventReportDecoder, DtEventReportInteraction, \
        paramName, netType, mutator)

#define DtDEFINE_SIMPLE_EVENTREP_PARAM_DECODER_WITH_CAST( \
    paramName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_PARAM_DECODER_WITH_CAST( \
        DtEventReportDecoder, DtEventReportInteraction, \
        paramName, netType, mutator, castExpr)
```

```
// Individual parameter decoders

DtDECLARE_EVENTREP_PARAM_DECODER(OriginatingEntity);
DtDECLARE_EVENTREP_PARAM_DECODER(ReceivingEntity);
DtDECLARE_EVENTREP_PARAM_DECODER(EventType);
DtDECLARE_EVENTREP_PARAM_DECODER(FixedDatums);
DtDECLARE_EVENTREP_PARAM_DECODER(VariableDatumSet);
};
#endif
```

hEventRepEnc.h

```

#if DtHLA

#include "interEncoder.h"

#define DtDECLARE_EVENTREP_PARAM_ENCODER(paramName) \
    DtDECLARE_PARAM_ENCODER(DtEventReportInteraction, paramName)

#define DtDEFINE_SIMPLE_EVENTREP_PARAM_ENCODER(paramName, netType, \
    inspector) \
    DtDEFINE_SIMPLE_PARAM_ENCODER(DtEventReportEncoder, \
        DtEventReportInteraction, paramName, netType, inspector)

// class DtEventReportEncoder:
//
// DtEventReportEncoder is a DtInteractionEncoder that registers
// parameter encoders for all of the EventReport interaction param-
// eters.

class DT_DLL_VL DtEventReportEncoder : public DtInteractionEncoder
{
public:

    DtEventReportEncoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtEventReportEncoder();

    // copy constructor
    DtEventReportEncoder(const DtEventReportEncoder& orig);

    // assignment operator
    DtEventReportEncoder& operator=(const DtEventReportEncoder&
orig);

protected:

    // Individual parameter encoders

    DtDECLARE_EVENTREP_PARAM_ENCODER(OriginatingEntity);
    DtDECLARE_EVENTREP_PARAM_ENCODER(ReceivingEntity);
    DtDECLARE_EVENTREP_PARAM_ENCODER(EventType);
    DtDECLARE_EVENTREP_PARAM_ENCODER(FixedDatums);
    DtDECLARE_EVENTREP_PARAM_ENCODER(VariableDatumSet);
};
#endif

```

hEventReplnt.h

```
#if DtHLA

#include "hInterWData.h"
#include "hostStructs.h"
#include "mathTypes.h"

#include "chooseDefines.h"

// class DtEventReportInteraction:
//
// Instances of DtEventReportInteraction are used to represent
// EventReport Interactions in the RPR FOM. The variable and fixed
// datum records are accessed through DtInteractionWithData member
// functions.

// EventReport interaction callback function signature.
typedef void (*DtEventReportInteractionCb)
    (DtEventReportInteraction* inter, void* usr);

class DT_DLL_VL DtEventReportInteraction : public DtInteractionWith-
Data
{
public:

    // default constructor
    DtEventReportInteraction();

    // destructor
    virtual ~DtEventReportInteraction();

    // copy constructor
    DtEventReportInteraction(const DtEventReportInteraction& orig);

    // assignment operator
    DtEventReportInteraction& operator=(
        const DtEventReportInteraction& orig);

    // Print the interaction's data.
    virtual void printData() const;

    // Set/Get sender ID
    virtual void setSenderId(const DtEntityIdentifier& id);
    virtual const DtEntityIdentifier& senderId() const;

    // Set/Get receiver ID
    virtual void setReceiverId(const DtEntityIdentifier& id);
    virtual const DtEntityIdentifier& receiverId() const;
```

```
// Set/Get event Type
virtual void setEventType(DtEventType type);
virtual DtEventType eventType() const;

// Name of the interaction (Not nec. the ClassName from the FOM).
virtual const char* name() const;

public:

    // Create a DtEventReportInteraction.
    // Caller is responsible for deletion.
    static DtInteraction* create();

    // Add (register)/Remove (deregister) function to be called when
    // interaction occurs.
    static void addCallback(DtExerciseConn* conn,
        DtEventReportInteractionCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtEventReportInteractionCb cb, void* usr);

protected:

    DtEntityIdentifier myOrigEnt;
    DtEntityIdentifier myReceiveEnt;
    DtEventType myEventType;
};
#endif
```

hExerciseConn.h

```
#ifndef DtExerciseConn_H_
#define DtExerciseConn_H_

#if DtHLA

#include "baseExConn.h"
#include "RTI.hh"
#include "vlTime.h"
#include "hostStructs.h"
#include "callbacks.h"
#include "tStampType.h"
#include "vrlRtiAmb.h"
#include "rprFomMap.h"

#include "chooseDefines.h"

// forward declarations
class DT_DLL_VL DtInteraction;
class DT_DLL_VL DtInteractionFactory;
class DT_DLL_VL DtHlaObjectFactory;
class DT_DLL_VL DtInteractionManager;
class DT_DLL_VL DtLocalHlaObjectManager;
class DT_DLL_VL DtReflectedHlaObjectManager;
class DT_DLL_VL DtStateMsg;
class DT_DLL_VL DtVrlFederateAmbassador;
class DT_DLL_VL DtFom;
class DT_DLL_VL DtFomMapper;

typedef DtVrlFederateAmbassador* (*DtFedAmbCreator)();
typedef DtVrlRtiAmbassador* (*DtRtiAmbCreator)();
typedef DtFomMapper* (*DtFomMapperCreator)(void* usr);
typedef void (*DtFomMapperDeletor)(DtFomMapper*);

// class DtExerciseConn:
//
// Instances of DtExerciseConn are used to provide an application's
// interface or connection to the network in an HLA environment (as
// opposed to DIS).

class DT_DLL_VL DtExerciseConn : public DtBaseExerciseConn
{
public:

    // Constructor. Mapper is a pointer to a new'ed FOM Mapper to
    use. The
    // FOM Mapper's init() function will be called by the DtExercise-
    Conn
```

```

    // constructor. The default value for mapper is a new'ed DtRpr-
    FomMapper.
    // The mapper is deleted by DtExerciseConn destructor. If mapper
    is NULL,
    // we create one using the creator function passed to setFomMap-
    perCreator
    // (default is an empty FOM Mapper). The FOM Mapper's init func-
    tion is
    // called after the RTI is initialized. If fedFileName is NULL,
    // <execName>.fed is used.
    DtExerciseConn(const char* execName, const char* federateName,
        DtFomMapper* mapper = DtRprFomMapper::create(NULL),
        const char* fedFileName = NULL);

    // Constructor. Similar to above, but rather than passing a FOM
    Mapper,
    // pass the name of a shared library (DSO or DLL) that contains a
    FOM
    // Mapper creation function called DtCreateFomMapper (and a FOM
    Mapper
    // destruction function called DtDeleteFomMapper. fomMapInitData
    is a
    // pointer to arbitrary data that you wish to have passed to the
    FOM Mapper
    // creation function in your FOM Mapper shared library.
    DtExerciseConn(const char* execName, const char* federateName,
        const char* dsoName, const char* fedFileName = NULL,
        void* fomMapInitData = NULL);

    // destructor
    virtual ~DtExerciseConn();

    // Set/Get a flag that determines whether local objects and
    interactions
    // will be reflected back to us. (Default in HLA is no, since the
    RTI does
    // not do this. The set call's return value indicates whether or
    not
    // reflecting could be enabled.
    virtual DtBoolean setReflecting(DtBoolean onOff);
    virtual DtBoolean reflecting() const;

    // Send a state message to the network.
    virtual void send(const DtStateMsg& msg);

    // Send an interaction to the network.
    virtual void send(const DtInteraction& msg);

    // Send a state message to the network which has been
    // time stamped with the current time.

```

```
virtual void sendStamped(const DtStateMsg& msg);

// Send an interaction to the network which has been
// time stamped with the current time.
virtual void sendStamped(const DtInteraction& msg);

// Read and process messages from the network until either no
more
// messages, or for timeout seconds, whichever happens first.
// Will read nReadPerTimeCheck messages at a time before checking
// for timeout.
// Returns number of incoming events which have been processed.
virtual int drainInput(DtTime timeout=-1, int nReadPerTimeCheck=100);

// Set objectID to use in the next event ID
virtual void setObjectIdForEventId(const DtGlobalObjectDesignator& id);

// Get a unique event ID
virtual const DtEventID& nextEventId();

// Get the RTI ambassador.
virtual DtVrIRtiAmbassador* rtiAmb();

// Get the federate ambassador.
virtual DtVrIFederateAmbassador* fedAmb();

// Get the FOM
virtual DtFom* fom();

// Get the interaction factory.
virtual DtInteractionFactory* interactionFactory();

// Get the object factory.
virtual DtHlaObjectFactory* objectFactory();

// Get the Fom Mapper
virtual DtFomMapper* fomMapper();

// Get the reflectedObjectManager
virtual DtReflectedHlaObjectManager* reflectedObjectManager();

// Get the localObjectManager
virtual DtLocalHlaObjectManager* localObjectManager();

// Set/Get time stamp type (DtTimeStampAbsolute or DtTimeStampRelative)
virtual void setTimeStampType(DtTimeStampType relOrAbs);
virtual DtTimeStampType timeStampType() const;
```



```
// Add an interaction callback by class handle, which is called
with the
// user data when the specified interaction is received. Rather
than using
// this function, user code should typically use each derived
// DtInteraction's addCallback function.
virtual void addInteractionCallback(RTI::InteractionClassHandle
h,
    DtReceiveInteractionCb cb,
    void* usr);

// Add an interaction callback by class name, which is called
with the user
// data when the specified interaction is received. Rather than
using this
// function, user code should typically use each derived DtInter-
action's
// addCallback function.
virtual void addInteractionCallbackByName(const char* className,
    DtReceiveInteractionCb cb,
    void* usr);

// Remove an interaction callback by class handle. Note: The
user data is
// significant for identifying the callback and needs to be the
same as
// when the callback was added. Rather than using this function,
user code
// should typically use each derived DtInteraction's addCallback
function.
virtual void removeInteractionCallback(RTI::InteractionClass-
Handle h,
    DtReceiveInteractionCb cb,
    void* usr);

// Remove an interaction callback by class name Note: The user
data is
// significant for identifying the callback and needs to be the
same as
// when the callback was added. Rather than using this function,
user code
// should typically use each derived DtInteraction's addCallback
function.
virtual void removeInteractionCallbackByName(const char* class-
Name,
    DtReceiveInteractionCb cb,
    void* usr);
```

```
// Add a discoverObject callback which is called with the user
data
// when the specified interaction is received.
virtual void addDiscoverObjectCallback(RTI::ObjectClassHandle h,
    DtDiscoverObjectCb cb,
    void* usr);

// Remove a discoverObject callback.
// Note: The user data is significant for identifying the call-
back
// and needs to be the same as when the callback was added.
virtual void removeDiscoverObjectCallback(RTI::ObjectClassHandle
h,
    DtDiscoverObjectCb cb,
    void* usr);

// Add a removeObject callback which is called with the user data
// when the specified interaction is received.
virtual void addRemoveObjectCallback(RTI::ObjectHandle id,
    DtRemoveObjectCb cb,
    void* usr);

// Remove a removeObject callback.
// Note: The user data is significant for identifying the call-
back
// and needs to be the same as when the callback was added.
virtual void removeRemoveObjectCallback(RTI::ObjectHandle id,
    DtRemoveObjectCb cb,
    void* usr);

// Set/Get flag that determines whether or not the DtExerciseConn
// destructor will attempt to destroy the federation execution.
// Default is true.
virtual void setDestroyFedExecFlag(DtBoolean flag);
virtual DtBoolean destroyFedExecFlag();

public:

    // Set/get the function to be used to create a federate ambas-
sador.
    // Call before creating a DtExerciseConn if you want to use
    // something other than a DtVrlFedAmbassador.
    static void setFedAmbCreator(DtFedAmbCreator creator);
    static DtFedAmbCreator fedAmbCreator();

    // Set/get function to be used to create an RTI ambassador.
    // Call before creating a DtExerciseConn if you want to use
    // something other than a DtDirectRtiAmbassador.
    static void setRtiAmbCreator(DtRtiAmbCreator creator);
    static DtRtiAmbCreator rtiAmbCreator();
```

```

    // Set/Get the default FomMapper creator - used to create a
    fomMapper when
    // a NULL FOM Mapper or invalid FOM Mapper DSO name is passed to
    the
    // constructor. Set before creating a DtExConn if you want it to
    have an
    // effect. The FomMapperCreator defaults to DtFomMapper::create.
    usr
    // allows you to specify some arbitrary initialization data that
    will be
    // passed to the FOM Mapper creator when it is called. VR-Link
    maintains a
    // pointer to the object passed as usr as long as the DtExercise-
    Conn and
    // DtFomMapper are around, and does not deleted it.
    static void setFomMapperCreator(DtFomMapperCreator creator,
        void* usr = NULL);
    static DtFomMapperCreator fomMapperCreator();

    // Returns a string indicating the VR-Link version number
    static const char* vrlinkVersion();

```

protected:

```

    // Note: The copy constructor and assignment operator are
    protected
    // because it is not clear what these would mean.
    // An execution name must be unique.
    // These should not be called.

    // copy constructor
    DtExerciseConn(const DtExerciseConn& orig);

    // assignment operator
    DtExerciseConn& operator=(const DtExerciseConn& orig);

    // Not virtual since called by ctor. NULL fedFileName means use
    // <execName>.fed. If FOM Mapper is NULL, use fomMapperCre-
    ator().
    void init(const char* execName, const char* federateName,
        const char* fedFileName, DtFomMapper* mapper);

    // Not virtual since called by ctor. NULL fedFileName means use
    // <execName>.fed. If can't get functions from dso, use
    // fomMapperCreator().
    void init(const char* execName, const char* federateName,
        const char* fedFileName, const char* dsoName, void* fomMapIn-
        itData);

```

```
// Not virtual since called by ctor. Get pointers to functions
called
// DtCreateFomMapper and DtDestroyFomMapper from DSO or DLL named
libName.
// If complain is DtFalse, don't print diagnostic if can't find
library.
// Pointers will point to NULL on failure.
void getFuncsFromSharedLib(const DtString& libName,
    DtFomMapperCreator* creator, DtFomMapperDeletor* deletor,
    DtBoolean complain = DtTrue);

// Not virtual since called by ctor. Initializes myFom from fed
file.
void initFom(const char* execName, const char* fedFileName);

// Return time to send with outgoing messages (default is real
time if
// absolute timestamping, sim time otherwise.
virtual DtTime currentTimeForStamping());

// Create the RTI ambassador. Not virtual since called by ctor.
Generates
// FatalError if unsuccessful.
void createRtiAmb();

// Create the federation execution. Not virtual since called by
ctor.
// Generates FatalError if unsuccessful.
void createFedEx();

// Create the federate ambassador. Not virtual since called by
ctor.
// Generates FatalError if unsuccessful.
void createFedAmb();

// Join the federation execution. Not virtual since called by
ctor.
// Generates FatalError if unsuccessful.
void joinFedEx();

protected:

char* myExecName;
char* myFedFileName;
char* myFedName;
RTI::FederateHandle myFedHandle;
DtVrlRtiAmbassador* myRtiAmb;
DtVrlFederateAmbassador* myFedAmb;

DtFom* myFom;
```

```
DtReflectedHlaObjectManager* myReflectedObjectManager;
DtLocalHlaObjectManager* myLocalObjectManager;
DtInteractionManager* myInteractionManager;

DtFomMapper* myFomMapper;
DtFomMapperCreator myDsoCreator;
DtFomMapperDeletor myDsoDeletor;

DtEventID myCurrentEventId;
DtTimeStampType myRelOrAbs;

DtBoolean myDestroyFedExFlag;
DtBoolean myUseFomMapperDeletor;

protected:

    static DtFedAmbCreator theFedAmbCreator;
    static DtRtiAmbCreator theRtiAmbCreator;
    static DtFomMapperCreator theFomMapperCreator;
    static void* theFomMapInitData;
};

inline DtVrlRtiAmbassador* DtExerciseConn::rtiAmb()
{
    return myRtiAmb;
}

inline DtVrlFederateAmbassador* DtExerciseConn::fedAmb()
{
    return myFedAmb;
}

inline DtReflectedHlaObjectManager* DtExerciseConn::reflectedObjectManager()
{
    return myReflectedObjectManager;
}

inline DtLocalHlaObjectManager* DtExerciseConn::localObjectManager()
{
    return myLocalObjectManager;
}

inline DtFom* DtExerciseConn::fom()
{
    return myFom;
}
```

```
// Macro definition so that exception.what() will work with pre-NG
RTI.
#ifndef RTI_RtiBaseException_h
#define what() _name
#endif

#endif
#endif
```

hFireDec.h

```

#if DtHLA

#include "interDecoder.h"

// This can be used to declare parameter processor functions
// which are static members of a class.

#define DtDECLARE_FIRE_PARAM_DECODER(paramName) \
    DtDECLARE_PARAM_DECODER(DtFireInteraction, paramName)

#define DtDEFINE_SIMPLE_FIRE_PARAM_DECODER(paramName, netType, \
mutator) \
    DtDEFINE_SIMPLE_PARAM_DECODER(DtFireDecoder, DtFireInteraction, \
                                paramName, netType, mutator)

#define DtDEFINE_SIMPLE_FIRE_PARAM_DECODER_WITH_CAST( \
    paramName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_PARAM_DECODER_WITH_CAST( \
        DtFireDecoder, DtFireInteraction, \
        paramName, netType, mutator, castExpr)

// class DtFireDecoder:
//
// Instances of DtFireDecoder are used to decode Fire Interactions
// from the network.

class DT_DLL_VL DtFireDecoder : public DtInteractionDecoder
{
public:

    // constructor
    DtFireDecoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtFireDecoder();

    // copy constructor
    DtFireDecoder(const DtFireDecoder& orig);

    // assignment operator
    DtFireDecoder& operator=(const DtFireDecoder& orig);

protected:

    // Individual parameter decoders

```

```
DtDECLARE_FIRE_PARAM_DECODER(EventID);
DtDECLARE_FIRE_PARAM_DECODER(FireControlSolutionRange);
DtDECLARE_FIRE_PARAM_DECODER(FireMissionIndex);
DtDECLARE_FIRE_PARAM_DECODER(FiringLocation);
DtDECLARE_FIRE_PARAM_DECODER(FiringObjectID);
DtDECLARE_FIRE_PARAM_DECODER(FuseType);
DtDECLARE_FIRE_PARAM_DECODER(InitialVelocityVector);
DtDECLARE_FIRE_PARAM_DECODER(MunitionObjectID);
DtDECLARE_FIRE_PARAM_DECODER(MunitionType);
DtDECLARE_FIRE_PARAM_DECODER(QuantityFired);
DtDECLARE_FIRE_PARAM_DECODER(RateOfFire);
DtDECLARE_FIRE_PARAM_DECODER(TargetObjectID);
DtDECLARE_FIRE_PARAM_DECODER(WarheadType);
};
#endif
```


hFireEnc.h

```

#if DtHLA

#include "interEncoder.h"

#define DtDECLARE_FIRE_PARAM_ENCODER(paramName) \
    DtDECLARE_PARAM_ENCODER(DtFireInteraction, paramName)

#define DtDEFINE_SIMPLE_FIRE_PARAM_ENCODER(paramName, netType, \
    inspector) \
    DtDEFINE_SIMPLE_PARAM_ENCODER(DtFireEncoder, DtFireInteraction, \
                                   paramName, netType, \
                                   inspector)

// class DtFireEncoder:
//
// DtFireEncoder is a DtInteractionEncoder that registers
// parameter encoders for all of the FIRE interaction parameters.

class DT_DLL_VL DtFireEncoder : public DtInteractionEncoder
{
public:

    DtFireEncoder(DtExerciseConn* exConn,
                  DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtFireEncoder();

    // copy constructor
    DtFireEncoder(const DtFireEncoder& orig);

    // assignment operator
    DtFireEncoder& operator=(const DtFireEncoder& orig);

protected:

    DtDECLARE_FIRE_PARAM_ENCODER(EventID);
    DtDECLARE_FIRE_PARAM_ENCODER(FireControlSolutionRange);
    DtDECLARE_FIRE_PARAM_ENCODER(FireMissionIndex);
    DtDECLARE_FIRE_PARAM_ENCODER(FiringLocation);
    DtDECLARE_FIRE_PARAM_ENCODER(FiringObjectID);
    DtDECLARE_FIRE_PARAM_ENCODER(FuseType);
    DtDECLARE_FIRE_PARAM_ENCODER(InitialVelocityVector);
    DtDECLARE_FIRE_PARAM_ENCODER(MunitionObjectID);
    DtDECLARE_FIRE_PARAM_ENCODER(MunitionType);
    DtDECLARE_FIRE_PARAM_ENCODER(QuantityFired);
    DtDECLARE_FIRE_PARAM_ENCODER(RateOfFire);

```

```
DtDECLARE_FIRE_PARAM_ENCODER(TargetObjectID);  
DtDECLARE_FIRE_PARAM_ENCODER(WarheadType);  
};  
#endif
```

hFireInter.h

```

#if DtHLA

#include "hIntWEncDec.h"
#include "hostStructs.h"
#include "mathTypes.h"

#include "chooseDefines.h"

// class DtFireInteraction:
//
// Instances of DtFireInteraction are used to represent
// Fire Interactions in the RPR FOM.

// Fire interaction callback function signature.
typedef void (*DtFireInteractionCb)(DtFireInteraction* inter, void*
usr);

class DT_DLL_VL DtFireInteraction : public DtInteractionWithEncDec
{
public:

    // default constructor
    DtFireInteraction();

    // destructor
    virtual ~DtFireInteraction();

    // copy constructor
    DtFireInteraction(const DtFireInteraction& orig);

    // assignment operator
    DtFireInteraction& operator=(const DtFireInteraction& orig);

    // Print the interaction's data.
    virtual void printData() const;

    // Set/Get the attacker identifier.
    virtual void setAttackerId(const DtGlobalObjectDesignator& id);
    virtual const DtGlobalObjectDesignator& attackerId() const;

    // Set/Get the target identifier.
    virtual void setTargetId(const DtGlobalObjectDesignator& id);
    virtual const DtGlobalObjectDesignator& targetId() const;

    // Set/Get the munition identifier.
    virtual void setMunitionId(const DtGlobalObjectDesignator& id);
    virtual const DtGlobalObjectDesignator& munitionId() const;

```

```
// Set/Get the event id.
virtual void setEventId(const DtEventID& id);
virtual const DtEventID& eventId() const;

// Set/Get the fire mission index.
virtual void setFireMissionIndex(unsigned long i);
virtual unsigned long fireMissionIndex() const;

// Set/Get the location in world coordinates.
virtual void setLocation(DtConstVector location);
virtual DtConstVector location() const;

// Set/Get the velocity.
virtual void setVelocity(DtConstVector velocity);
virtual DtConstVector velocity() const;

// Set/Get the range.
virtual void setRange(double range);
virtual double range() const;

// Set/Get the detonator fuze type.
virtual void setFuseType(DtDetonatorFuze f);
virtual DtDetonatorFuze fuseType() const;

// Set/Get the munition type.
virtual void setMunitionType(const DtEntityType& t);
virtual const DtEntityType& munitionType() const;

// Set/Get the quantity.
virtual void setQuantity(int q);
virtual int quantity() const;

// Set/Get the rate.
virtual void setRate(int r);
virtual int rate() const;

// Set/Get the warhead type.
virtual void setWarheadType(DtWarheadType t);
virtual DtWarheadType warheadType() const;

// Name of the interaction (Not nec. the ClassName from the FOM).
virtual const char* name() const;

public:

    // Create a DtFireInteraction.
    // Caller is responsible for deletion.
    static DtInteraction* create();
```

```
// Add (register)/Remove (deregister) function to be called when
// interaction occurs.
static void addCallback(DtExerciseConn* conn,
    DtFireInteractionCb cb, void* usr);
static void removeCallback(DtExerciseConn* conn,
    DtFireInteractionCb cb, void* usr);
```

protected:

```
DtGlobalObjectDesignator myAttackerId;
DtGlobalObjectDesignator myTargetId;
DtGlobalObjectDesignator myMunitionId;
DtEventID myEventId;
unsigned long myFireMissionIndex;
DtVector myLocation;
DtVector myVelocity;
double myRange;
DtDetonatorFuze myFuseType;
DtEntityType myMunitionType;
int myQuantity;
int myRate;
DtWarheadType myWarheadType;
```

protected:

```
    static char* theClassName;
};
#endif
```

hlInteraction.h

```
#if DtHLA

#include "RTI.hh"
#include "MachineTypes.h"
#include "recIntrCbInfo.h"
#include "vlTime.h"
#include "tStampType.h"
#include "str.h"

#include "chooseDefines.h"

// class DtInteraction:
//
// DtInteraction is an abstract base class for representing HLA
// interactions (or events) (as opposed to HLA objects).

class DT_DLL_VL DtInteraction
{
public:

    // default constructor
    DtInteraction();

    // destructor
    virtual ~DtInteraction();

    // copy constructor
    DtInteraction(const DtInteraction& orig);

    // assignment operator
    DtInteraction& operator=(const DtInteraction& orig);

    // Get the parameter handle value pair set for the interaction.
    virtual const RTI::ParameterHandleValuePairSet& phvps() const =
0;

    // Set contents from a parameter handle value pair set
    virtual void setFromPhvps(
        const RTI::ParameterHandleValuePairSet& pvList) = 0;

    // Sets the interaction's ExerciseConn and class descriptor. If
    // classDesc is NULL, derived class' versions choose a reasonable
    default.
    // Without being told what ExConn or class descriptor to use,
    // many of the HLA-specific functions won't work properly.
    // However, it is usually not necessary for application
    // code to call this function, since it is called by
```

```
// DtExerciseConn::send on the outgoing side, and by the
// DtInteractionFactory on the incoming side.
virtual void setExConn(DtExerciseConn* exConn,
    DtInterClassDesc* classDesc = NULL);

// Get the interaction class handle. Returns 0 if no classDesc.
virtual RTI::InteractionClassHandle interactionClassHandle()
const;

// Get the interaction class name. Returns NULL if no classDesc.
virtual char* interactionClassName() const;

// Get the number of parameters. Returns 0 if no classDesc.
virtual int numParameters() const;

// Returns DtTrue if the interaction class is needed by the
federation
// DtFalse otherwise. Returns DtFalse if no classDesc.
virtual DtBoolean neededByFederation() const;

// Print the interaction's host rep data, including header
virtual void print() const;

// Print the header info
virtual void printHeader() const;

// Print the interaction's host rep data.
virtual void printData() const;

// Print the header followed by the interaction's parameters.
virtual void printParams(DtBoolean withHex) const;

// Name of the interaction (Not nec. the ClassName from the FOM).
virtual const char* name() const = 0;

// Get the current exercise connection
virtual DtExerciseConn* exerciseConn() const;

// Get the current class descriptor we're using for this
// interaction.
virtual DtInterClassDesc* classDesc() const;

// Set/Get the user-defined tag.
virtual void setTag(const DtString& tag);
virtual const DtString& tag() const;

// Set the time of validity of the message, and whether this
// represents relative or absolute time. Time is in seconds, but
only
// time modulo an hour is stored.
```

```
virtual void setTimeStamp(DtTime time,
    DtTimeStampType type = DtTimeStampRelative);

// Return the time of validity of the message (in seconds past an
hour)
// as set by setTimeStamp.
virtual DtTime timeStamp() const;

// Return the type of the time stamp.
virtual DtTimeStampType timeStampType() const;

// Since timestamp only represents number of seconds past the
hour,
// guessTimeValid chooses the hour that would make the timestamp
// closest to reference time, and returns a time that is times-
tamp
// seconds past that hour.
virtual DtTime guessTimeValid(DtTime referenceTime) const;

public:

// Set/Get flag that tells us whether we're using the RPR FOM
// convention that the UserDefinedTag in updates and interactions
// represents a DIS-style timestamp.
static void setTagIsAsciiTime(DtBoolean val);
static DtBoolean tagIsAsciiTime();

// Register/Deregister callback function cb with conn, on
// all interaction class name (char*) contained in classes.
static void addCallback(DtExerciseConn* conn,
    DtList* classes, DtReceiveInteractionCb cb, void* usr);
static void removeCallback(DtExerciseConn* conn,
    DtList* classes, DtReceiveInteractionCb cb, void* usr);

// Can be used instead of above for a single class.
static void addCallback(DtExerciseConn* conn,
    const char* className, DtReceiveInteractionCb cb, void* usr);
static void removeCallback(DtExerciseConn* conn,
    const char* className, DtReceiveInteractionCb cb, void* usr);

// Returns maximum supported number of interaction classes.
static int maxInteractionClasses();

protected:

// Returns the FOM interaction class name to use.
// Default implementation is to ask the fomMapper.
virtual char* interactionClassToUse(
    DtExerciseConn* exConn) const;
```



```

// Returns a pointer to a private pvList that can be used as
// a return value by phvps. It's guaranteed to be able to fit
// at least numParameters() parameters.
virtual RTI::ParameterHandleValuePairSet* privPvList() const;

// Get rid of constness on the "this" pointer in order to access
// non-const functions from const member functions (such as
simple
// accessors). This should be used sparingly since it bypasses
// the const protection.
// Note: This is essentially like an operator and is
//       non-virtual since can't override based on return type.
//       Allows derived classes to also define this member.
DtInteraction* self() const;

protected:

    DtString myTag;
    DtInterClassDesc* myClassDesc;
    DtExerciseConn* myExConn;
    DtTime myTime;
    DtTimeStampType myTimeType;

protected:

    static DtBoolean theTagIsAsciiTime;
    static RTI::ParameterHandleValuePairSet* thePvList;
    static int thePvListMaxSize;
    static const int theMaxInteractionClasses;
};
#include "intClassDesc.h"

#endif

```

hInterFactory.h

```
#if DtHLA

#include "vllist.h"
#include "RTI.hh"
#include "MachineTypes.h"

#include "chooseDefines.h"

// class DtInteractionFactory:
//
// Instances of DtInteractionFactory are used to create derived
// DtInteractions based on Interaction Class.

class DT_DLL_VL DtInteractionFactory
{
public:

    // Interaction creation function signature.
    typedef DtInteraction* (*DtInteractionCreator)();

public:

    // constructor
    DtInteractionFactory(DtExerciseConn* exConn);

    // destructor
    virtual ~DtInteractionFactory();

    // copy constructor
    DtInteractionFactory(const DtInteractionFactory& orig);

    // assignment operator
    DtInteractionFactory& operator=(const DtInteractionFactory&
orig);

    // Create the specified interaction.
    // Note: Caller is responsible for deleting.
    // Returns NULL if interaction creation function is not in the
factory.
    virtual DtInteraction* createInteraction(
        RTI::InteractionClassHandle handle,
        const RTI::ParameterHandleValuePairSet& pairSet);

    // Add an interaction creation function to the factory.
    // The function must return a new'ed interaction object which
will
    // be deleted by the user.
```

```
    // Returns any previous interaction creation function which was
    // replaced by the add. Second version registers the creator by
name,
    // rather than by handle.
    virtual DtInteractionCreator addCreator(
        RTI::InteractionClassHandle handle,
        DtInteractionCreator creator);
    virtual DtInteractionCreator addCreator(
        const char* className,
        DtInteractionCreator creator);

protected:

    // myCreators is an array of DtInteraction::theMaxInteraction-
Classes
    // items.
    DtInteractionCreator* myCreators;
    DtExerciseConn* myExConn;
};
#endif
```

hInterWData.h

```
#if DtHLA

#include "hIntWEncDec.h"
#include "hostStructs.h"
#include "mathTypes.h"
#include "datumRecSet.h"

#include "chooseDefines.h"

// class DtInteractionWithData:
//
// Instances of DtInteractionWithData are used to represent
// RPR FOM Interactions that use fixed and var datum records.

class DT_DLL_VL DtInteractionWithData : public DtInteractionWith-
EncDec
{
public:

    // default constructor
    DtInteractionWithData();

    // destructor
    virtual ~DtInteractionWithData();

    // copy constructor
    DtInteractionWithData(const DtInteractionWithData& orig);

    // assignment operator
    DtInteractionWithData& operator=(
        const DtInteractionWithData& orig);

    // Print the interaction's data.
    virtual void printData() const;

    virtual int numFixedFields() const;

    // returns number of variable length fields
    virtual int numVarFields() const;

    // Returns the datum param found in the index'th variable
    // datum record. Index is between 1 and numRecords. Returns
    // DtDatumParam(0) if index is out of bounds.
    virtual DtDatumParam varDatumId(int index) const;

    // Returns the datum param found in the index'th fixed
    // datum record. Index is between 1 and numRecords. Returns
```

```

// DtDatumParam(0) if index is out of bounds.
virtual DtDatumParam fixedDatumId(int index) const;

// Returns the datum param found in the index'th fixed or variable
// datum record, depending on whether fixedOrVar is DtFixed or
DtVar.
// Index is between 1 and numRecords. Returns
// DtDatumParam(0) if index is out of bounds.
virtual DtDatumParam datumId(DtDatumRecordType fixedOrVar, int
index) const;

// Returns the number of bits of data contained in the value
// fields of the index'th fixed or variable datum record,
// depending on whether fixedOrVar is DtFixed or DtVar. If
DtFixed,
// the answer is always 32.
virtual int datumSizeBits(DtDatumRecordType fixedOrVar, int
index) const;

// Returns the number of bytes of data contained in the value
// fields of the index'th fixed or variable datum record,
// depending on whether fixedOrVar is DtFixed or DtVar. If
DtFixed,
// the answer is always 4.
virtual int datumSizeBytes(DtDatumRecordType fixedOrVar, int
index) const;

// Looks through all fixed and variable records for one
containing
// param as the parameter ID. If it finds one, fills supplied
fixedOrVar
// pointer with DtFixed or DtVar, and index with the index to
indicate
// which record was found, and fills retcode with DtDatumValid.
If param
// was not found, retcode is filled with DtDatumBadId, and index
and
// fixedOrVar are invalid.
virtual void getIndex(DtDatumParam param, DtDatumRecordType*
fixedOrVar,
int* index, DtDatumRetCode* retcode = NULL) const;

// ***** INSPECTING DATUM VALUES BY INDEX *****
// If you know the data type of the value contained in a particular
// record, use one of the following to return it. fixedOrVar
(either DtFixed
// or DtVar) and index (between 1 and numRecords) indicate which
record

```

```
// you are inspecting. datumValByteArray is used when the data
// type is
// a character string, or something other than a long, unsigned
// long, float
// or double. It returns a pointer to a character string containing
// the
// exact bytes in the network representation that represents the
// value
// being inspected. The supplied retcode pointer will be filled
// with
// DtDatumValid if the value was successfully inspected, and DtDa-
// tumBadLength
// if either the index was invalid, or the size of the data indi-
// cated that
// it could not be interpreted as the chosen data type.
```

```
virtual DtFloat32 datumValFloat32(DtDatumRecordType fixedOrVar,
int index,
    DtDatumRetCode* retcode = NULL) const;
virtual DtFloat64 datumValFloat64(DtDatumRecordType fixedOrVar,
int index,
    DtDatumRetCode* retcode = NULL) const;
virtual unsigned long datumValUnsigned32(DtDatumRecordType fixe-
dOrVar,
    int index, DtDatumRetCode* retcode = NULL) const;
virtual long datumValInt32(DtDatumRecordType fixedOrVar, int
index,
    DtDatumRetCode* retcode = NULL) const;
virtual const char* datumValByteArray(DtDatumRecordType fixe-
dOrVar,
    int index) const;
```

```
// ***** INSPECTING DATUM VALUES BY PARAM ID *****
// If you know the data type of the value contained in a particular
// record, use one of the following to return it. The particular
// record
// inspected will be the first one containing param as the datum ID.
// datumValByteArray is used when the data type is
// a character string, or something other than a long, unsigned
// long, float
// or double. It returns a pointer to a character string containing
// the
// exact bytes in the network representation that represents the
// value
// being inspected. The supplied retcode pointer will be filled
// with
// DtDatumValid if the value was successfully inspected, and DtDa-
// tumBadLength
// if either param was not found, or the size of the data indicated
// that
```

```

// it could not be interpreted as the chosen data type.

virtual DtFloat32 datumValFloat32(DtDatumParam param,
    DtDatumRetCode* retcode = NULL) const;
virtual DtFloat64 datumValFloat64(DtDatumParam param,
    DtDatumRetCode* retcode = NULL) const;
virtual unsigned long datumValUnsigned32(DtDatumParam param,
    DtDatumRetCode* retcode = NULL) const;
virtual long datumValInt32(DtDatumParam param,
    DtDatumRetCode* retcode = NULL) const;
virtual const char* datumValByteArray(DtDatumParam param,
    DtDatumRetCode* retcode = NULL) const;

// ***** MUTATOR FUNCTIONS *****

    // Initialize to reflect the fixed parameters
    // and variable parameters and sizes indicated. fixedParams is a
    NULL-
    // terminated array of DtDatumParams. specList is an array of
    DtVarDatumSpecs
    // terminated with the DtNullVarDatumSpec constant. This call
    can be made
    // instead of calls to setNumVarDatumRecords, setNumFixedDatum-
    Record,
    // and repeated calls to setDatumParam and setVarDataBytes.
    virtual void initDataSection(const DtDatumParam* fixedParams,
        const DtVarDatumSpec* specList);

    // Sets the number of fixed or variable fields to newNum. Previ-
    ously
    // existing fixed fields with indices greater than newNum are
    deleted.
    // If newNum exceeds the number of existing fields, trailing new
    fields
    // are created with zeros as datum params, sizes (for var only)
    and values.
    virtual void setNumFixedFields(int newNum);
    virtual void setNumVarFields(int newNum);

    // Set the datum ID in the index'th fixed or variable datum record
    to
    // param, depending on whether fixedOrVar is DtFixed or DtVar.
    Supplied
    // retcode pointer is filled with DtDatumValid if successful, or
    DtDatumBadIndex
    // if the index is not valid.
    virtual void setDatumParam(DtDatumRecordType fixedOrVar, int
    index, DtDatumParam param,
        DtDatumRetCode* retcode = NULL);

```

```
// Allocates length bytes for the index'th variable datum record.
// Should be called before using one of the setDatumVal functions
// to actually set the data. Supplied retcode pointer is set to
// DtDatumValid if successful, or DtDatumBadIndex if the index
// is not valid.
virtual void setVarDataBytes(int index, int length, DtDatumRet-
Code* retcode = NULL);

// ***** SETTING DATA VALUES BY INDEX *****
// Use one of the following to set the data value in a fixed or vari-
able datum
// record, depending on the data type of your data. fixedOrVar
(either DtFixed
// or DtVar) and index (between 1 and numRecords) indicate which
record
// you are modifying. setDatumValByteArray is used when the data
type is
// a character string, or something other than a long, unsigned
long, float
// or double. It takes a pointer to a character string containing
the
// exact bytes that should be used in the network representation.
// The supplied retcode pointer will be filled with
// DtDatumValid if the value was successfully set, DtDatumBadIndex
if the
// index is invalid, and DtDatumBadLength if the size of the data
indicated that
// it could not be set from as the chosen data type.

virtual void setDatumValFloat64(DtDatumRecordType fixedOrVar, int
index, DtFloat64 val,
DtDatumRetCode* retcode = NULL);
virtual void setDatumValFloat32(DtDatumRecordType fixedOrVar, int
index, DtFloat32 val,
DtDatumRetCode* retcode = NULL);
virtual void setDatumValInt32(DtDatumRecordType fixedOrVar, int
index, long val,
DtDatumRetCode* retcode = NULL);
virtual void setDatumValUnsigned32(DtDatumRecordType fixedOrVar,
int index,
unsigned long val, DtDatumRetCode* retcode = NULL);
virtual void setDatumValByteArray(DtDatumRecordType fixedOrVar,
int index,
const char* val, DtDatumRetCode* retcode = NULL);

// ***** SETTING DATA VALUES BY PARAM ID *****
// Use one of the following to set the data value in a fixed or vari-
able datum
// record, depending on the data type of your data. The particular
record modified
```

```

// will be the first one containing param as its parameter ID.
// setDatumValByteArray is used when the data type is
// a character string, or something other than a long, unsigned
// long, float
// or double. It takes a pointer to a character string containing
// the
// exact bytes that should be used in the network representation.
// The supplied retcode pointer will be filled with
// DtDatumValid if the value was successfully set, DtDatumBadId if
// param
// is invalid, and DtDatumBadLength if the size of the data indi-
// cated that
// it could not be set from the chosen data type.

    virtual void setDatumValFloat64(DtDatumParam param, DtFloat64
val,
        DtDatumRetCode* retcode = NULL);
    virtual void setDatumValFloat32(DtDatumParam param, DtFloat32
val,
        DtDatumRetCode* retcode = NULL);
    virtual void setDatumValInt32(DtDatumParam param, long val,
        DtDatumRetCode* retcode = NULL);
    virtual void setDatumValUnsigned32(DtDatumParam param, unsigned
long val,
        DtDatumRetCode* retcode = NULL);
    virtual void setDatumValByteArray(DtDatumParam param, const char*
val,
        DtDatumRetCode* retcode = NULL);

    // Return the DtDatumRecordSet which an InteractionWithData
    // uses to store its fixed and var records. In general,
    // the wrapper functions above should be used rather than going
    // directly
    // through the DtDatumRecordSet, which is necessary only when
    // doing
    // encoding and decoding of network representations.
    virtual DtDatumRecordSet* datumRecSet();
    virtual const DtDatumRecordSet* datumRecSet() const;

protected:

    DtDatumRecordSet myRecSet;
};
#endif

```

hIntWEncDec.h

```
#if DtHLA

#include "RTI.hh"
#include "hInteraction.h"

#include "chooseDefines.h"

// class DtInteractionWithEncDec:
//
// DtInteractionWithEncDec is a DtInteraction that uses an encoder
// and
// decoder to implement the virtual functions phvps and
// setFromPhvps.

class DT_DLL_VL DtInteractionWithEncDec : public DtInteraction
{
public:

    // default constructor
    DtInteractionWithEncDec();

    // destructor
    virtual ~DtInteractionWithEncDec();

    // copy constructor
    DtInteractionWithEncDec(const DtInteractionWithEncDec& orig);

    // assignment operator
    DtInteractionWithEncDec& operator=(const DtInteractionWithEncDec&
orig);

    // Sets the interaction's ExerciseConn and class descriptor. If
    // classDesc is NULL, a default is chosen by calling
    // interactionClassToUse(). Without being told what ExConn or
class
    // descriptor to use, many of the HLA-specific functions won't
    // work properly. However, it is usually not necessary for
application
    // code to call this function, since it is called by
    // DtExerciseConn::send on the outgoing side, and by the
    // DtInteractionFactory on the incoming side.
    virtual void setExConn(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc = NULL);

    // Get the parameter handle value pair set for the interaction.
    virtual const RTI::ParameterHandleValuePairSet& phvps() const;
```

```

// Set contents from a parameter handle value pair set
virtual void setFromPhvps(
    const RTI::ParameterHandleValuePairSet& pvList);

// Specify an encoder, decoder, and interaction data repository
// to use in encoding and decoding. Called by setExConn (using
// encoder and decoder obtained through encoderToUse() and
// decoderToUse()) so call AFTER setExConn if you want to
// override defaults. Caller is responsible for deleting enc,
dec and
// data, but should do so only after they are no longer used by
the
// interaction.
virtual void useThese(DtInteractionEncoder* enc,
    DtInteractionDecoder* dec,
    DtInteractionRep* data);

// Return the encoder we are using
virtual DtInteractionEncoder* encoder() const;

// Return the decoder we are using
virtual DtInteractionDecoder* decoder() const;

protected:

    // Called by decoderToUse to obtain a new'ed decoder. Default
    // implmentation is to ask the FOM Mapper. The object is automat-
ically
    // deleted in the interaction's destructor.
    virtual DtInteractionDecoder* createDecoder(
        DtExerciseConn* exConn) const;

    // Called by encoderToUse to obtain a new'ed encoder. Default
    // implmentation is to ask the FOM Mapper. The object is automat-
ically
    // deleted in the interaction's destructor.
    virtual DtInteractionEncoder* createEncoder(
        DtExerciseConn* exConn) const;

    // Returns a decoder to use for decoding the interaction.
    virtual DtInteractionDecoder* decoderToUse(
        DtExerciseConn* exConn) const;

    // Returns an encoder to use for encoding the interaction.
    virtual DtInteractionEncoder* encoderToUse(
        DtExerciseConn* exConn) const;

protected:

    DtInteractionDecoder* myDecoder;

```

```
DtInteractionEncoder* myEncoder;  
DtInteractionDecoder* myCreatedDecoder;  
DtInteractionEncoder* myCreatedEncoder;  
DtInteractionRep* myData;  
};  
#endif
```

hlaObj.h

```

#if DtHLA

#include "RTI.hh"
#include "MachineTypes.h"
#include "remObjCbInfo.h"
#include "vlTime.h"
#include "objRefCbInfo.h"
#include "vlprint.h"
#include "objClassDesc.h"

#include "chooseDefines.h"

// class DtHlaObject:
//
// Instances of DtHlaObject are used to represent HLA objects (as
// opposed to interactions) which are communicated across the
// network.

class DT_DLL_VL DtHlaObject
{
    // DtReflectedHlaObjectManager is a friend
    // so that it can manage these objects.
    friend class DtReflectedHlaObjectManager;

public:

    DtHlaObject(DtExerciseConn* conn, RTI::ObjectHandle id,
                const char* name,
                DtObjClassDesc* classDesc);

    // constructor
    DtHlaObject(DtExerciseConn* conn, const char* name,
                DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtHlaObject();

    // Get the object's identifier.
    virtual const RTI::ObjectHandle& objectId() const;

    // Get the object's identifier.
    virtual const char* name() const;

    // Get the object's class handle.
    virtual RTI::ObjectClassHandle objectClassHandle() const;

    // Get the number of attributes associated with this object.

```

```
virtual int numAttributes() const;

// Get the attribute handle set.
virtual const RTI::AttributeHandleSet& attrHandleSet();

// Get the exercise connection for this object.
virtual DtExerciseConn* exerciseConn() const;

// Get the class descriptor for the object's class
virtual DtObjClassDesc* classDesc();

//
// Local Object specific functions.
//

// Do the initialization required to begin publishing this
object.
virtual void setupUpdating();

// Update (publish) this object on the network. Called by
publisher's tick
// functions to generate and send needed data. Default implemen-
tation is
// to call the pure virtual generateUpdate, and then send the
generated
// message.
virtual void update();

// Called by update to actually generate an outgoing message.
Derived
// class implementations should keep track of which attributes
are updated
// when, should check update conditions, and add values for only
those
// attributes that currently need to be sent. subscribedAttrs is
the set
// of attributes to which remote federates have subscribed.
requiredAttrs
// is the set of attributes that have been requested or forced by
// application code since the attribute was last sent. stateMsg
is the
// message to fill out. Default implementation is NULL, but the
function
// is not pure virtual since some subclasses might want to over-
ride
// update rather than overriding generateUpdate.
virtual void generateUpdate(
    const RTI::AttributeHandleSet& subscribedAttrs,
    const RTI::AttributeHandleSet& requiredAttrs,
    DtStateMsg* stateMsg);
```

```
// Do the deinitialization required to stop publishing this
object.
virtual void cleanupFromUpdating();

// Set yourself up so that on the next tick, you send an update
// for the appropriate attributes. (Can't send them within this
// function since this is called from within an RTI service).
virtual void processProvideUpdate(
    const RTI::AttributeHandleSet& theAttributes);

// returns the union of all sets of attributes requested since
the
// last call to clearRequestedAttributes.
virtual const RTI::AttributeHandleSet& requestedAttributes();

// clears the set of requested attributes. Should be called
after
// request has been serviced.
virtual void clearRequestedAttributes();

// Specify a set of attributes that should be included in the
ahvps the
// next time update is called, even if they would not otherwise
need to be
// sent. The call with no arguments, indicates the set of all of
the
// object's published attributes. Multiple calls are cumulative,
until the
// set is cleared with clearForcedAttributes, which should happen
within
// update.
virtual void forceUpdate();
virtual void forceUpdate(const RTI::AttributeHandleSet& set);

// Clear (empty) the set of forced attributes.
virtual void clearForcedAttributes();

// returns the set of attributes that are currently needed by the
// federation, i.e. those that have been specified in startUp-
dates
// calls, but not those that have been specified in stopUpdates.
virtual const RTI::AttributeHandleSet& attributesNeededByFedera-
tion();

// returns DtTrue if the class is needed by the federation,
// i.e. we've received a startRegistration or startUpdates
// service call from the RTI for the object's class.
DtBoolean classNeededByFederation();
```

```
// If the class has never been published, publish it with all
attributes
// using DtObjClassDesc::publish.
virtual void setupClassForUpdating();

//
// Remote Object specific functions.
//

// Do the initialization required to handle reflection of object.
virtual void setupReflecting();

// Calls reflect, and any postReflect callbacks
// Override reflect rather than this function
// to change the way an HLA object reflects a stateMsg.
virtual void callReflect(const DtStateMsg& stateMsg);

// Do the deinitialization required to stop handling reflection
of object.
virtual void cleanupFromReflecting();

// Request an update for the object for the set of attributes
indicated by
// attrs (or all subscribed attributes if attrs is NULL).
virtual void requestUpdate(RTI::AttributeHandleSet* attrs =
NULL);

// Add (register) function to be called when object is removed.
virtual void addRemoveObjectCb(DtRemoveObjectCb cb,
    void* usr);

// Remove (deregister) function to be called when object is
removed.
virtual void removeRemoveObjectCb(DtRemoveObjectCb cb,
    void* usr);

// Add (register) function to be called after each reflectAttr-
Values
virtual void addPostReflectCb(DtObjectReflectAttributeValuesCb
cb,
    void* usr);

// Remove (deregister) function to be called after each reflec-
tAttrValues
virtual void removePostReflectCb(DtObjectReflectAttributeValuesCb
cb,
    void* usr);

virtual DtTime lastSimTimeUpdateReceived() const;
```

```

    // Prints contents of a state msg.  Base class prints only header
    info.
    virtual void printMsg(const DtStateMsg& msg) const;

public:

    // Get maximum number of object class handles supported
    static int maxObjectClasses();

protected:

    // copy constructor
    DtHlaObject(const DtHlaObject& orig);

    // assignment operator
    DtHlaObject& operator=(const DtHlaObject& orig);

    // Process notification of removal of the object from the
    network.
    // Call back all the remove object callbacks.
    virtual void processRemoveObject();

    // Handle reflection of object (receipt of state message) from
    the network.
    // called by callReflect.  Override this function rather than
    callReflect
    // to change the way an HLA object reflects a stateMsg.
    virtual void reflect(const DtStateMsg& stateMsg) = 0;

    // Remove all callbacks from the remove callback list.
    // Non-virtual since meant to be called from dtor.
    void wipeRemoveList();

    // Virtual version of above
    virtual void clearRemoveList();

    // Remove all callbacks from the reflect callback list.
    // Non-virtual since meant to be called from dtor.
    void wipeReflectList();

    // Copy another HLA object's remove callback list into this
    // object's remove callback list.
    // Non-virtual since meant to be called from ctor.
    void copyRemoveList(const DtHlaObject& orig);

    // Set the time an update was last received or sent.
    virtual void setLastSimTimeUpdateReceived(DtTime time);

    // apply callbacks after a state msg is reflected

```

```
virtual void applyPostReflectCallbacks(const DtStateMsg&
stateMsg);

// Callback for handling turnUpdatesOnForObjectInstance RTI call.
virtual void processUpdatesOn(
    RTI::ObjectHandle hand,
    const RTI::AttributeHandleSet& set);

// Callback for handling turnUpdatesOffForObjectInstance RTI
call.
virtual void processUpdatesOff(
    RTI::ObjectHandle hand,
    const RTI::AttributeHandleSet& set);

protected:

// Callback for handling state messages for the HLA object.
// Casts hlaObj to DtHlaObject* and calls reflect().
static void reflectAttributeValuesCb(const DtStateMsg& stateMsg,
    void* hlaObj);

// Callback for handling turnUpdatesOnForObjectInstance RTI call.
static void DtHlaObject::updatesOnCb(
    RTI::ObjectHandle hand,
    const RTI::AttributeHandleSet& attributeSet,
    void* usr);

// Callback for handling turnUpdatesOffForObjectInstance RTI
call.
static void DtHlaObject::updatesOffCb(
    RTI::ObjectHandle hand,
    const RTI::AttributeHandleSet& attributeSet,
    void* usr);

protected:

char* myName;
// Needed attributes is a per-object concept in RTI1.3
RTI::AttributeHandleSet* myNeededAttrs;

DtExerciseConn* myExConn;
RTI::ObjectHandle myObjectId;
DtBoolean myReflecting;
DtBoolean myUpdating;
DtList myRemoveCbList;
DtTime myLastSimTimeUpdateReceived;
DtList myReflectCbList;
DtObjClassDesc* myClassDesc;

// Requested by other federates.
```

```
RTI::AttributeHandleSet* myRequestedAttributes;

// Requested by a call to setForcedUpdate.
RTI::AttributeHandleSet* myForcedAttributes;

// Union of forced and requested - not computed until needed.
RTI::AttributeHandleSet* myRequiredAttributes;

protected:

    static const int theMaxObjectClasses;
};
#endif
```

hlaObjFact.h

```
#if DtHLA

#include "hlaObj.h"

#include "chooseDefines.h"

// class DtHlaObjectFactory:
//
// Instances of DtHlaObjectFactory are used to create HLA objects.

class DT_DLL_VL DtHlaObjectFactory
{
public:

#if DtRTI103
    // HLA object creation function signature.
    typedef DtHlaObject* (*DtHlaObjectCreator)
        (DtExerciseConn* conn, RTI::ObjectHandle id, DtObjClassDesc*
cDesc);
#else
    // HLA object creation function signature.
    typedef DtHlaObject* (*DtHlaObjectCreator)
        (DtExerciseConn* conn, RTI::ObjectHandle id,
        const char* name, DtObjClassDesc* cDesc);
#endif

public:

    // constructor
    DtHlaObjectFactory(DtExerciseConn* conn);

    // destructor
    virtual ~DtHlaObjectFactory();

    // copy constructor
    DtHlaObjectFactory(const DtHlaObjectFactory& orig);

    // assignment operator
    DtHlaObjectFactory& operator=(const DtHlaObjectFactory& orig);

#if DtRTI103
    // Create the specified object.
    // Note: Caller is responsible for deleting.
    // Returns NULL if object creation function is not in the
factory.
    virtual DtHlaObject* createHlaObject(
        RTI::ObjectClassHandle handle,
```

```
        RTI::ObjectHandle id);
#else
    // Create the specified object.
    // Note: Caller is responsible for deleting.
    // Returns NULL if object creation function is not in the
factory.
    virtual DtHlaObject* createHlaObject(
        RTI::ObjectClassHandle handle,
        RTI::ObjectHandle id,
        const char* name);
#endif

    // Add an object creation function to the factory.
    // The function must return a new'ed object which will
    // be deleted by the user.
    // Returns any previous object creation function which was
    // replaced by the add.
    virtual DtHlaObjectCreator addCreator(RTI::ObjectClassHandle
handle,
        DtHlaObjectCreator creator);

protected:
    // myCreators is an array of DtHlaObject::theMaxObjectClasses
items.
    DtHlaObjectCreator* myCreators;
    DtExerciseConn* myConn;
};
#endif
```

hlaObjWithSR.h

```
#if DtHLA

#include "hlaObj.h"

#include "chooseDefines.h"

// class DtHlaObjectWithStateRep:
//
// Instances of DtHlaObjectWithStateRep are used to represent an
// HLA object via a state representation.

class DT_DLL_VL DtHlaObjectWithStateRep : public DtHlaObject
{
public:

#if DtRTI103
    // constructor
    DtHlaObjectWithStateRep(DtExerciseConn* conn, RTI::ObjectHandle
id,
        DtObjClassDesc* classDesc);
#else
    // constructor
    DtHlaObjectWithStateRep(DtExerciseConn* conn, RTI::ObjectHandle
id,
        const char* name,
        DtObjClassDesc* classDesc);
    // constructor
    DtHlaObjectWithStateRep(DtExerciseConn* conn, const char* name,
        DtObjClassDesc* classDesc);
#endif

    // destructor
    virtual ~DtHlaObjectWithStateRep();

    // copy constructor
    DtHlaObjectWithStateRep(const DtHlaObjectWithStateRep& orig);

    // assignment operator
    DtHlaObjectWithStateRep& operator=(const DtHlaObjectWithStateRep&
orig);
    //
    // Local Entity specific functions.
    //

    // Set the state repositories (local representation and remote
    // reflection), and the encoder and decoder which are used to
    // send and receive HLA state messages from/into the repository.
```

```

    virtual void useTheseForUpdating(DtStateRepository* stateRep,
        DtStateRepository* asSeenByRemote,
        DtHlaStateEncoder* encoder,
        DtHlaStateDecoder* decoder);

    // Called by update() to actually generate an outgoing message.
    This
    // class's implementation uses an encoder, and stateRepositories
    to perform
    // this function.  subscribedAttrs is the set of attributes to
    which remote
    // federates have subscribed.  requiredAttrs is the set of
    attributes that
    // have been requested or forced by application code since the
    attribute
    // was last sent.  stateMsg is the message to fill out.
    virtual void generateUpdate(
        const RTI::AttributeHandleSet& subscribedAttrs,
        const RTI::AttributeHandleSet& requiredAttrs,
        DtStateMsg* stateMsg);

    //
    // Remote Entity specific functions.
    //

    // Set the state repository for holding a remote reflection
    // and the decoder which is used to receive HLA state messages
    // into the repository.
    virtual void useTheseForReflecting(DtStateRepository* stateRep,
        DtHlaStateDecoder* decoder);

    // Receive representation (remote reflection) from the network.
    virtual void reflect(const DtStateMsg& stateMsg);

    // Prints contents of a state msg.  Base class prints only header
    info.
    virtual void printMsg(const DtStateMsg& msg) const;

public:

#if DtRTI103
    // Creator function for this class
    static DtHlaObject* create(DtExerciseConn* conn,
        RTI::ObjectHandle id,
        DtObjClassDesc* classDesc);
#else
    // Creator function for this class
    static DtHlaObject* create(DtExerciseConn* conn,
        RTI::ObjectHandle id,
        const char* name,

```

```
        DtObjClassDesc* classDesc);
#endif

protected:

    DtHlaStateDecoder* myDecoder;
    DtHlaStateEncoder* myEncoder;
    DtStateRepository* myStateRep;
    DtStateRepository* myAsSeenByRemote;
};
#if DtRTI103
#else
#endif
#endif
```


hLcDec.h

```

#ifdef DtHLA

#include "interDecoder.h"

// This can be used to declare parameter processor functions
// which are static members of a class.

#define DtDECLARE_LGR_CONTROL_PARAM_DECODER(paramName) \
    DtDECLARE_PARAM_DECODER(DtLgrControlInteraction, paramName)

// class DtLgrControlDecoder:
//
// Instances of DtLgrControlDecoder are used to decode LgrControl
// Interactions
// from the network.

class DT_DLL_VL DtLgrControlDecoder : public DtInteractionDecoder
{
public:

    // constructor
    DtLgrControlDecoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtLgrControlDecoder();

    // copy constructor
    DtLgrControlDecoder(const DtLgrControlDecoder& orig);

    // assignment operator
    DtLgrControlDecoder& operator=(const DtLgrControlDecoder& orig);

protected:

    // Individual parameter decoders

    DtDECLARE_LGR_CONTROL_PARAM_DECODER(LgrControlPdu);
};
#endif

```

hLcEnc.h

```
#if DtHLA

#include "interEncoder.h"

#define DtDECLARE_LGR_CONTROL_PARAM_ENCODER(paramName) \
    DtDECLARE_PARAM_ENCODER(DtLgrControlInteraction, paramName)

#define DtDEFINE_SIMPLE_LGR_CONTROL_PARAM_ENCODER(paramName, \
    netType, inspector) \
    DtDEFINE_SIMPLE_PARAM_ENCODER(DtLgrControlEncoder, DtLgrCon- \
    trolInteraction, \
    paramName, netType, inspector)

// class DtLgrControlEncoder:
//
// DtLgrControlEncoder is a DtInteractionEncoder that registers
// parameter encoders for all of the LGR_CONTROL interaction param- \
// eters.

class DT_DLL_VL DtLgrControlEncoder : public DtInteractionEncoder
{
public:

    // add parameter encoder "encodeLgrControlPdu"
    DtLgrControlEncoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtLgrControlEncoder();

    // copy constructor
    DtLgrControlEncoder(const DtLgrControlEncoder& orig);

    // assignment operator
    DtLgrControlEncoder& operator=(const DtLgrControlEncoder& orig);

protected:

    // Individual parameter encoders

    DtDECLARE_LGR_CONTROL_PARAM_ENCODER(LgrControlPdu);
};
#endif
```

hLcInter.h

```

#if DtHLA

#include "chooseDefines.h"
#include "hIntWEncDec.h"
#include "lcPdu.h"

// class DtLgrControlInteraction:
//
// Instances of DtLgrControlInteraction are used to represent
// LgrControl Interactions in HLA.

// LgrControl interaction callback function signature.
typedef void (*DtLgrControlInteractionCb)(DtLgrControlInteraction*
inter,
void* usr);

class DT_DLL_VL DtLgrControlInteraction : public DtInteractionWith-
EncDec
{
public:

    // default constructor
    DtLgrControlInteraction();

    // destructor
    virtual ~DtLgrControlInteraction();

    // copy constructor
    DtLgrControlInteraction(const DtLgrControlInteraction& orig);

    // assignment operator
    DtLgrControlInteraction& operator=(const DtLgrControlInteraction&
orig);

    // Print the interaction's data.
    virtual void printData() const;

    // Get/Set a logger control PDU
    virtual const DtLgrControlPdu* lcPdu() const;
    virtual void setLcPdu(const DtLgrControlPdu& pdu);
    virtual void setFromNetPacket(const DtNetPacket* pack);

    // Name of the interaction
    virtual const char* name() const;

public:

```

```
// Create a DtLgrControlInteraction.
// Caller is responsible for deletion.
static DtInteraction* create();

// Add (register)/Remove (deregister) function to be called when
// interaction occurs.
static void addCallback(DtExerciseConn* conn,
    DtLgrControlInteractionCb cb, void* usr);
static void removeCallback(DtExerciseConn* conn,
    DtLgrControlInteractionCb cb, void* usr);

protected:

    DtLgrControlPdu* myLcPdu;
};
#endif
```

hObjPub.h

```

#if DtHLA

#ifndef DtObjectPublisher_H_
#define DtObjectPublisher_H_

#include "chooseDefines.h"
#include "RTI.hh"
#include "hostStructs.h"
#include "hlaObjWithSR.h"

// class DtObjectPublisher:
//
// Instances of DtObjectPublisher are used for local or source simu-
// lations
// of objects, (i.e. these objects are not dependent on data being
// produced elsewhere on the network for their existence). These
// object send data to the network.

class DT_DLL_VL DtObjectPublisher
{
public:

    // Constructor. Object is not fully constructed until setHlaOb-
    // ject is
    // called - typically by subclass constructor.
    DtObjectPublisher(DtExerciseConn* conn,
        DtStateRepository* stateRep,
        DtStateRepository* asSeenByRemote);

    // Destructor
    virtual ~DtObjectPublisher();

    // A DtObjectPublisher is not capable of communicating data
    // through the RTI
    // until setHlaObject is called. It is typically called by
    // subclass
    // constructors. If encoder or decoder are NULL, the FOM Mapper
    // is
    // consulted. This class assumes ownership of the
    // encoder/decoder objects
    // passed to setHlaObject, and will delete them on destruction.
    // If
    // objectName is NULL, a name is obtained from the RTI.
    virtual void setHlaObject(
        const char* className,
        DtHlaStateEncoder* encoder = NULL,
        DtHlaStateDecoder* decoder = NULL,

```

```
    const char* objectName = NULL);

    // Same as above, but you pass object class name rather than
    handle.
    virtual void setHlaObject(
        RTI::ObjectClassHandle hand,
        DtHlaStateEncoder* encoder = NULL,
        DtHlaStateDecoder* decoder = NULL,
        const char* objectName = NULL);

    // Detach HLA object - bringing the DtObjectPublisher back to the
    state it
    // was in before setHlaObject was called. Typically called only
    by
    // destructor.
    virtual void detachHlaObject();

    // Returns a pointer to the DtStateRepository - through
    // which you can set or inspect current state information.
    virtual DtStateRepository* stateRep();

    // Synonym for stateRep()
    virtual DtStateRepository* sr();

    // Returns a pointer to the Decoder being used to decode
    // state messages that we send back into the asSeenByRemote
    // SR.
    virtual DtHlaStateDecoder* decoder();

    // Returns a pointer to the Encoder being used to encode
    // current state into an outgoing message.
    virtual DtHlaStateEncoder* encoder();

    // Get the object's hlaObject
    virtual DtHlaObject* hlaObject();

    // Get our exercise connection
    virtual DtExerciseConn* exerciseConn();

    // Get the object's local identifier.
    virtual const DtObjectId& id() const;

    // Get the object's local identifier.
    virtual const DtObjectId& objectId() const;

    // Get the object's local identifier.
    virtual const DtObjectId& localId() const;

    // Get the object's name.
    virtual const DtString& name() const;
```

```
// Get the HLA object's global object designator -
// the object name in RTI1.3.3, but the objectId in RTI1.0.3.
virtual const DtGlobalObjectDesignator& globalId() const;

// Specify a set of attributes that should be included in the
ahvps the
// next time tick is called, even if they would not otherwise need
to be
// sent. The call with no arguments, indicates the set of all of
the
// object's attributes. Multiple calls are cumulative, until the
set is
// cleared with clearForcedAttributes, which should happen within
tick.
virtual void forceUpdate();
virtual void forceUpdate(const RTI::AttributeHandleSet& set);

// Tick (update) this object.
virtual void tick();

protected:

// copy constructor
DtObjectPublisher(const DtObjectPublisher& orig);

// assignment operator
DtObjectPublisher& operator= (const DtObjectPublisher& orig);

protected:

DtExerciseConn* myExConn;
DtHlaObjectWithStateRep* myHlaObj;
DtStateRepository* myStateRep;
DtStateRepository* myAsSeenByRemote;
DtHlaStateEncoder* myEncoder;
DtHlaStateDecoder* myDecoder;
DtObjectId myObjectId;
DtString myName;

public:

// These constructors and init functions are obsolete, but are
here for
// backwards compatibility with pre-3.4 releases. Please use the
// constructor and setHlaObject functions described at the top of
this
// class definition instead.
DtObjectPublisher(DtExerciseConn* conn, const char* name = NULL);
```

```
void init(RTI::ObjectClassHandle hand, DtStateRepository*
stateRep,
    DtStateRepository* asSeenByRemote, DtHlaStateEncoder* encoder
= NULL,
    DtHlaStateDecoder* decoder = NULL);
void init(const char* className, DtStateRepository* stateRep,
    DtStateRepository* asSeenByRemote, DtHlaStateEncoder* encoder
= NULL,
    DtHlaStateDecoder* decoder = NULL);
};
#endif    //DtObjectPublisher_H_
```


hostStructs.h

```

#include "chooseDefines.h"
#include "MachineTypes.h"
#include "NetStructs.h"
#include "disEnums.h"
#include <string.h>
#include <stdio.h>
#include "globalObjDes.h"

#if DtHLA
#include "RTI.hh"
#endif

#if DtHLA && DtRTI103
#define ObjectHandle ObjectID
#endif

typedef DtU16 DtExerciseId;

enum
{
    DtkObjectIsUnknown = -1,
    DtkObjectIsDis      = 0,
    DtkObjectIsHla      = 1
};

/*for backwards compatibility */
typedef struct DtSimulationAddress
{
    DtU16    DtSite;
    DtU16    DtHost;
} DtSimulationAddress;

#include "eventId.h"

#if DtHLA

class DT_DLL_VL DtFederateId
{
public:
    DtFederateId();
    DtFederateId(const char* str);
    DtFederateId(int site, int app);
    DtFederateId(const DtNetFederateId& netId);

    const DtFederateId& operator=(const DtFederateId &netId);
    const DtNetFederateId& operator=(const DtNetFederateId &netId);

```

```
int operator==(const DtFederateId &otherGuy) const;
int operator!=(const DtFederateId &otherGuy) const;

void init(int site, int app);

operator DtNetFederateId () const;

int siteId() const;
void setSiteId(int siteId);

int applicationId() const;
void setApplicationId(int id);

char *string() const;

static const DtFederateId& nullId();

protected:
    int mySiteId;
    int myAppId;
};

#endif // DtHLA

#if !i386
const DtU16 DtDeferredEntityId = ((1<<16)-1);
#else
const DtU16 DtDeferredEntityId = 0xff; /* CJW: THIS IS A BUG */
#endif

class DT_DLL_VL DtEntityIdentifier
{
public:
    DtEntityIdentifier(const char* str);
    DtEntityIdentifier(int site, int host, int entityNum);
    DtEntityIdentifier(DtNetEntityIdentifier netId);
    DtEntityIdentifier();

    void init(int site, int host, int entityNum);

    void setFromNet(DtNetEntityIdentifier netId)
    {
        init(netId.DtSimulator.DtSite, netId.DtSimulator.DtHost,
            netId.DtEntity);
    }

    operator DtNetEntityIdentifier () const
    {
```

```

    DtNetEntityIdentifier id;

    id.DtSimulator.DtSite = site();
    id.DtSimulator.DtHost = host();
    id.DtEntity = entityNum();
    return(id);
}

int site() const {return (int) DtSimulator.DtSite;}
int host() const {return (int) DtSimulator.DtHost;}
int entityNum() const {return (int) DtEntity;}

void setSite(int site) {DtSimulator.DtSite = site;}
void setHost(int host) {DtSimulator.DtHost = host;}
void setEntityNum(int entityNum) {DtEntity = entityNum;}

int sameHost(const DtEntityIdentifier &otherGuy) const;
int sameEntity(const DtEntityIdentifier &otherGuy) const;
int matchPattern(const DtEntityIdentifier &pat) const;

int operator==(const DtEntityIdentifier &otherGuy) const{return
sameEntity(otherGuy);}
int operator!=(const DtEntityIdentifier &otherGuy)
const{if(sameEntity(otherGuy)){return 0;}else{return 1;}}

char *string() const;

static const DtEntityIdentifier& nullId();

public:
    DtSimulationAddress DtSimulator;
    DtU16 DtEntity;
};

#if !DTHLA
typedef DtEntityIdentifier DtObjectId;
#else
class DT_DLL_VL DtObjectId
{
public:

    DtObjectId();
    DtObjectId(RTI::ObjectHandle id);
    DtObjectId(const char* str);
    DtObjectId& operator=(const DtObjectId& orig);
    DtObjectId(const DtObjectId& orig);

#if DT_LITTLE_ENDIAN
    // On big endian machines, this is the same
    // as DtObjectId(RTI::ObjectHandle id); defined above.

```

```
DtObjectId(DtNetObjectId netId);
#endif // DT_LITTLE_ENDIAN
operator RTI::ObjectHandle() const;

char *string() const;
static const DtObjectId& nullId();

protected:

    RTI::ObjectHandle    myId;
};
#endif

#define DtAny -1

class DT_DLL_VL DtEntityType
{
public:
    DtEntityType(const char* p);
    DtEntityType(int kind, int domain, int country, int category,
        int subCategory, int specific, int extra);
    DtEntityType();
    DtEntityType(DtNetEntityType netType);
    void setFromNet(DtNetEntityType netType)
    {
        init(netType.DtEntityKind, netType.DtDomain,
            netType.DtCountry,
            netType.DtCategory, netType.DtSubCategory, netType.DtSpe-
            cific,
            netType.DtExtra);
    }

    operator DtNetEntityType() const;

    void init(int kind, int domain, int country, int category,
        int subCategory, int specific, int extra);
    char *string() const;

    static const DtEntityType& nullType();

public:
    int DtEntityKind; /* this conflicts with enum of same name, so
must
    be before funcs that use it. Leave for awhile
    back-compat for awhile. */
    int DtDomain;
    int DtCountry;
    int DtCategory;
    int DtSubCategory;
    int DtSpecific;
```

```

    int DtExtra;

public:
/*  int entityKind() const {return(DtEntityKind);}    */
    int kind() const {return(DtEntityKind);}
    int domain() const {return(DtDomain);}
    int country() const {return(DtCountry);}
    int category() const {return(DtCategory);}
    int subCategory() const {return(DtSubCategory);}
    int specific() const {return(DtSpecific);}
    int extra() const {return(DtExtra);}

/*  void setEntityKind(int kind){DtEntityKind = kind;} */
    void setKind(int kind){DtEntityKind = kind;}
    void setDomain(int domain){DtDomain = domain;}
    void setCountry(int country){DtCountry = country;}
    void setCategory(int category){DtCategory = category;}
    void setSubCategory(int subCategory){DtSubCategory = subCategory;}
    void setSpecific(int specific){DtSpecific = specific;}
    void setExtra(int extra){DtExtra = extra;}

    /* the funcs used to take longs, so for back-compat: */
    void setKind(long kind){DtEntityKind = (int)kind;}
    void setDomain(long domain){DtDomain = (int)domain;}
    void setCountry(long country){DtCountry = (int)country;}
    void setCategory(long category){DtCategory = (int)category;}
    void setSubCategory(long subCategory){DtSubCategory =
(int)subCategory;}
    void setSpecific(long specific){DtSpecific = (int)specific;}
    void setExtra(long extra){DtExtra = (int)extra;}

    DtBoolean matchPattern(const DtEntityType &pat) const;
    DtBoolean DtTypeMatchPattern(const DtEntityType &pat) const
    { return matchPattern(pat); }

    /* == returns 1 if they are equal, 0 if they are not equal */
    int match(const DtEntityType &type) const
    {return operator==(type);}
    int operator==(const DtEntityType &type) const;

    /* != returns 0 if they are equal, nonzero if they are not equal
    */
    int operator!=(const DtEntityType &type) const;
};

class DT_DLL_VL DtRadioEntityType : public DtEntityType
{
public:
    DtRadioEntityType();

```

```
DtRadioEntityType(const char* p);
DtRadioEntityType(int kind, int domain, int country, int category,
    int nomenclatureVersion, int nomenclature);
DtRadioEntityType(DtNetRadioEntityType netType);
void setFromNet(DtNetRadioEntityType netType);

int nomenclatureVersion() const;
int nomenclature() const;

void setNomenclatureVersion(int version);
void setNomenclature(int nom);

char *string() const;

static const DtRadioEntityType& nullType();

operator DtNetRadioEntityType() const;
};

class DT_DLL_VL DtBurstDescriptor
{
public:
    DtBurstDescriptor()
    {
        DtEntityType type;
        init(&type, DtWarheadOther, DtFuzeOther, 0, 0);
    }

    DtBurstDescriptor(DtNetBurstDescriptor *netBurst)
    {
        DtEntityType type(netBurst->DtMunition);
        init(&type,
            DtWarheadType((int)netBurst->DtWarhead),
            DtDetonatorFuze((int)netBurst->DtFuze),
            (int) netBurst->DtQuantity,
            (int) netBurst->DtRate);
    }

    DtBurstDescriptor(DtEntityType *type, DtWarheadType warhead,
        DtDetonatorFuze fuze, DtU16 quantity, DtU16 rate)
    {
        init(type, warhead, fuze, quantity, rate);
    }

    operator DtNetBurstDescriptor () const
    {
        DtNetBurstDescriptor b;
        b.DtMunition = DtMunition;
        b.DtWarhead = DtWarhead;
    }
};
```

```
        b.DtFuze = DtFuze;
        b.DtQuantity = DtQuantity;
        b.DtRate = DtRate;
        return(b);
    }

    void init(DtEntityType *type, DtWarheadType warhead,
              DtDetonatorFuze fuze, int quantity, int rate)
    {
        DtMunition = *type;
        DtWarhead = warhead;
        DtFuze = fuze;
        DtQuantity = quantity;
        DtRate = rate;
    }

public:
    DtEntityType      DtMunition;
    DtWarheadType     DtWarhead;
    DtDetonatorFuze   DtFuze;
    int                DtQuantity;
    int                DtRate;
};
```

hRadioRcvrPub.h

```
#if DtHLA

#ifndef DtRadioReceiverPublisher_H_
#define DtRadioReceiverPublisher_H_

#include "hObjPub.h"

// class DtRadioReceiverPublisher:
//
// Instances of DtRadioReceiverPublisher are used for local or
// source
// simulations of radio receivers, (i.e. these entities are not
// dependent
// on data being produced elsewhere on the network for their exist-
// ence).
// These radio receivers send data to the network.

class DT_DLL_VL DtRadioReceiverPublisher : public DtObjectPublisher
{
public:

    // Function prototype for functions that create
    // DtRadioReceiverRepositories
    typedef DtRadioReceiverRepository* (*DtStateRepCreator)();

public:

    // constructor
    DtRadioReceiverPublisher(RTI::ObjectClassHandle h,
        DtExerciseConn* conn, const char* name = NULL);

    // constructor
    DtRadioReceiverPublisher(DtExerciseConn* conn, const char* name =
    NULL);

    // destructor
    virtual ~DtRadioReceiverPublisher();

    // Returns a pointer to the DtRadioReceiverRepository - through
    // which you can set or inspect current state information.
    virtual DtRadioReceiverRepository* receiverRep();

    // Synonym for receiverRep()
    virtual DtRadioReceiverRepository* rsr();

    // Returns the name of the FOM object class to use. Default
    implementation
}
```



```
// is to ask the FOM Mapper.
virtual const char* classToUse() const;

public:

    // Give us a function to be used for creating SRs.
    static DtStateRepCreator setStateRepCreator(DtStateRepCreator
creator);

protected:

    // copy constructor
    DtRadioReceiverPublisher(const DtRadioReceiverPublisher& orig);

    // copy constructor
    DtRadioReceiverPublisher& operator=
        (const DtRadioReceiverPublisher& orig);

protected:

    static DtStateRepCreator theStateRepCreator;
};
#endif    //DtRadioReceiverPublisher_H_
```

hRadioXmitPub.h

```
#if DtHLA

#ifndef DtRadioTransmitterPublisher_H_
#define DtRadioTransmitterPublisher_H_

#include "hObjPub.h"

// class DtRadioTransmitterPublisher:
//
// Instances of DtRadioTransmitterPublisher are used for local or
// source
// simulations of radio transmitters, (i.e. these entities are not
// dependent
// on data being produced elsewhere on the network for their exist-
// ence).
// These radio transmitters send data to the network.

class DT_DLL_VL DtRadioTransmitterPublisher : public DtObjectPub-
lisher
{
public:

    // Function prototype for functions that create
    // DtRadioTransmitterRepositories
    typedef DtRadioTransmitterRepository* (*DtStateRepCreator)();

public:

    // constructor
    DtRadioTransmitterPublisher(RTI::ObjectClassHandle h,
        DtExerciseConn* conn, const char* name = NULL);

    // constructor
    DtRadioTransmitterPublisher(DtExerciseConn* conn, const char*
name = NULL);

    // destructor
    virtual ~DtRadioTransmitterPublisher();

    // Returns a pointer to the DtRadioTransmitterRepository -
    through
    // which you can set or inspect current state information.
    virtual DtRadioTransmitterRepository* radioXmitRep();

    // Synonym for radioXmitRep()
    virtual DtRadioTransmitterRepository* tsr();
};

#endif
```

```
    // Returns the name of the FOM object class to use.  Default
implementation
    // is to ask the FOM Mapper.
    virtual const char* classToUse() const;

public:

    // Give us a function to be used for creating SRs.
    static DtStateRepCreator setStateRepCreator(DtStateRepCreator
creator);

protected:

    // copy constructor
    DtRadioTransmitterPublisher(const DtRadioTransmitterPublisher&
orig);

    // copy constructor
    DtRadioTransmitterPublisher& operator=
        (const DtRadioTransmitterPublisher& orig);

protected:

    static DtStateRepCreator theStateRepCreator;
};
#endif    //DtRadioTransmitterPublisher_H_
```

hRawBinRSDec.h

```
#if DtHLA

#include "chooseDefines.h"
#include "interDecoder.h"

// This can be used to declare parameter processor functions
// which are static members of a class.

#define DtDECLARE_RAW_BIN_RADIO_SIGNAL_PARAM_DECODER(paramName) \
    DtDECLARE_PARAM_DECODER(DtSignalInteraction, paramName)

#define DtDEFINE_SIMPLE_RAW_BIN_RADIO_SIGNAL_PARAM_DECODER(param- \
    Name, \
        netType, mutator) \
    DtDEFINE_SIMPLE_PARAM_DECODER(DtRawBinaryRadioSignalDecoder, \
        DtSignalInteraction, paramName, netType, mutator)

#define
DtDEFINE_SIMPLE_RAW_BIN_RADIO_SIGNAL_PARAM_DECODER_WITH_CAST( \
    paramName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_PARAM_DECODER_WITH_CAST( \
        DtRawBinaryRadioSignalDecoder, DtSignalInteraction, \
        paramName, netType, mutator, castExpr)

// class DtRawBinaryRadioSignalDecoder:
//
// Instances of DtRawBinaryRadioSignalDecoder are

class DT_DLL_VL DtRawBinaryRadioSignalDecoder : public DtInterac-
tionDecoder
{
public:

    // default constructor
    DtRawBinaryRadioSignalDecoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtRawBinaryRadioSignalDecoder();

    // copy constructor
    DtRawBinaryRadioSignalDecoder(
        const DtRawBinaryRadioSignalDecoder& orig);

    // assignment operator
    DtRawBinaryRadioSignalDecoder& operator=(
        const DtRawBinaryRadioSignalDecoder& orig);
```

protected:

```
DtDECLARE_RAW_BIN_RADIO_SIGNAL_PARAM_DECODER(HostRadioID);
DtDECLARE_RAW_BIN_RADIO_SIGNAL_PARAM_DECODER(DataRate);
DtDECLARE_RAW_BIN_RADIO_SIGNAL_PARAM_DECODER(SignalDataLength);
DtDECLARE_RAW_BIN_RADIO_SIGNAL_PARAM_DECODER(SignalData);
DtDECLARE_RAW_BIN_RADIO_SIGNAL_PARAM_DECODER(TacticalDataLink-
Type);
    DtDECLARE_RAW_BIN_RADIO_SIGNAL_PARAM_DECODER(TDLMessageCount);
};
#endif    // DtHLA
```

hRawBinRSEnc.h

```
#if DtHLA

#include "chooseDefines.h"
#include "interEncoder.h"

#define DtDECLARE_RAW_BINARY_RADIO_SIGNAL_PARAM_ENCODER(paramName)
\
    DtDECLARE_PARAM_ENCODER(DtSignalInteraction, paramName)

#define
DtDEFINE_SIMPLE_RAW_BINARY_RADIO_SIGNAL_PARAM_ENCODER(paramName, \
    netType, inspector) \
    DtDEFINE_SIMPLE_PARAM_ENCODER(DtRawBinaryRadioSignalEncoder, \
    DtSignalInteraction, paramName, netType, inspector)

// class DtRawBinaryRadioSignalEncoder:
//
// Instances of DtRawBinaryRadioSignalEncoder are

class DT_DLL_VL DtRawBinaryRadioSignalEncoder : public DtInterac-
tionEncoder
{
public:

    // default constructor
    DtRawBinaryRadioSignalEncoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtRawBinaryRadioSignalEncoder();

    // copy constructor
    DtRawBinaryRadioSignalEncoder(const DtRawBinaryRadioSignalEn-
coder& orig);

    // assignment operator
    DtRawBinaryRadioSignalEncoder& operator=(
        const DtRawBinaryRadioSignalEncoder& orig);

protected:

    DtDECLARE_RAW_BINARY_RADIO_SIGNAL_PARAM_ENCODER(HostRadioID);
    DtDECLARE_RAW_BINARY_RADIO_SIGNAL_PARAM_ENCODER(DataRate);
    DtDECLARE_RAW_BINARY_RADIO_SIGNAL_PARAM_ENCODER(SignalDataL-
ength);
    DtDECLARE_RAW_BINARY_RADIO_SIGNAL_PARAM_ENCODER(SignalData);
```

```
    DtDECLARE_RAW_BINARY_RADIO_SIGNAL_PARAM_ENCODER(Tactical-  
DataLinkType);  
    DtDECLARE_RAW_BINARY_RADIO_SIGNAL_PARAM_ENCODER(TDLMessage-  
Count);  
};  
#endif    // DtHLA
```

hRefBeamList.h

```
#if DtHLA

#include "hRefObjList.h"
#include "hRefBeam.h"

// class DtReflectedEmitterBeamList:
//
// Instances of DtReflectedEmitterBeamList manage the list of
// of remote EmitterBeams in the exercise.

class DT_DLL_VL DtReflectedEmitterBeamList : public DtReflectedOb-
jectList
{
public:

    // Constructor.
    DtReflectedEmitterBeamList(DtExerciseConn* exConn);

    // Constructor - only subscribe to the RTI::ObjectClassHandles
    // in handleList
    DtReflectedEmitterBeamList(
        DtExerciseConn* exConn, const DtList *handleList);

    // destructor
    virtual ~DtReflectedEmitterBeamList();

    // Look up beam by identifier.
    virtual DtReflectedEmitterBeam* lookup(RTI::ObjectHandle id);

#if !DtRTI103
    // Look up beam by name.
    virtual DtReflectedEmitterBeam* lookup(
        const DtGlobalObjectDesignator& objName);
#endif

    // Get the first/last beam in the list.
    virtual DtReflectedEmitterBeam* first() const;
    virtual DtReflectedEmitterBeam* last() const;

    // EmitterBeam added callback registration/deregistration
    virtual void addEmitterBeamAdditionCallback(
        DtEmitterBeamCallbackFunction cb, void* userData);
    virtual void removeEmitterBeamAdditionCallback(
        DtEmitterBeamCallbackFunction cb, void* userData);

    // EmitterBeam removed callback registration/deregistration
    virtual void addEmitterBeamRemovalCallback(
```



```

        DtEmitterBeamCallbackFunction cb, void* userData);
virtual void removeEmitterBeamRemovalCallback(
        DtEmitterBeamCallbackFunction cb, void* userData);

protected:

    // copy constructor
    DtReflectedEmitterBeamList(const DtReflectedEmitterBeamList&
orig);

    // assignment operator
    DtReflectedEmitterBeamList& operator=(
        const DtReflectedEmitterBeamList& orig);

    // Override of virtual function. Here, we create a DtReflectedE-
mitterBeam.
    virtual DtReflectedObject* newReflectedObject(DtHlaObject* obj)
const;

    // Called from processDiscoverObject() after a remote beam has
    // been added to the list. Default implementation prints a
message.
    // Subclasses can override this function to add functionality on
    // addition. obj will be an instance of a class derived from
    // DtReflectedObj.
    virtual void objectAdded(DtReflectedObject* obj);

    // Returns a DtList of RTI::ObjectClassHandles that represent the
list
    // of object classes to which this list should subscribe.
    virtual const DtList* dfltHandleList();
};
#endif    // DtHLA

```

hRefDesigList.h

```
#if DtHLA

#include "hRefObjList.h"
#include "hRefDesigntr.h"

// class DtReflectedDesignatorList:
//
// Instances of DtReflectedDesignatorList manage the list of
// of remote Designators in the exercise.

class DT_DLL_VL DtReflectedDesignatorList : public DtReflectedObjectList
{
public:

    // Constructor - obtains the set of object classes to subscribe
    // to from the exConn's fomMapper.
    DtReflectedDesignatorList(DtExerciseConn* exConn);

    // Constructor - only subscribe to the RTI::ObjectClassHandles
    // in handleList
    DtReflectedDesignatorList(
        DtExerciseConn* exConn, const DtList *handleList);

    // destructor
    virtual ~DtReflectedDesignatorList();

    // Look up designator by identifier.
    virtual DtReflectedDesignator* lookup(RTI::ObjectHandle id);

#if !DtRTI103
    // Look up designator by name.
    virtual DtReflectedDesignator* lookup(
        const DtGlobalObjectDesignator& objName);
#endif

    // Get the first/last designator in the list.
    virtual DtReflectedDesignator* first() const;
    virtual DtReflectedDesignator* last() const;

    // Designator added callback registration/deregistration
    virtual void addDesignatorAdditionCallback(
        DtDesignatorCallbackFunction cb, void* userData);
    virtual void removeDesignatorAdditionCallback(
        DtDesignatorCallbackFunction cb, void* userData);

    // Designator removed callback registration/deregistration
```

```
virtual void addDesignatorRemovalCallback(
    DtDesignatorCallbackFunction cb, void* userData);
virtual void removeDesignatorRemovalCallback(
    DtDesignatorCallbackFunction cb, void* userData);

protected:

    // copy constructor
    DtReflectedDesignatorList(const DtReflectedDesignatorList& orig);

    // assignment operator
    DtReflectedDesignatorList& operator=(const DtReflectedDesignator-
List& orig);

    // Override of virtual function. Here, we create a DtReflected-
Designator.
    virtual DtReflectedObject* newReflectedObject(DtHlaObject* obj)
const;

    // Called from processDiscoverObject() after a remote designator
has
    // been added to the list. Default implementation prints a
message.
    // Subclasses can override this function to add functionality on
    // addition. obj will be an instance of a class derived from
DtReflectedObj.
    virtual void objectAdded(DtReflectedObject* obj);

    // Returns a DtList of RTI::ObjectClassHandles that represent the
list
    // of object classes to which this list should subscribe.
    virtual const DtList* dfltHandleList();
};
#endif    // DtHLA
```

hRefEmSysList.h

```
#if DtHLA

#include "hRefObjList.h"
#include "hRefLEmitrSys.h"
#include "hRefBeamList.h"

// class DtReflectedEmitterSystemList:
//
// Instances of DtReflectedEmitterSystemList manage the list of
// of remote EmitterSystems in the exercise.

class DT_DLL_VL DtReflectedEmitterSystemList : public DtReflectedObjectList
{
public:

    // Constructor
    // If beamsUpdateSystems == DtTrue, then a system's post-update
callbacks
    // are called whenever one of the beams in the system is updated.
    DtReflectedEmitterSystemList(DtExerciseConn* exConn,
        DtBoolean beamsUpdateSystems = DtTrue);

    // Constructor - only subscribe to the RTI::ObjectClassHandles
    // in handleList
    DtReflectedEmitterSystemList(DtExerciseConn* exConn,
        const DtList *handleList, DtBoolean beamsUpdateSystems =
DtTrue);

    // destructor
    virtual ~DtReflectedEmitterSystemList();

    // Look up emitter system by identifier.
    virtual DtReflectedEmitterSystem* lookup(RTI::ObjectHandle id);

    // lookup an emitter from its object ID and system ID
    virtual DtReflectedEmitterSystem* lookup(
        const DtGlobalObjectDesignator& emittingEntity,
        int systemId) const;

#if !DtRTI103

    // Look up emitter system by name.
    virtual DtReflectedEmitterSystem* lookup(
        const DtGlobalObjectDesignator& objName);

#endif
#endif
```

```

// Get the first/last emitter system in the list.
virtual DtReflectedEmitterSystem* first() const;
virtual DtReflectedEmitterSystem* last() const;

// EmitterSystem added callback registration/deregistration
// Called when an emitter system is added due to discovery of
// a system OR beam.
virtual void addEmitterSystemAdditionCallback(
    DtEmitterSystemCallbackFunction cb, void* userData);
virtual void removeEmitterSystemAdditionCallback(
    DtEmitterSystemCallbackFunction cb, void* userData);

// EmitterSystem removed callback registration/deregistration
// Called when an emitter system is removed due to discovery of
// a system OR beam.
virtual void addEmitterSystemRemovalCallback(
    DtEmitterSystemCallbackFunction cb, void* userData);
virtual void removeEmitterSystemRemovalCallback(
    DtEmitterSystemCallbackFunction cb, void* userData);

protected:

    // copy constructor
    DtReflectedEmitterSystemList(const DtReflectedEmitterSystemList&
orig);

    // assignment operator
    DtReflectedEmitterSystemList& operator=
        (const DtReflectedEmitterSystemList& orig);

    // Process the discovery of a remote emitter system on the
network.
    // Can't use base version, since we need to account for systems
that
    // are already in the list by virtue of discovered beams.
    virtual void processDiscoverObject(DtHlaObject* obj);

    // Process the removal of a remote emitter system from the
network.
    // Removes the remote emitter system from this list.
    // Can't use base version, since we need to keep systems that
    // still have live beams.
    virtual void processRemoveObject(DtHlaObject* obj);

    // Called by similar callback functions registered with myBeam-
List
    // on beam additions/removals. Addition function adds the beam
to the

```

```
// set of orphan beams (since we don't know what system he's part
of),
// the removal function removes the beam's beamSR from his
system's
// systemSR. Update function calls the beam's system update
callbacks if
// beamsUpdateSystems == DtTrue in the reflected system list
constructor.
virtual void processEmitterBeamAddition(DtReflectedEmitterBeam*
beam);
virtual void processEmitterBeamUpdate(DtReflectedEmitterBeam*
beam);
virtual void processEmitterBeamRemoval(DtReflectedEmitterBeam*
beam);

// Check if any orphanBeams have non-NULL emittingSystemId. If
so,
// add them to their appropriate system (creating one if it
doesn't
// exist), and delete them from orphan list.
virtual void resolveOrphans();

// Override of virtual function. Here, we create a DtReflectedE-
mitterSystem.
virtual DtReflectedObject* newReflectedObject(DtHlaObject* obj)
const;

// Create a remote emitter system object for the HLA object.
// Caller is responsible for deletion.
virtual DtReflectedEmitterSystem* newReflectedEmitterSystem(
    const DtGlobalObjectDesignator& des) const;

// Called from processDiscoverObject() after a remote transmitter
has
// been added to the list. Default implementation prints a
message.
// Subclasses can override this function to add functionality on
// addition. obj will be an instance of a class derived from
DtReflectedObj.
virtual void objectAdded(DtReflectedObject* obj);

// Returns whether or not the object should be timed out.
// Assumes that timeout processing is enabled. Can't use base
version
// since we don't want to time out systems with live beams.
virtual DtBoolean shouldBeTimedOut(DtReflectedObject* object);

// Returns a DtList of RTI::ObjectClassHandles that represent the
list
// of object classes to which this list should subscribe.
```

```
    virtual const DtList* dfltHandleList();

protected:

    // Callback functions registered with myBeamList on beam additions, updates,
    // and removals. Calls processEmitterBeamAddition/Update/Removal.
    static void emitterBeamAdditionCb(
        DtReflectedEmitterBeam* beam, void* usr);
    static void emitterBeamUpdateCb(
        DtReflectedEmitterBeam* beam, void* usr);
    static void emitterBeamRemovalCb(
        DtReflectedEmitterBeam* beam, void* usr);

    // Callback that calls resolveOrphans.
    static void resolveOrphansPostDrainCb(void *usr);

protected:

    DtList myOrphanBeams;
    DtReflectedEmitterBeamList myBeamList;
    DtBoolean myBeamsUpdateSystems;
};
#endif    //DtHLA
```

hRef/AggList.h

```
#if DtHLA

#include "hRefObjList.h"
#include "hReflectedAgg.h"

// class DtReflectedAggregateList:
//
// Instances of DtReflectedAggregateList are used to manage remote
// aggregates
// from the network.

class DT_DLL_VL DtReflectedAggregateList : public DtReflectedObjectList
{
public:

    // Constructor - obtains the set of object classes to subscribe
    // to from the exConn's fomMapper.
    DtReflectedAggregateList(DtExerciseConn* exConn,
        DtBoolean smoothCapable = DtFalse);

    // Constructor - only subscribe to the RTI::ObjectClassHandles
    // in handleList
    DtReflectedAggregateList(DtExerciseConn* exConn,
        const DtList *handleList, DtBoolean smoothCapable = DtFalse);

    // destructor
    virtual ~DtReflectedAggregateList();

    // Look up aggregate by identifier.
    virtual DtReflectedAggregate* lookup(RTI::ObjectHandle id);

#if !DtRTI103

    // Look up aggregate by name.
    virtual DtReflectedAggregate* lookup(
        const DtGlobalObjectDesignator& objName);

#endif

    // Get the first/last entity in the list.
    virtual DtReflectedAggregate* first() const;
    virtual DtReflectedAggregate* last() const;

    // Aggregate added callback registration/deregistration
    virtual void addAggregateAdditionCallback(
        DtAggregateCallbackFunction cb, void* userData);

}
```



```

virtual void removeAggregateAdditionCallback(
    DtAggregateCallbackFunction cb, void* userData);

// Aggregate removed callback registration/deregistration
virtual void addAggregateRemovalCallback(
    DtAggregateCallbackFunction cb, void* userData);
virtual void removeAggregateRemovalCallback(
    DtAggregateCallbackFunction cb, void* userData);

protected:

    // copy constructor
    DtReflectedAggregateList(const DtReflectedAggregateList& orig);

    // assignment operator
    DtReflectedAggregateList& operator=(const DtReflectedAggregate-
List& orig);

    // Asks the FOM Mapper for the DtList of RTI::ObjectClassHandles
to use.
    virtual const DtList* dfltHandleList();

    // Override of virtual function. Calls newReflectedAggregate.
    virtual DtReflectedObject* newReflectedObject(DtHlaObject* obj)
const;

    // Create a remote aggregate object for the HLA object.
    // Caller is responsible for deletion.
    virtual DtReflectedAggregate* newReflectedAggregate(
        DtHlaObject* obj) const;

    // Virtual function override.
    // Calls removeAndDelete(DtReflectedAggregate*....)
    virtual void removeAndDelete(DtReflectedObject* object, char*
msg);

    // Default calls down to base class version, but can be over-
ridden.
    virtual void removeAndDelete(
        DtReflectedAggregate* aggregate, char* msg = NULL);

    // Called from processDiscoverObject() after a remote transmitter
has
    // been added to the list. Calls aggregateAdded.
    virtual void objectAdded(DtReflectedObject* obj);

    // Called by objectAdded (which is called from processDiscoverOb-
ject())
    // after a remote aggregate has been added to the list. Default

```

```
// implementation prints a message. Subclasses can override this
function
// to add functionality on addition.
virtual void aggregateAdded(DtReflectedAggregate* ent);

protected:

    DtBoolean mySmoothCapable;
};
#endif
```

```

# $ ;

#if DtHLA

#ifndef DtReflectedEmitterBeam_H_
#define DtReflectedEmitterBeam_H_

#include "emittBeamSR.h"
#include "hReflectedObj.h"

// Callback function that takes a DtReflectedObject and user data
typedef void (*DtEmitterBeamCallbackFunction)(
    DtReflectedEmitterBeam* obj, void* userData);

// class DtReflectedEmitterBeam:
//
// Instances of DtReflectedEmitterBeam represent EmitterBeam
// object simulated by remote applications. State data can be
// viewed
// through it 's state repository.

class DT_DLL_VL DtReflectedEmitterBeam : public DtReflectedObject
{
public:

    // Function prototype for functions that create DtEmitterBeamRe-
    // positories
    typedef DtEmitterBeamRepository* (*DtStateRepCreator)();

public:

    // constructor
    DtReflectedEmitterBeam(DtHlaObject* obj, DtExerciseConn* conn);

    // destructor
    virtual ~DtReflectedEmitterBeam();

    // Get the beam 's state repository.
    virtual DtEmitterBeamRepository* beamRep() const;

    // Synonym for beamRep()
    virtual DtEmitterBeamRepository* bsr() const;

    // Set beam state repository. Only necessary when you want to
    // use something other than the default, which is created inter-
    // nally.
    virtual void setBsr(DtEmitterBeamRepository* bsr);

    // Get the next remote beam.

```

```
virtual DtReflectedEmitterBeam* next() const;

// Get the previous remote beam.
virtual DtReflectedEmitterBeam* prev() const;

// Get the next remote beam.
// Will cycle back to the first remote beam.
virtual DtReflectedEmitterBeam* wrapNext() const;

// Get the previous remote beam.
// Will cycle back to the last remote beam.
virtual DtReflectedEmitterBeam* wrapPrev() const;

// beam state callback registration/deregistration
virtual void addPostUpdateCallback(
    DtEmitterBeamCallbackFunction cb, void* userData);

virtual void removePostUpdateCallback(
    DtEmitterBeamCallbackFunction cb, void* userData);

public:

    // Give us a function to be used for creating SRs.
    static DtStateRepCreator setStateRepCreator(DtStateRepCreator
creator);

protected:

    // copy constructor
    DtReflectedEmitterBeam(const DtReflectedEmitterBeam& orig);

    // assignment operator
    DtReflectedEmitterBeam& operator=(
        const DtReflectedEmitterBeam& orig);

protected:

    DtEmitterBeamRepository* myConstructedBsr;

protected:

    static DtStateRepCreator theStateRepCreator;
};
#endif //DtReflectedEmitterBeam_H_
```

hReflDesigntr.h

```

#if DtHLA

#include "designatorSR.h"
#include "hReflectedObj.h"

// Callback function that takes a DtReflectedObject and user data
typedef void (*DtDesignatorCallbackFunction)(
    DtReflectedDesignator* obj, void* userData);

// class DtReflectedDesignator:
//
// Instances of DtReflectedDesignator are

class DT_DLL_VL DtReflectedDesignator : public DtReflectedObject
{
public:

    // Function prototype for functions that create DtDesignatorRe-
    // positories
    typedef DtDesignatorRepository* (*DtStateRepCreator)();

public:

    // constructor
    DtReflectedDesignator(DtHlaObject* obj, DtExerciseConn* conn);

    // destructor
    virtual ~DtReflectedDesignator();

    // Get the designator's state repository.
    virtual DtDesignatorRepository* designatorRep() const;

    // Synonym for designatorRep()
    virtual DtDesignatorRepository* dsr() const;

    // Get the next remote designator.
    virtual DtReflectedDesignator* next() const;

    // Get the previous remote designator.
    virtual DtReflectedDesignator* prev() const;

    // Get the next remote designator.
    // Will cycle back to the first remote designator.
    virtual DtReflectedDesignator* wrapNext() const;

    // Get the previous remote designator.
    // Will cycle back to the last remote designator.

```

```
virtual DtReflectedDesignator* wrapPrev() const;

// designator state callback registration/deregistration
virtual void addPostUpdateCallback(
    DtDesignatorCallbackFunction cb, void* userData);

virtual void removePostUpdateCallback(
    DtDesignatorCallbackFunction cb, void* userData);

public:

    // Give us a function to be used for creating SRs.
    static DtStateRepCreator setStateRepCreator(DtStateRepCreator
creator);

protected:

    // copy constructor
    DtReflectedDesignator(const DtReflectedDesignator& orig);

    // assignment operator
    DtReflectedDesignator& operator=(const DtReflectedDesignator&
orig);

protected:

    static DtStateRepCreator theStateRepCreator;
};
#endif    //DtHLA
```

hReflectedAgg.h

```

#ifdef DtHLA

#include "aggregateSR.h"
#include "hReflectedObj.h"

// Callback function that takes a DtReflectedObject and user data
typedef void (*DtAggregateCallbackFunction)(
    DtReflectedAggregate* obj, void* userData);

// class DtReflectedAggregate:
//
// Instances of DtReflectedAggregate are used to represent a simulation
// aggregate reflected across a network (i.e. this is not an full
// simulation running on this machine).

class DT_DLL_VL DtReflectedAggregate : public DtReflectedObject
{
public:

    // Function prototype for functions that create DtAggregateStateRepositories
    typedef DtAggregateStateRepository* (*DtStateRepCreator)();

public:

    // constructor
    DtReflectedAggregate(DtHlaObject* obj, DtExerciseConn* conn);

    // destructor
    virtual ~DtReflectedAggregate();

    // Returns a pointer to the AggregateStateRepository - through
    // which you can inspect current state information.
    virtual DtAggregateStateRepository* aggregateStateRep() const;

    // Synonym for aggregateStateRep()
    virtual DtAggregateStateRepository* asr() const;

    // Get the next remote aggregate.
    virtual DtReflectedAggregate* next() const;

    // Get the previous remote aggregate.
    virtual DtReflectedAggregate* prev() const;

    // Get the next remote aggregate.
    // Will cycle back to the first remote aggregate.

```

```
virtual DtReflectedAggregate* wrapNext() const;

// Get the previous remote aggregate.
// Will cycle back to the last remote aggregate.
virtual DtReflectedAggregate* wrapPrev() const;

// Aggregate state callback registration/deregistration
virtual void addPostUpdateCallback(
    DtAggregateCallbackFunction cb, void* userData);
virtual void removePostUpdateCallback(
    DtAggregateCallbackFunction cb, void* userData);

public:

    // Give us a function to be used for creating SRs.
    static DtStateRepCreator setStateRepCreator(DtStateRepCreator
creator);

public:

    // Get the aggregate HLA object's id
    // For backwards compatibility. Use id() instead.
    virtual RTI::ObjectHandle aggregateObjectId() const;

protected:

    // copy constructor
    DtReflectedAggregate(const DtReflectedAggregate& orig);

    // assignment operator
    DtReflectedAggregate& operator=(const DtReflectedAggregate&
orig);

protected:

    static DtStateRepCreator theStateRepCreator;
};
#endif // DtHLA
```


hReflectedEnt.h

```

#ifdef DtHLA

#include "entitySR.h"
#include "hReflectedObj.h"

// Callback function that takes a DtReflectedObject and user data
typedef void (*DtEntityCallbackFunction)(
    DtReflectedEntity* obj, void* userData);

// class DtReflectedEntity:
//
// Instances of DtReflectedEntity are used to represent a simulation
// entity reflected across a network (i.e. this is not an full
// simulation running on this machine).

class DT_DLL_VL DtReflectedEntity : public DtReflectedObject
{
public:

    // Function prototype for functions that create DtEntityStateRe-
    // positories
    typedef DtEntityStateRepository* (*DtStateRepCreator)();

public:

    // constructor
    DtReflectedEntity(DtHlaObject* obj, DtExerciseConn* conn);

    // destructor
    virtual ~DtReflectedEntity();

    // Returns a pointer to the EntityStateRepository - through
    // which you can inspect current state information.
    virtual DtEntityStateRepository* entityStateRep() const;

    // Synonym for entityStateRep()
    virtual DtEntityStateRepository* esr() const;

    // Get the next remote entity.
    virtual DtReflectedEntity* next() const;

    // Get the previous remote entity.
    virtual DtReflectedEntity* prev() const;

    // Get the next remote entity.
    // Will cycle back to the first remote entity.
    virtual DtReflectedEntity* wrapNext() const;

```

```
// Get the previous remote entity.
// Will cycle back to the last remote entity.
virtual DtReflectedEntity* wrapPrev() const;

// Entity state callback registration/deregistration
virtual void addPostUpdateCallback(
    DtEntityCallbackFunction cb, void* userData);
virtual void removePostUpdateCallback(
    DtEntityCallbackFunction cb, void* userData);

public:

    // Give us a function to be used for creating SRs.
    static DtStateRepCreator setStateRepCreator(DtStateRepCreator
creator);

public:

    // For backwards compatibility. Use DtStateRepCreator instead.
    typedef DtStateRepCreator DtEsrCreator;

    // For backwards compatibility. Use id() instead.
    virtual RTI::ObjectHandle entityObjectId() const;

    // For backwards compatibility. Use setStateRepCreator instead.
    static DtEsrCreator setEsrCreator(DtEsrCreator creator);

protected:

    // copy constructor
    DtReflectedEntity(const DtReflectedEntity& orig);

    // assignment operator
    DtReflectedEntity& operator=(const DtReflectedEntity& orig);

protected:

    static DtStateRepCreator theStateRepCreator;
};
#endif // DtHLA
```

hReflectedObj.h

```

#ifdef DtHLA

#include "chooseDefines.h"
#include "vllist.h"
#include "vlTime.h"
#include "globalObjDes.h"
#include "hlaObjWithSR.h"
#include "refObjCbInfo.h"
#include "hostStructs.h"
#include "str.h"

// class DtReflectedObject:
//
// Instances of DtReflectedObject are used to represent a simulation
// entity reflected across a network (i.e. this is not an full
// simulation running on this machine).

class DT_DLL_VL DtReflectedObject : public DtListedObj
{
public:

    // Function prototype for functions that create DtStateRepositories
    typedef DtStateRepository* (*DtBaseStateRepCreator)();

public:

    // Primary constructor
    DtReflectedObject(DtExerciseConn* conn,
        DtStateRepository* stateRep);

    // Older constructor. Use the primary constructor followed by
    // setHlaObject instead.
    DtReflectedObject(DtHlaObject* obj,
        DtExerciseConn* conn, DtBaseStateRepCreator creator);

    // Older constructor. Use the primary constructor followed by
    // setNoHlaObject instead.
    DtReflectedObject(const DtObjectId& id,
        const DtGlobalObjectDesignator& des,
        DtExerciseConn* conn, DtBaseStateRepCreator creator);

    // destructor
    virtual ~DtReflectedObject();

    // Returns a pointer to the StateRepository - through

```

```
// which you can inspect current state information.
virtual DtStateRepository* stateRep() const;

// Synonym for stateRep()
virtual DtStateRepository* sr() const;

// Returns a pointer to the Decoder being used to decode
// state messages into our SR.
virtual DtHlaStateDecoder* decoder();

// Get the entity's hlaObject
virtual DtHlaObject* hlaObject() const;

// Get the HLA object's id
virtual const DtObjectId& objectId() const;

// Get the HLA object's id (local ID).
virtual const DtObjectId& id() const;

// Get the HLA object's id (local ID).
virtual const DtObjectId& localId() const;

// Get the HLA object name
virtual const DtString& name() const;

// Get the HLA object's global object designator -
// the object name in RTI1.3.3, but the objectId in RTI1.0.3.
virtual const DtGlobalObjectDesignator& globalId() const;

// Returns the value of simulation time at the time
// an update was last received for the entity.
virtual DtTime lastSimTimeUpdated() const;

// Object state callback registration/deregistration
virtual void baseAddPostUpdateCallback(
    DtReflectedObjectCallbackFunction cb, void* userData);
virtual void baseRemovePostUpdateCallback(
    DtReflectedObjectCallbackFunction cb, void* userData);

// Get the next remote object.
virtual DtReflectedObject* nextObj() const;

// Get the previous remote object.
virtual DtReflectedObject* prevObj() const;

// Get the next remote object.
// Will cycle back to the first remote object.
virtual DtReflectedObject* wrapNextObj() const;

// Get the previous remote object.
```

```
// Will cycle back to the last remote object.
virtual DtReflectedObject* wrapPrevObj() const;

// Start reflecting this object, using this decoder. If decoder
is NULL,
// we get one from the FOM Mapper. The decoder is deleted by the
// DtReflectedObject destructor. Usually called by derived class
// constructor.
virtual void setHlaObject(DtHlaObject* obj,
    DtHlaStateDecoder* decoder = NULL);

// Used to set up a proxy DtReflectedEntity - one that is not
actually
// managing a real reflected HLA object.
virtual void setNoHlaObject(const DtObjectId& id,
    const DtGlobalObjectDesignator& des);

// Ends managing of current HLA object.
virtual void detachHlaObject();

protected:

// copy constructor
DtReflectedObject(const DtReflectedObject& orig);

// assignment operator
DtReflectedObject& operator=(const DtReflectedObject& orig);

// Request that the owner of this entity send an attribute update
virtual void requestUpdate();

// process the post update callbacks
virtual void processPostUpdateCallbacks();

protected:

// calls requestUpdate
static void requestUpdatePostDrainCb(void* usr);

// callback from HlaObject to process post update callbacks
static void processPostUpdateCallbacksCb(
    const DtStateMsg& stateMsg,
    DtHlaObject* obj,
    void* usr);

protected:

DtExerciseConn* myConn;
DtHlaObjectWithStateRep* myHlaObj;
DtStateRepository* myStateRep;
```

```
DtHlaStateDecoder* myDecoder;  
DtBoolean myRequestCbRegistered;  
DtIntrusiveList myStateUpdateCallbacks;  
DtObjectId myId;  
DtString myName;  
};  
#endif
```

hRefObjList.h

```

#if DtHLA

#include "chooseDefines.h"
#include "objIdHashLst.h"
#include "RTI.hh"
#include "MachineTypes.h"
#include "vlTime.h"
#include "globalObjDes.h"
#include "refObjCbInfo.h"

// class DtReflectedObjectList:
//
// Instances of DtReflectedObjectList are list of instances of
// classes
// derived from DtReflectedObject. A DtReflectedObjectList manages
// the comings
// and goings of its reflected objects based on discoveries and
// removes
// received from the RTI.

class DT_DLL_VL DtReflectedObjectList
{
public:

    // Constructor
    DtReflectedObjectList(DtExerciseConn* exConn);

    // destructor
    virtual ~DtReflectedObjectList();

    // Look up object by identifier.
    virtual DtReflectedObject* lookupObj(RTI::ObjectHandle id);

    // Look up object by global ID (which is the object name in
    RTI1.3.3).
    virtual DtReflectedObject* lookupObj(
        const DtGlobalObjectDesignator& objName);

    // Get the first/last object in the list.
    virtual DtReflectedObject* firstObj() const;
    virtual DtReflectedObject* lastObj() const;

    // Get number of objects
    virtual int count() const;

    // Get my exerciseConn.
    virtual DtExerciseConn *exerciseConn() const;

```

```
// Register callbacks with exercise connection so that we get
// notified when an object of this class is discovered. Automat-
ically
// called for all object classes passed to the constructor, or
for
// the default set of objects classes if the ctor without a
handleList
// is used. But if you want to handle your own subscription and
// update requests, pass an empty handleSet to the constructor,
// use DtObjectClassDesc's registerInterest and requestUpdates,
// then use this function to tell the ROL to manage the objects
that
// get discovered.
virtual void manageObjectClass(RTI::ObjectClassHandle handle);

// Have the application act as though we had never discovered the
// object. We call the RTI's localDelete service, and delete the
// DtHlaObject, which causes any DtReflectedObjects associated
with
// id (in this list and in any others) to be removed. The object
// might be re-discovered later.
virtual void removeObject(const DtObjectId& id);

// Object added callback registration/deregistration
virtual void addObjectAdditionCallback(
    DtReflectedObjectCallbackFunction cb, void* userData);
virtual void removeObjectAdditionCallback(
    DtReflectedObjectCallbackFunction cb, void* userData);

// Object removed callback registration/deregistration
virtual void addObjectRemovalCallback(
    DtReflectedObjectCallbackFunction cb, void* userData);
virtual void removeObjectRemovalCallback(
    DtReflectedObjectCallbackFunction cb, void* userData);

// Print the SRs of all objects
virtual void print();

// Set whether or not timeout processing occurs.
// Defaults to off.
virtual void setTimeoutProcessing(DtBoolean onOff);

// Set/Get the remote object timeout period.
virtual void setTimeoutInterval(DtTime time);
virtual DtTime timeoutInterval();

public:

    // Set/Get flag that determines whether we request updates
```



```

// from all objects of the object classes we subscribe to.
// Default is true.
static void setRequestClassUpdateFlag(DtBoolean onOff);
static DtBoolean requestClassUpdateFlag();

// Set/Get flag that determines whether we request updates
// from an object as we discover it. Default is true.
static void setRequestObjectUpdateFlag(DtBoolean onOff);
static DtBoolean requestObjectUpdateFlag();

// Post-drain callback that calls findAlreadyDiscoveredObjects()
static void findDiscoveredObjectsCb(void* usr);

protected:

// copy constructor
DtReflectedObjectList(const DtReflectedObjectList& orig);

// assignment operator
DtReflectedObjectList& operator=(const DtReflectedObjectList&
orig);

// Process the discovery of a remote object on the network.
// Adds the remote object to this list.
virtual void processDiscoverObject(DtHlaObject* obj);

// Process the removal of a remote object from the network.
// Removes the remote object from this list.
virtual void processRemoveObject(DtHlaObject* obj);

// Handle the receipt of attributes for a remote object which
// was timed out. Reinstates the remote object in this list.
virtual void processTimedOutObjectReflect(const DtStateMsg& msg,
DtHlaObject* obj);

// Remove any remote objects which have not been updated during
// the timeout period. Gets called automatically by a postDrain-
Callback
// registered by the constructor. Users should not need to call
this.
virtual void doTimeouts();

// Create a remote object object for the HLA object.
// Caller is responsible for deletion.
virtual DtReflectedObject* newReflectedObject(DtHlaObject* obj)
const = 0;

// Remove the remote object from the list if found and delete it.
// Prints a message on deletion if specified.
virtual void removeAndDelete(DtReflectedObject* obj,

```

```
    char* msg = NULL);

    // Called from processDiscoverObject() after a remote object has
    // been added to the list.
    // Defaults to do nothing.
    // Subclasses can override this function to add functionality on
    // addition.
    virtual void objectAdded(DtReflectedObject* object);

    // Returns whether or not the object should be timed out.
    // Assumes that timeout processing is enabled.
    virtual DtBoolean shouldBeTimedOut(DtReflectedObject* object);

    // Times out object by removing it from the remote object list and
    // deleting it. Manages callbacks to reinstate remote object
    // upon receiving attribute updates for the timed out object.
    virtual void timeOutObject(DtReflectedObject* object);

    // Manages callbacks and data used for reinstating a remote
    object
    // which was timed out.
    virtual void unTimeOutObject(DtHlaObject* obj);

    // Register interest in this FOM object class. Typically called
    by derived
    // class constructor.
    virtual void addObjectClass(RTI::ObjectClassHandle handle);

    // Register interest in this FOM object class by className.
    Typically
    // called by derived class constructor.
    virtual void addObjectClassByName(const char* className);

    // Register interest in a whole list of RTI::ObjectClassHandles
    all at
    // once. Uses addObjectClass above.
    virtual void init(const DtList* handleList);

    // Empty the object list and the timeout list.
    // Non-virtual since meant to be called from destructor.
    void emptyLists();

    // Process object addition/removal callbacks
    virtual void processObjectAdditionCallbacks(
        DtReflectedObject* object);
    virtual void processObjectRemovalCallbacks(
        DtReflectedObject* object);

    // When list is created, we call this function to go through the
    list of
```

```

    // all discovered HLA objects to see if any should be managed by
    this
    // list. Otherwise, this list only hears about objects that are
    discovered
    // by the app after the creation of this list.
    virtual void findAlreadyDiscoveredObjects();

```

protected:

```

    // Cast the void* to a DtReflectedObjectList* and calls
    // processDiscoverObject().
    static void discoverObjectCb(DtHlaObject* hlaObj, void* rel);

    // Cast the void* to a DtReflectedObjectList* and calls
    // processRemoveObject().
    static void removeObjectCb(DtHlaObject* hlaObj, void* rel);

    // Cast the void* to a DtReflectedObjectList* and calls
    // processTimedOutObjectReflect().
    static void timedOutObjectReflectCb(const DtStateMsg& msg,
        DtHlaObject* obj,
        void* usr);

    // calls doTimeouts
    static void doTimeoutsPostDrainCb(void* usr);

```

protected:

```

    DtExerciseConn* myExConn;
    DtObjectIdHashList myObjectList;
    DtList myTimedOutObjectList;
    DtList myHandleList;
    DtIntrusiveList myObjectAddedCallbacks;
    DtIntrusiveList myObjectRemovedCallbacks;
    DtTime myTimeoutInterval;
    DtBoolean myTimeoutProcessingFlag;

```

protected:

```

    static DtBoolean theRequestClassUpdateFlag;
    static DtBoolean theRequestObjectUpdateFlag;
};
#endif    // DtHLA

```

hRefIEmitrSys.h

```
#if DtHLA

#include "emitterSysSR.h"
#include "hReflectedObj.h"

// Callback function that takes a DtReflectedObject and user data
typedef void (*DtEmitterSystemCallbackFunction)(
    DtReflectedEmitterSystem* obj, void* userData);

// class DtReflectedEmitterSystem:
//
// Instances of DtReflectedEmitterSystem represent EmitterSystem
// object simulated by remote applications. State data can be
// viewed
// through it's state repository.

class DT_DLL_VL DtReflectedEmitterSystem : public DtReflectedObject
{
public:

    // Function prototype for functions that create DtEmitterSystem-
    // Repositories
    typedef DtEmitterSystemRepository* (*DtStateRepCreator)();

public:

    // constructor
    // If beamsUpdateSystem == DtTrue, then a system's post-update
    // callbacks
    // are called whenever one of the beams in the system is updated.
    DtReflectedEmitterSystem(DtHlaObject* obj,
        DtExerciseConn* conn, DtBoolean beamsUpdateSystem);

    // Constructor used to construct a placeholder system when we
    // discover a beam whose system hasn't been discovered yet.
    DtReflectedEmitterSystem(const DtObjectId& id,
        const DtGlobalObjectDesignator& des,
        DtExerciseConn* conn, DtBoolean beamsUpdateSystem);

    // destructor
    virtual ~DtReflectedEmitterSystem();

    // Get the emitter system's state repository.
    virtual DtEmitterSystemRepository* emitterSysRep() const;

    // Synonym for emitterSysRep()
    virtual DtEmitterSystemRepository* esr() const;

};

#endif
```

```

// Get the next remote emitterSystem.
virtual DtReflectedEmitterSystem* next() const;

// Get the previous remote emitterSystem.
virtual DtReflectedEmitterSystem* prev() const;

// Get the next remote emitterSystem.
// Will cycle back to the first remote emitterSystem.
virtual DtReflectedEmitterSystem* wrapNext() const;

// Get the previous remote emitterSystem.
// Will cycle back to the last remote emitterSystem.
virtual DtReflectedEmitterSystem* wrapPrev() const;

// emitterSystem state callback registration/deregistration
virtual void addPostUpdateCallback(
    DtEmitterSystemCallbackFunction cb, void* userData);

virtual void removePostUpdateCallback(
    DtEmitterSystemCallbackFunction cb, void* userData);

// Tell the emitter system that one of its beams has been updated
virtual void beamUpdated();

public:

    // Give us a function to be used for creating SRs.
    static DtStateRepCreator setStateRepCreator(DtStateRepCreator
creator);

protected:

    // copy constructor
    DtReflectedEmitterSystem(const DtReflectedEmitterSystem& orig);

    // assignment operator
    DtReflectedEmitterSystem& operator=(
        const DtReflectedEmitterSystem& orig);

protected:

    static DtStateRepCreator theStateRepCreator;

protected:

    DtBoolean myBeamsUpdateSystem;
};
#endif //DtReflectedEmitterSystem_H_

```

hRefEntList.h

```
#if DtHLA

#include "hRefObjList.h"
#include "hReflectedEnt.h"

// class DtReflectedEntityList:
//
// Instances of DtReflectedEntityList are used to manage remote
// entities
// from the network.

class DT_DLL_VL DtReflectedEntityList : public DtReflectedObjectList
{
public:

    // Constructor - obtains the set of object classes to subscribe
    // to from the exConn's fomMapper.
    DtReflectedEntityList(DtExerciseConn* exConn,
        DtBoolean smoothCapable = DtFalse);

    // Constructor - only subscribe to the RTI::ObjectClassHandles
    // in handleList
    DtReflectedEntityList(DtExerciseConn* exConn,
        const DtList *handleList, DtBoolean smoothCapable = DtFalse);

    // destructor
    virtual ~DtReflectedEntityList();

    // Look up entity by identifier.
    virtual DtReflectedEntity* lookup(RTI::ObjectHandle id);

#if !DtRTI103

    // Look up entity by name.
    virtual DtReflectedEntity* lookup(const DtGlobalObjectDesignator&
objName);

#endif

    // Get the first/last entity in the list.
    virtual DtReflectedEntity* first() const;
    virtual DtReflectedEntity* last() const;

    // Entity added callback registration/deregistration
    virtual void addEntityAdditionCallback(
        DtEntityCallbackFunction cb, void* userData);
    virtual void removeEntityAdditionCallback(
```

```

        DtEntityCallbackFunction cb, void* userData);

// Entity removed callback registration/deregistration
virtual void addEntityRemovalCallback(
    DtEntityCallbackFunction cb, void* userData);
virtual void removeEntityRemovalCallback(
    DtEntityCallbackFunction cb, void* userData);

protected:

    // copy constructor
    DtReflectedEntityList(const DtReflectedEntityList& orig);

    // assignment operator
    DtReflectedEntityList& operator=(const DtReflectedEntityList&
orig);

    // Asks the FOM Mapper for the DtList of RTI::ObjectClassHandles
to use.
    virtual const DtList* dfltHandleList();

    // Override of virtual function. Calls newReflectedEntity.
    virtual DtReflectedObject* newReflectedObject(DtHlaObject* obj)
const;

    // Create a remote entity object for the HLA object.
    // Caller is responsible for deletion.
    virtual DtReflectedEntity* newReflectedEntity(DtHlaObject* obj)
const;

    // Virtual function override.
    // Calls removeAndDelete(DtReflectedEntity*....)
    virtual void removeAndDelete(DtReflectedObject* object, char*
msg);

    // Default calls down to base class version, but can be over-
ridden.
    virtual void removeAndDelete(DtReflectedEntity* entity, char* msg
= NULL);

    // Called from processDiscoverObject() after a remote transmitter
has
    // been added to the list. Calls entityAdded.
    virtual void objectAdded(DtReflectedObject* obj);

    // Called by objectAdded (which is called from processDiscoverOb-
ject())
    // after a remote entity has been added to the list. Default
    // implementation prints a message. Subclasses can override this
function

```

```
// to add functionality on addition.  
virtual void entityAdded(DtReflectedEntity* ent);  
  
protected:  
  
    DtBoolean mySmoothCapable;  
};  
#endif
```


hRefIRadRecvr.h

```

#ifdef DtHLA

#ifndef DtReflectedRadioReceiver_H_
#define DtReflectedRadioReceiver_H_

#include "radioRecvrSR.h"
#include "hReflectedObj.h"

// Callback function that takes a DtReflectedObject and user data
typedef void (*DtRadioReceiverCallbackFunction)(
    DtReflectedRadioReceiver* obj, void* userData);

// class DtReflectedRadioReceiver:
//
// Instances of DtReflectedRadioReceiver represent RadioReceiver
// object simulated by remote applications. State data can be
// viewed
// through it's state repository.

class DT_DLL_VL DtReflectedRadioReceiver : public DtReflectedObject
{
public:

    // Function prototype for functions that create DtRadioReceiver-
    // Repositories
    typedef DtRadioReceiverRepository* (*DtStateRepCreator)();

public:

    // constructor
    DtReflectedRadioReceiver(DtHlaObject* obj, DtExerciseConn* conn);

    // destructor
    virtual ~DtReflectedRadioReceiver();

    // Get the receiver's state repository.
    virtual DtRadioReceiverRepository* receiverRep() const;

    // Synonym for receiverRep()
    virtual DtRadioReceiverRepository* rsr() const;

    // Get the next remote receiver.
    virtual DtReflectedRadioReceiver* next() const;

    // Get the previous remote receiver.
    virtual DtReflectedRadioReceiver* prev() const;

```

```
// Get the next remote receiver.
// Will cycle back to the first remote receiver.
virtual DtReflectedRadioReceiver* wrapNext() const;

// Get the previous remote receiver.
// Will cycle back to the last remote receiver.
virtual DtReflectedRadioReceiver* wrapPrev() const;

// receiver state callback registration/deregistration
virtual void addPostUpdateCallback(
    DtRadioReceiverCallbackFunction cb, void* userData);

virtual void removePostUpdateCallback(
    DtRadioReceiverCallbackFunction cb, void* userData);

public:

    // Give us a function to be used for creating SRs.
    static DtStateRepCreator setStateRepCreator(DtStateRepCreator
creator);

protected:

    // copy constructor
    DtReflectedRadioReceiver(const DtReflectedRadioReceiver& orig);

    // assignment operator
    DtReflectedRadioReceiver& operator=(
        const DtReflectedRadioReceiver& orig);

protected:

    static DtStateRepCreator theStateRepCreator;
};
#endif //DtReflectedRadioReceiver_H_
```

hRefRadXmitr.h

```

#if DtHLA

#include "radioXmitSR.h"
#include "hReflectedObj.h"

// Callback function that takes a DtReflectedObject and user data
typedef void (*DtRadioTransmitterCallbackFunction)(
    DtReflectedRadioTransmitter* obj, void* userData);

// class DtReflectedRadioTransmitter:
//
// Instances of DtReflectedRadioTransmitter represent RadioTrans-
// mitter
// object simulated by remote applications. State data can be
// viewed
// through it's state repository.

class DT_DLL_VL DtReflectedRadioTransmitter : public DtReflectedOb-
ject
{
public:

    // Function prototype for functions that create DtRadioTransmit-
    // terRepositories
    typedef DtRadioTransmitterRepository* (*DtStateRepCreator)();

public:

    // constructor
    DtReflectedRadioTransmitter(DtHlaObject* obj, DtExerciseConn*
conn);

    // destructor
    virtual ~DtReflectedRadioTransmitter();

    // Get the transmitter's state repository.
    virtual DtRadioTransmitterRepository* transmitterRep() const;

    // Synonym for transmitterRep()
    virtual DtRadioTransmitterRepository* tsr() const;

    // Get the next remote transmitter.
    virtual DtReflectedRadioTransmitter* next() const;

    // Get the previous remote transmitter.
    virtual DtReflectedRadioTransmitter* prev() const;

```

```
// Get the next remote transmitter.
// Will cycle back to the first remote transmitter.
virtual DtReflectedRadioTransmitter* wrapNext() const;

// Get the previous remote transmitter.
// Will cycle back to the last remote transmitter.
virtual DtReflectedRadioTransmitter* wrapPrev() const;

// transmitter state callback registration/deregistration
virtual void addPostUpdateCallback(
    DtRadioTransmitterCallbackFunction cb, void* userData);

virtual void removePostUpdateCallback(
    DtRadioTransmitterCallbackFunction cb, void* userData);

public:

    // Give us a function to be used for creating SRs.
    static DtStateRepCreator setStateRepCreator(DtStateRepCreator
creator);

protected:

    // copy constructor
    DtReflectedRadioTransmitter(const DtReflectedRadioTransmitter&
orig);

    // assignment operator
    DtReflectedRadioTransmitter& operator=(
        const DtReflectedRadioTransmitter& orig);

protected:

    static DtStateRepCreator theStateRepCreator;
};
#endif //DtReflectedRadioTransmitter_H_
```

hRefRadRvList.h

```

#if DtHLA

#include "hRefObjList.h"
#include "hRefRadRecvr.h"

// class DtReflectedRadioReceiverList:
//
// Instances of DtReflectedRadioReceiverList manage the list of
// of remote RadioReceivers in the exercise.

class DT_DLL_VL DtReflectedRadioReceiverList : public DtReflectedOb-
jectList
{
public:

    // Constructor
    DtReflectedRadioReceiverList(DtExerciseConn* exConn);

    // Constructor - only subscribe to the RTI::ObjectClassHandles
    // in handleList
    DtReflectedRadioReceiverList(
        DtExerciseConn* exConn, const DtList *handleList);

    // destructor
    virtual ~DtReflectedRadioReceiverList();

    // Look up receiver by identifier.
    virtual DtReflectedRadioReceiver* lookup(RTI::ObjectHandle id);

#if !DtRTI103
    // Look up receiver by name.
    virtual DtReflectedRadioReceiver* lookup(
        const DtGlobalObjectDesignator& objName);
#endif

    // Get the first/last receiver in the list.
    virtual DtReflectedRadioReceiver* first() const;
    virtual DtReflectedRadioReceiver* last() const;

    // RadioReceiver added callback registration/deregistration
    virtual void addRadioReceiverAdditionCallback(
        DtRadioReceiverCallbackFunction cb, void* userData);
    virtual void removeRadioReceiverAdditionCallback(
        DtRadioReceiverCallbackFunction cb, void* userData);

    // RadioReceiver removed callback registration/deregistration
    virtual void addRadioReceiverRemovalCallback(

```

```
        DtRadioReceiverCallbackFunction cb, void* userData);
virtual void removeRadioReceiverRemovalCallback(
        DtRadioReceiverCallbackFunction cb, void* userData);

protected:

    // copy constructor
    DtReflectedRadioReceiverList(const DtReflectedRadioReceiverList&
orig);

    // assignment operator
    DtReflectedRadioReceiverList& operator=(
        const DtReflectedRadioReceiverList& orig);

    // Override of virtual function.
    // Here, we create a DtReflectedRadioReceiver.
    virtual DtReflectedObject* newReflectedObject(DtHlaObject* obj)
const;

    // Called from processDiscoverObject() after a remote receiver
has
    // been added to the list. Default implementation prints a
message.
    // Subclasses can override this function to add functionality on
// addition. obj will be an instance of a class derived from
// DtReflectedObj.
    virtual void objectAdded(DtReflectedObject* obj);

    // Returns a DtList of RTI::ObjectClassHandles that represent the
list
    // of object classes to which this list should subscribe.
    virtual const DtList* dfltHandleList();
};
#endif    //DtHLA
```

hRefRadXmList.h

```

#if DtHLA

#include "hRefObjList.h"
#include "hRefRadXmitr.h"

// class DtReflectedRadioTransmitterList:
//
// Instances of DtReflectedRadioTransmitterList manage the list of
// of remote RadioTransmitters in the exercise.

class DT_DLL_VL DtReflectedRadioTransmitterList : public DtReflectedObjectList
{
public:

    // Constructor.
    DtReflectedRadioTransmitterList(DtExerciseConn* exConn);

    // Constructor - only subscribe to the RTI::ObjectClassHandles
    // in handleList
    DtReflectedRadioTransmitterList(
        DtExerciseConn* exConn, const DtList *handleList);

    // destructor
    virtual ~DtReflectedRadioTransmitterList();

    // Look up transmitter by identifier.
    virtual DtReflectedRadioTransmitter* lookup(RTI::ObjectHandle
id);

#if !DtRTI103
    // Look up transmitter by name.
    virtual DtReflectedRadioTransmitter* lookup(
        const DtGlobalObjectDesignator& objName);
#endif

    // Get the first/last transmitter in the list.
    virtual DtReflectedRadioTransmitter* first() const;
    virtual DtReflectedRadioTransmitter* last() const;

    // RadioTransmitter added callback registration/deregistration
    virtual void addRadioTransmitterAdditionCallback(
        DtRadioTransmitterCallbackFunction cb, void* userData);
    virtual void removeRadioTransmitterAdditionCallback(
        DtRadioTransmitterCallbackFunction cb, void* userData);

    // RadioTransmitter removed callback registration/deregistration

```

```
virtual void addRadioTransmitterRemovalCallback(
    DtRadioTransmitterCallbackFunction cb, void* userData);
virtual void removeRadioTransmitterRemovalCallback(
    DtRadioTransmitterCallbackFunction cb, void* userData);

protected:

    // copy constructor
    DtReflectedRadioTransmitterList(
        const DtReflectedRadioTransmitterList& orig);

    // assignment operator
    DtReflectedRadioTransmitterList& operator=(
        const DtReflectedRadioTransmitterList& orig);

    // Override of virtual function. Here, we create a
    // DtReflectedRadioTransmitter.
    virtual DtReflectedObject* newReflectedObject(DtHlaObject* obj)
const;

    // Called from processDiscoverObject() after a remote transmitter
    has
    // been added to the list. Default implementation prints a
    message.
    // Subclasses can override this function to add functionality on
    // addition. obj will be an instance of a class derived from
    // DtReflectedObj.
    virtual void objectAdded(DtReflectedObject* obj);

    // Returns a DtList of RTI::ObjectClassHandles that represent the
    list
    // of object classes to which this list should subscribe.
    virtual const DtList* dfltHandleList();
};
#endif //DtHLA
```


hRemObjReqDec.h

```

#if DtHLA

#include "interDecoder.h"

// This can be used to declare parameter processor functions
// which are static members of a class.

#define DtDECLARE_REMOBJREQ_PARAM_DECODER(paramName) \
    DtDECLARE_PARAM_DECODER(DtRemoveObjectRequestInter, paramName)

#define DtDEFINE_SIMPLE_REMOBJREQ_PARAM_DECODER( \
    paramName, netType, mutator) \
    DtDEFINE_SIMPLE_PARAM_DECODER( \
        DtRemoveObjectRequestDecoder, DtRemoveObjectRequestInter, \
        paramName, netType, mutator)

#define DtDEFINE_SIMPLE_REMOBJREQ_PARAM_DECODER_WITH_CAST( \
    paramName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_PARAM_DECODER_WITH_CAST( \
        DtRemoveObjectRequestDecoder, DtRemoveObjectRequestInter, \
        paramName, netType, mutator, castExpr)

// class DtRemoveObjectRequestDecoder:
//
// Instances of DtRemoveObjectRequestDecoder are used to decode
// RemoveObjectRequest Interactions from the network.

class DT_DLL_VL DtRemoveObjectRequestDecoder : public DtInteraction-
Decoder
{
public:

    // constructor
    DtRemoveObjectRequestDecoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtRemoveObjectRequestDecoder();

    // copy constructor
    DtRemoveObjectRequestDecoder(const DtRemoveObjectRequestDecoder&
orig);

    // assignment operator
    DtRemoveObjectRequestDecoder& operator=(
        const DtRemoveObjectRequestDecoder& orig);

```

protected:

```
    // Individual parameter decoders

    DtDECLARE_REMOBJREQ_PARAM_DECODER(ObjectIDs);
    DtDECLARE_REMOBJREQ_PARAM_DECODER(RequestID);
};
#endif
```

hRemObjReqEnc.h

```

#if DtHLA

#include "interEncoder.h"

#define DtDECLARE_REMOBJREQ_PARAM_ENCODER(paramName) \
    DtDECLARE_PARAM_ENCODER(DtRemoveObjectRequestInter, paramName)

#define DtDEFINE_SIMPLE_REMOBJREQ_PARAM_ENCODER(paramName, netType, \
    inspector) \
    DtDEFINE_SIMPLE_PARAM_ENCODER(DtRemoveObjectRequestEncoder, \
    DtRemoveObjectRequestInter, paramName, netType, inspector)

// class DtRemoveObjectRequestEncoder:
//
// DtRemoveObjectRequestEncoder is a DtInteractionEncoder that
// registers
// parameter encoders for all of the RemoveObjectRequest interaction
// parameters.

class DT_DLL_VL DtRemoveObjectRequestEncoder : public DtInteraction-
Encoder
{
public:

    DtRemoveObjectRequestEncoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtRemoveObjectRequestEncoder();

    // copy constructor
    DtRemoveObjectRequestEncoder(const DtRemoveObjectRequestEncoder&
orig);

    // assignment operator
    DtRemoveObjectRequestEncoder& operator=(
        const DtRemoveObjectRequestEncoder& orig);

protected:

    // Individual parameter encoders

    DtDECLARE_REMOBJREQ_PARAM_ENCODER(ObjectIDs);
    DtDECLARE_REMOBJREQ_PARAM_ENCODER(RequestID);
};
#endif

```

hRemObjReqInt.h

```
#if DtHLA

#include "hIntWEncDec.h"
#include "hostStructs.h"
#include "mathTypes.h"
#include "objDesList.h"

#include "chooseDefines.h"

// class DtRemoveObjectRequestInter:
//
// Instances of DtRemoveObjectRequestInter are used to represent
// RemoveObjectRequest Interactions in the RPR FOM.

// RemoveObjectRequest interaction callback function signature.
typedef void (*DtRemoveObjectRequestInterCb)
    (DtRemoveObjectRequestInter* inter, void* usr);

class DT_DLL_VL DtRemoveObjectRequestInter : public DtInteraction-
WithEncDec
{
public:

    // default constructor
    DtRemoveObjectRequestInter();

    // destructor
    virtual ~DtRemoveObjectRequestInter();

    // copy constructor
    DtRemoveObjectRequestInter(const DtRemoveObjectRequestInter&
orig);

    // assignment operator
    DtRemoveObjectRequestInter& operator=(
        const DtRemoveObjectRequestInter& orig);

    // Print the interaction's data.
    virtual void printData() const;

    // Return the Object ID list
    virtual const DtGlobalObjectDesignatorList& objectIDs() const;
    virtual DtGlobalObjectDesignatorList* objectIDs();

    // Set/Get number of IDs.
    virtual void setNumIds(DtU8 numIds);
    virtual DtU8 numIds() const;
```

```
// Set/Get the nth ObjectId.
virtual void setNthObjectId(
    DtInt32 idIndex, const DtGlobalObjectDesignator& id);
virtual const DtGlobalObjectDesignator& nthObjectId(DtInt32
idIndex) const;

// Set/Get request ID
virtual void setRequestId(DtU32 id);
virtual DtU32 requestId() const;

// Name of the interaction (Not nec. the ClassName from the FOM).
virtual const char* name() const;

public:

    // Create a DtRemoveObjectRequestInter.
    // Caller is responsible for deletion.
    static DtInteraction* create();

    // Add (register)/Remove (deregister) function to be called when
    // interaction occurs.
    static void addCallback(DtExerciseConn* conn,
        DtRemoveObjectRequestInterCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtRemoveObjectRequestInterCb cb, void* usr);

protected:

    DtGlobalObjectDesignatorList myObjectList;
    DtU32 myRequestId;

protected:

    static char* theClassName;
};
#endif
```

hRemObjResDec.h

```
#if DtHLA

#include "interDecoder.h"

// This can be used to declare parameter processor functions
// which are static members of a class.

#define DtDECLARE_REMOBJRES_PARAM_DECODER(paramName) \
    DtDECLARE_PARAM_DECODER(DtRemoveObjectResultInter, paramName)

#define DtDEFINE_SIMPLE_REMOBJRES_PARAM_DECODER( \
    paramName, netType, mutator) \
    DtDEFINE_SIMPLE_PARAM_DECODER( \
        DtRemoveObjectResultDecoder, DtRemoveObjectResultInter, \
        paramName, netType, mutator)

#define DtDEFINE_SIMPLE_REMOBJRES_PARAM_DECODER_WITH_CAST( \
    paramName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_PARAM_DECODER_WITH_CAST( \
        DtRemoveObjectResultDecoder, DtRemoveObjectResultInter, \
        paramName, netType, mutator, castExpr)

// class DtRemoveObjectResultDecoder:
//
// Instances of DtRemoveObjectResultDecoder are used to decode
// RemoveObjectResult Interactions from the network.

class DT_DLL_VL DtRemoveObjectResultDecoder : public DtInteraction-
Decoder
{
public:

    // constructor
    DtRemoveObjectResultDecoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtRemoveObjectResultDecoder();

    // copy constructor
    DtRemoveObjectResultDecoder(const DtRemoveObjectResultDecoder&
orig);

    // assignment operator
    DtRemoveObjectResultDecoder& operator=(
        const DtRemoveObjectResultDecoder& orig);
```

protected:

```
    // Individual parameter decoders

    DtDECLARE_REMOBJRES_PARAM_DECODER(RemoveObjectResult);
    DtDECLARE_REMOBJRES_PARAM_DECODER(RequestID);

    // RPR FOM 0.7 compatibility
    DtDECLARE_REMOBJRES_PARAM_DECODER(RemoveObjectResultPre08);
};
#endif
```

hRemObjResEnc.h

```
#if DtHLA

#include "interEncoder.h"

#define DtDECLARE_REMOBJRES_PARAM_ENCODER(paramName) \
    DtDECLARE_PARAM_ENCODER(DtRemoveObjectResultInter, paramName)

#define DtDEFINE_SIMPLE_REMOBJRES_PARAM_ENCODER(paramName, netType, \
    inspector) \
    DtDEFINE_SIMPLE_PARAM_ENCODER(DtRemoveObjectResultEncoder, \
        DtRemoveObjectResultInter, paramName, netType, inspector)

// class DtRemoveObjectResultEncoder:
//
// DtRemoveObjectResultEncoder is a DtInteractionEncoder that regis-
// ters
// parameter encoders for all of the RemoveObjectResult interaction
// parameters.

class DT_DLL_VL DtRemoveObjectResultEncoder : public DtInteraction-
Encoder
{
public:

    DtRemoveObjectResultEncoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtRemoveObjectResultEncoder();

    // copy constructor
    DtRemoveObjectResultEncoder(const DtRemoveObjectResultEncoder&
orig);

    // assignment operator
    DtRemoveObjectResultEncoder& operator=(
        const DtRemoveObjectResultEncoder& orig);

protected:

    // Individual parameter encoders

    DtDECLARE_REMOBJRES_PARAM_ENCODER(RemoveObjectResult);
    DtDECLARE_REMOBJRES_PARAM_ENCODER(RequestID);

    // RPR FOM 0.7 compatibility
    DtDECLARE_REMOBJRES_PARAM_ENCODER(RemoveObjectResultPre08);
```



```
};  
#endif
```

hRemObjResInt.h

```
#if DtHLA

#include "hIntWEncDec.h"
#include "hostStructs.h"
#include "mathTypes.h"

#include "chooseDefines.h"

// class DtRemoveObjectResultInter:
//
// Instances of DtRemoveObjectResultInter are used to represent
// RemoveObjectResult Interactions in the RPR FOM.

// RemoveObjectResult interaction callback function signature.
typedef void (*DtRemoveObjectResultInterCb)
    (DtRemoveObjectResultInter* inter, void* usr);

class DT_DLL_VL DtRemoveObjectResultInter : public DtInteraction-
WithEncDec
{
public:

    // default constructor
    DtRemoveObjectResultInter();

    // destructor
    virtual ~DtRemoveObjectResultInter();

    // copy constructor
    DtRemoveObjectResultInter(const DtRemoveObjectResultInter& orig);

    // assignment operator
    DtRemoveObjectResultInter& operator=(
        const DtRemoveObjectResultInter& orig);

    // Print the interaction's data.
    virtual void printData() const;

    // Set/Get create object result
    virtual void setRemoveObjectResult(DtRemoveObjectResult result);
    virtual DtRemoveObjectResult removeObjectResult() const;

    // Set/Get request ID
    virtual void setRequestId(DtU32 id);
    virtual DtU32 requestId() const;

    // Name of the interaction (Not nec. the ClassName from the FOM).
```

```
    virtual const char* name() const;

public:

    // Create a DtRemoveObjectResultInter.
    // Caller is responsible for deletion.
    static DtInteraction* create();

    // Add (register)/Remove (deregister) function to be called when
    // interaction occurs.
    static void addCallback(DtExerciseConn* conn,
        DtRemoveObjectResultInterCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtRemoveObjectResultInterCb cb, void* usr);

protected:

    DtRemoveObjectResult myRemoveObjectResult;
    DtU32 myRequestId;

protected:

    static char* theClassName;
};
#endif
```

hRemovEntDec.h

```
#if DtHLA

#include "interDecoder.h"

// This can be used to declare parameter processor functions
// which are static members of a class.

#define DtDECLARE_REMOVEVENT_PARAM_DECODER(paramName) \
    DtDECLARE_PARAM_DECODER(DtRemoveEntityInteraction, paramName)

#define DtDEFINE_SIMPLE_REMOVEVENT_PARAM_DECODER( \
    paramName, netType, mutator) \
    DtDEFINE_SIMPLE_PARAM_DECODER( \
        DtRemoveEntityDecoder, DtRemoveEntityInteraction, \
        paramName, netType, mutator)

#define DtDEFINE_SIMPLE_REMOVEVENT_PARAM_DECODER_WITH_CAST( \
    paramName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_PARAM_DECODER_WITH_CAST( \
        DtRemoveEntityDecoder, DtRemoveEntityInteraction, \
        paramName, netType, mutator, castExpr)

// class DtRemoveEntityDecoder:
//
// Instances of DtRemoveEntityDecoder are used to decodeRemoveEn-
// tity
// Interactions from the network.

class DT_DLL_VL DtRemoveEntityDecoder : public DtInteractionDecoder
{
public:

    // constructor
    DtRemoveEntityDecoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtRemoveEntityDecoder();

    // copy constructor
    DtRemoveEntityDecoder(const DtRemoveEntityDecoder& orig);

    // assignment operator
    DtRemoveEntityDecoder& operator=(const DtRemoveEntityDecoder&
orig);

protected:
```

```
// Individual parameter decoders

DtDECLARE_REMOVENT_PARAM_DECODER(OriginatingEntity);
DtDECLARE_REMOVENT_PARAM_DECODER(ReceivingEntity);
DtDECLARE_REMOVENT_PARAM_DECODER(RequestID);
};
#endif
```

hRemovEntEnc.h

```
#if DtHLA

#include "interEncoder.h"

#define DtDECLARE_REMOVENT_PARAM_ENCODER(paramName) \
    DtDECLARE_PARAM_ENCODER(DtRemoveEntityInteraction, paramName)

#define DtDEFINE_SIMPLE_REMOVENT_PARAM_ENCODER(paramName, netType, \
    inspector) \
    DtDEFINE_SIMPLE_PARAM_ENCODER(DtRemoveEntityEncoder, \
        DtRemoveEntityInteraction, paramName, netType, inspector)

// class DtRemoveEntityEncoder:
//
// DtRemoveEntityEncoder is a DtInteractionEncoder that registers
// parameter encoders for all of the RemoveEntity interaction param-
// eters.

class DT_DLL_VL DtRemoveEntityEncoder : public DtInteractionEncoder
{
public:

    DtRemoveEntityEncoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtRemoveEntityEncoder();

    // copy constructor
    DtRemoveEntityEncoder(const DtRemoveEntityEncoder& orig);

    // assignment operator
    DtRemoveEntityEncoder& operator=(const DtRemoveEntityEncoder&
orig);

protected:

    // Individual parameter encoders

    DtDECLARE_REMOVENT_PARAM_ENCODER(OriginatingEntity);
    DtDECLARE_REMOVENT_PARAM_ENCODER(ReceivingEntity);
    DtDECLARE_REMOVENT_PARAM_ENCODER(RequestID);
};
#endif
```

hRemovEntInt.h

```

#if DtHLA

#include "hIntWEncDec.h"
#include "hostStructs.h"
#include "mathTypes.h"

#include "chooseDefines.h"

// class DtRemoveEntityInteraction:
//
// Instances of DtRemoveEntityInteraction are used to represent
// RemoveEntity Interactions in the RPR FOM.

// RemoveEntity interaction callback function signature.
typedef void (*DtRemoveEntityInteractionCb)(DtRemoveEntityInterac-
tion* inter,
void* usr);

class DT_DLL_VL DtRemoveEntityInteraction : public DtInteraction-
WithEncDec
{
public:

    // default constructor
    DtRemoveEntityInteraction();

    // destructor
    virtual ~DtRemoveEntityInteraction();

    // copy constructor
    DtRemoveEntityInteraction(const DtRemoveEntityInteraction& orig);

    // assignment operator
    DtRemoveEntityInteraction& operator=(const DtRemoveEntityInterac-
tion& orig);

    // Print the interaction's data.
    virtual void printData() const;

    // Set/Get sender ID
    virtual void setSenderId(const DtEntityIdentifier& id);
    virtual const DtEntityIdentifier& senderId() const;

    // Set/Get receiver ID
    virtual void setReceiverId(const DtEntityIdentifier& id);
    virtual const DtEntityIdentifier& receiverId() const;

```

```
// Set/Get request ID
virtual void setRequestId(int id);
virtual int requestId() const;

// Name of the interaction (Not nec. the ClassName from the FOM).
virtual const char* name() const;

public:

    // Create a DtRemoveEntityInteraction.
    // Caller is responsible for deletion.
    static DtInteraction* create();

    // Add (register)/Remove (deregister) function to be called when
    // interaction occurs.
    static void addCallback(DtExerciseConn* conn,
        DtRemoveEntityInteractionCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtRemoveEntityInteractionCb cb, void* usr);

protected:

    DtEntityIdentifier myOrigEnt;
    DtEntityIdentifier myReceiveEnt;
    int myRequestId;

protected:

    static char* theClassName;
};
#endif
```


hRepCompDec.h

```

#if DtHLA

#include "interDecoder.h"

// This can be used to declare parameter processor functions
// which are static members of a class.

#define DtDECLARE_REPCOMP_PARAM_DECODER(paramName) \
    DtDECLARE_PARAM_DECODER(DtRepairCompleteInteraction, paramName)

#define DtDEFINE_SIMPLE_REPCOMP_PARAM_DECODER( \
    paramName, netType, mutator) \
    DtDEFINE_SIMPLE_PARAM_DECODER( \
        DtRepairCompleteDecoder, DtRepairCompleteInteraction, \
        paramName, netType, mutator)

#define DtDEFINE_SIMPLE_REPCOMP_PARAM_DECODER_WITH_CAST( \
    paramName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_PARAM_DECODER_WITH_CAST( \
        DtRepairCompleteDecoder, DtRepairCompleteInteraction, \
        paramName, netType, mutator, castExpr)

// class DtRepairCompleteDecoder:
//
// Instances of DtRepairCompleteDecoder are used to decode Repair-
// Complete
// Interactions from the network.

class DT_DLL_VL DtRepairCompleteDecoder : public DtInteractionDe-
coder
{
public:

    // constructor
    DtRepairCompleteDecoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtRepairCompleteDecoder();

    // copy constructor
    DtRepairCompleteDecoder(const DtRepairCompleteDecoder& orig);

    // assignment operator
    DtRepairCompleteDecoder& operator=(const DtRepairCompleteDecoder&
orig);

```

protected:

```
    // Individual parameter decoders

    DtDECLARE_REPCOMP_PARAM_DECODER(ReceivingObject);
    DtDECLARE_REPCOMP_PARAM_DECODER(RepairingObject);
    DtDECLARE_REPCOMP_PARAM_DECODER(Repair);
};
#endif
```

hRepCompEnc.h

```

#ifdef DtHLA

#include "interEncoder.h"

#define DtDECLARE_REPCOMP_PARAM_ENCODER(paramName) \
    DtDECLARE_PARAM_ENCODER(DtRepairCompleteInteraction, paramName)

#define DtDEFINE_SIMPLE_REPCOMP_PARAM_ENCODER(paramName, netType, \
    inspector) \
    DtDEFINE_SIMPLE_PARAM_ENCODER(DtRepairCompleteEncoder, \
        DtRepairCompleteInteraction, paramName, netType, inspector)

// class DtRepairCompleteEncoder:
//
// DtRepairCompleteEncoder is a DtInteractionEncoder that registers
// parameter encoders for all of the RepairComplete interaction
// parameters.

class DT_DLL_VL DtRepairCompleteEncoder : public DtInteractionEn-
coder
{
public:

    DtRepairCompleteEncoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtRepairCompleteEncoder();

    // copy constructor
    DtRepairCompleteEncoder(const DtRepairCompleteEncoder& orig);

    // assignment operator
    DtRepairCompleteEncoder& operator=(const DtRepairCompleteEncoder&
orig);

protected:

    // Individual parameter encoders

    DtDECLARE_REPCOMP_PARAM_ENCODER(ReceivingObject);
    DtDECLARE_REPCOMP_PARAM_ENCODER(RepairingObject);
    DtDECLARE_REPCOMP_PARAM_ENCODER(Repair);
};
#endif

```

hRepComplInter.h

```
#if DtHLA

#include "hIntWEncDec.h"
#include "hostStructs.h"
#include "mathTypes.h"

#include "chooseDefines.h"

// class DtRepairCompleteInteraction:
//
// Instances of DtRepairCompleteInteraction are used to represent
// RepairComplete Interactions in the RPR FOM.

// RepairComplete interaction callback function signature.
typedef void (*DtRepairCompleteInteractionCb)
    (DtRepairCompleteInteraction* inter, void* usr);

class DT_DLL_VL DtRepairCompleteInteraction : public DtInteraction-
WithEncDec
{
public:

    // default constructor
    DtRepairCompleteInteraction();

    // destructor
    virtual ~DtRepairCompleteInteraction();

    // copy constructor
    DtRepairCompleteInteraction(const DtRepairCompleteInteraction&
orig);

    // assignment operator
    DtRepairCompleteInteraction& operator=(
        const DtRepairCompleteInteraction& orig);

    // Print the interaction's data.
    virtual void printData() const;

    // Set/Get the receiving object.
    virtual void setReceivingId(const DtGlobalObjectDesignator &id);
    virtual const DtGlobalObjectDesignator &receivingId() const;

    // Set/Get the repairing object.
    virtual void setRepairingId(const DtGlobalObjectDesignator &id);
    virtual const DtGlobalObjectDesignator &repairingId() const;
```

```
// Set/Get the repair type.
virtual void setRepairType(DtRepairType type);
virtual DtRepairType repairType() const;

// Name of the interaction (Not nec. the ClassName from the FOM).
virtual const char* name() const;

public:

    // Create a DtRepairCompleteInteraction.
    // Caller is responsible for deletion.
    static DtInteraction* create();

    // Add (register)/Remove (deregister) function to be called when
    // interaction occurs.
    static void addCallback(DtExerciseConn* conn,
        DtRepairCompleteInteractionCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtRepairCompleteInteractionCb cb, void* usr);

protected:

    DtGlobalObjectDesignator myReceivingObject;
    DtGlobalObjectDesignator myRepairingObject;
    DtRepairType myRepairType;
};
#endif
```

hRepRespDec.h

```
#if DtHLA

#include "interDecoder.h"

// This can be used to declare parameter processor functions
// which are static members of a class.

#define DtDECLARE_REPRES_PARAM_DECODER(paramName) \
    DtDECLARE_PARAM_DECODER(DtRepairResponseInteraction, paramName)

#define DtDEFINE_SIMPLE_REPRES_PARAM_DECODER( \
    paramName, netType, mutator) \
    DtDEFINE_SIMPLE_PARAM_DECODER( \
        DtRepairResponseDecoder, DtRepairResponseInteraction, \
        paramName, netType, mutator)

#define DtDEFINE_SIMPLE_REPRES_PARAM_DECODER_WITH_CAST( \
    paramName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_PARAM_DECODER_WITH_CAST( \
        DtRepairResponseDecoder, DtRepairResponseInteraction, \
        paramName, netType, mutator, castExpr)

// class DtRepairResponseDecoder:
//
// Instances of DtRepairResponseDecoder are used to decode RepairRe-
// sponse
// Interactions from the network.

class DT_DLL_VL DtRepairResponseDecoder : public DtInteractionDe-
coder
{
public:

    // constructor
    DtRepairResponseDecoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtRepairResponseDecoder();

    // copy constructor
    DtRepairResponseDecoder(const DtRepairResponseDecoder& orig);

    // assignment operator
    DtRepairResponseDecoder& operator=(const DtRepairResponseDecoder&
orig);
```

protected:

```
    // Individual parameter decoders

    DtDECLARE_REPRES_PARAM_DECODER(ReceivingObject);
    DtDECLARE_REPRES_PARAM_DECODER(RepairingObject);
    DtDECLARE_REPRES_PARAM_DECODER(RepairResult);
};
#endif
```

hRepRespEnc.h

```
#if DtHLA

#include "interEncoder.h"

#define DtDECLARE_REPRES_PARAM_ENCODER(paramName) \
    DtDECLARE_PARAM_ENCODER(DtRepairResponseInteraction, paramName)

#define DtDEFINE_SIMPLE_REPRES_PARAM_ENCODER(paramName, netType, \
    inspector) \
    DtDEFINE_SIMPLE_PARAM_ENCODER(DtRepairResponseEncoder, \
        DtRepairResponseInteraction, paramName, netType, inspector)

// class DtRepairResponseEncoder:
//
// DtRepairResponseEncoder is a DtInteractionEncoder that registers
// parameter encoders for all of the RepairResponse interaction
// parameters.

class DT_DLL_VL DtRepairResponseEncoder : public DtInteractionEn-
coder
{
public:

    DtRepairResponseEncoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtRepairResponseEncoder();

    // copy constructor
    DtRepairResponseEncoder(const DtRepairResponseEncoder& orig);

    // assignment operator
    DtRepairResponseEncoder& operator=(const DtRepairResponseEncoder&
orig);

protected:

    // Individual parameter encoders

    DtDECLARE_REPRES_PARAM_ENCODER(ReceivingObject);
    DtDECLARE_REPRES_PARAM_ENCODER(RepairingObject);
    DtDECLARE_REPRES_PARAM_ENCODER(RepairResult);
};
#endif
```


hRepResplntr.h

```

#if DtHLA

#include "hIntWEncDec.h"
#include "hostStructs.h"
#include "mathTypes.h"

#include "chooseDefines.h"

// class DtRepairResponseInteraction:
//
// Instances of DtRepairResponseInteraction are used to represent
// RepairResponse Interactions in the RPR FOM.

// RepairResponse interaction callback function signature.
typedef void (*DtRepairResponseInteractionCb)
    (DtRepairResponseInteraction* inter, void* usr);

class DT_DLL_VL DtRepairResponseInteraction : public DtInteraction-
WithEncDec
{
public:

    // default constructor
    DtRepairResponseInteraction();

    // destructor
    virtual ~DtRepairResponseInteraction();

    // copy constructor
    DtRepairResponseInteraction(const DtRepairResponseInteraction&
orig);

    // assignment operator
    DtRepairResponseInteraction& operator=(
        const DtRepairResponseInteraction& orig);

    // Print the interaction's data.
    virtual void printData() const;

    // Set/Get the receiving object.
    virtual void setReceivingId(const DtGlobalObjectDesignator &id);
    virtual const DtGlobalObjectDesignator &receivingId() const;

    // Set/Get the repairing object.
    virtual void setRepairingId(const DtGlobalObjectDesignator &id);
    virtual const DtGlobalObjectDesignator &repairingId() const;

```

```
// Set/Get the repair type.
virtual void setRepairResult(DtRepairResult result);
virtual DtRepairResult repairResult() const;

// Name of the interaction (Not nec. the ClassName from the FOM).
virtual const char* name() const;

public:

    // Create a DtRepairResponseInteraction.
    // Caller is responsible for deletion.
    static DtInteraction* create();

    // Add (register)/Remove (deregister) function to be called when
    // interaction occurs.
    static void addCallback(DtExerciseConn* conn,
        DtRepairResponseInteractionCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtRepairResponseInteractionCb cb, void* usr);

protected:

    DtGlobalObjectDesignator myReceivingObject;
    DtGlobalObjectDesignator myRepairingObject;
    DtRepairResult myRepairResult;
};
#endif
```

hResCancDec.h

```

#if DtHLA

#include "interDecoder.h"

// This can be used to declare parameter processor functions
// which are static members of a class.

#define DtDECLARE_RESCANC_PARAM_DECODER(paramName) \
    DtDECLARE_PARAM_DECODER(DtResupplyCancelInteraction, paramName)

#define DtDEFINE_SIMPLE_RESCANC_PARAM_DECODER( \
    paramName, netType, mutator) \
    DtDEFINE_SIMPLE_PARAM_DECODER( \
        DtResupplyCancelDecoder, DtResupplyCancelInteraction, \
        paramName, netType, mutator)

#define DtDEFINE_SIMPLE_RESCANC_PARAM_DECODER_WITH_CAST( \
    paramName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_PARAM_DECODER_WITH_CAST( \
        DtResupplyCancelDecoder, DtResupplyCancelInteraction, \
        paramName, netType, mutator, castExpr)

// class DtResupplyCancelDecoder:
//
// Instances of DtResupplyCancelDecoder are used to decode Resupply-
// Cancel
// Interactions from the network.

class DT_DLL_VL DtResupplyCancelDecoder : public DtInteractionDe-
coder
{
public:

    // constructor
    DtResupplyCancelDecoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtResupplyCancelDecoder();

    // copy constructor
    DtResupplyCancelDecoder(const DtResupplyCancelDecoder& orig);

    // assignment operator
    DtResupplyCancelDecoder& operator=(const DtResupplyCancelDecoder&
orig);

```

protected:

```
    // Individual parameter decoders

    DtDECLARE_RESCANC_PARAM_DECODER(ReceivingObject);
    DtDECLARE_RESCANC_PARAM_DECODER(SupplyingObject);
};
#endif
```

hResCancEnc.h

```

#if DtHLA

#include "interEncoder.h"

#define DtDECLARE_RESCANC_PARAM_ENCODER(paramName) \
    DtDECLARE_PARAM_ENCODER(DtResupplyCancelInteraction, paramName)

#define DtDEFINE_SIMPLE_RESCANC_PARAM_ENCODER(paramName, netType, \
    inspector) \
    DtDEFINE_SIMPLE_PARAM_ENCODER(DtResupplyCancelEncoder, \
    DtResupplyCancelInteraction, paramName, netType, inspector)

// class DtResupplyCancelEncoder:
//
// DtResupplyCancelEncoder is a DtInteractionEncoder that registers
// parameter encoders for all of the ResupplyCancel interaction
// parameters.

class DT_DLL_VL DtResupplyCancelEncoder : public DtInteractionEn-
coder
{
public:

    DtResupplyCancelEncoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtResupplyCancelEncoder();

    // copy constructor
    DtResupplyCancelEncoder(const DtResupplyCancelEncoder& orig);

    // assignment operator
    DtResupplyCancelEncoder& operator=(const DtResupplyCancelEncoder&
orig);

protected:

    // Individual parameter encoders

    DtDECLARE_RESCANC_PARAM_ENCODER(ReceivingObject);
    DtDECLARE_RESCANC_PARAM_ENCODER(SupplyingObject);
};
#endif

```

hResCancelInter.h

```
#if DtHLA

#include "hIntWEncDec.h"
#include "hostStructs.h"
#include "mathTypes.h"

#include "chooseDefines.h"

// class DtResupplyCancelInteraction:
//
// Instances of DtResupplyCancelInteraction are used to represent
// ResupplyCancel Interactions in the RPR FOM.

// ResupplyCancel interaction callback function signature.
typedef void (*DtResupplyCancelInteractionCb)
    (DtResupplyCancelInteraction* inter, void* usr);

class DT_DLL_VL DtResupplyCancelInteraction : public DtInteraction-
WithEncDec
{
public:

    // default constructor
    DtResupplyCancelInteraction();

    // destructor
    virtual ~DtResupplyCancelInteraction();

    // copy constructor
    DtResupplyCancelInteraction(const DtResupplyCancelInteraction&
orig);

    // assignment operator
    DtResupplyCancelInteraction& operator=(
        const DtResupplyCancelInteraction& orig);

    // Print the interaction's data.
    virtual void printData() const;

    // Set/Get the receiving object.
    virtual void setReceivingId(const DtGlobalObjectDesignator
&requester);
    virtual const DtGlobalObjectDesignator& receivingId() const;

    // Set/Get the supplying object.
    virtual void setSupplyingId(const DtGlobalObjectDesignator
&servicer);
}
```

```
virtual const DtGlobalObjectDesignator& supplyingId() const;

// Name of the interaction (Not nec. the ClassName from the FOM).
virtual const char* name() const;

public:

    // Create a DtResupplyCancelInteraction.
    // Caller is responsible for deletion.
    static DtInteraction* create();

    // Add (register)/Remove (deregister) function to be called when
    // interaction occurs.
    static void addCallback(DtExerciseConn* conn,
        DtResupplyCancelInteractionCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtResupplyCancelInteractionCb cb, void* usr);

protected:

    DtGlobalObjectDesignator myReceivingObject;
    DtGlobalObjectDesignator mySupplyingObject;
};
#endif
```

hResOffDec.h

```
#if DtHLA

#include "interDecoder.h"

// This can be used to declare parameter processor functions
// which are static members of a class.

#define DtDECLARE_RESOFF_PARAM_DECODER(paramName) \
    DtDECLARE_PARAM_DECODER(DtResupplyOfferInteraction, paramName)

#define DtDEFINE_SIMPLE_RESOFF_PARAM_DECODER( \
    paramName, netType, mutator) \
    DtDEFINE_SIMPLE_PARAM_DECODER( \
        DtResupplyOfferDecoder, DtResupplyOfferInteraction, \
        paramName, netType, mutator)

#define DtDEFINE_SIMPLE_RESOFF_PARAM_DECODER_WITH_CAST( \
    paramName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_PARAM_DECODER_WITH_CAST( \
        DtResupplyOfferDecoder, DtResupplyOfferInteraction, \
        paramName, netType, mutator, castExpr)

// class DtResupplyOfferDecoder:
//
// Instances of DtResupplyOfferDecoder are used to decode Resupply-
// Offer
// Interactions from the network.

class DT_DLL_VL DtResupplyOfferDecoder : public DtInteractionDecoder
{
public:

    // constructor
    DtResupplyOfferDecoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtResupplyOfferDecoder();

    // copy constructor
    DtResupplyOfferDecoder(const DtResupplyOfferDecoder& orig);

    // assignment operator
    DtResupplyOfferDecoder& operator=(const DtResupplyOfferDecoder&
orig);

protected:
```



```
// Individual parameter decoders

DtDECLARE_RESOFF_PARAM_DECODER(ReceivingObject);
DtDECLARE_RESOFF_PARAM_DECODER(SupplyingObject);
DtDECLARE_RESOFF_PARAM_DECODER(Supplies);
};
#endif
```

hResOffEnc.h

```
#if DtHLA

#include "interEncoder.h"

#define DtDECLARE_RESOFF_PARAM_ENCODER(paramName) \
    DtDECLARE_PARAM_ENCODER(DtResupplyOfferInteraction, paramName)

#define DtDEFINE_SIMPLE_RESOFF_PARAM_ENCODER(paramName, netType, \
    inspector) \
    DtDEFINE_SIMPLE_PARAM_ENCODER(DtResupplyOfferEncoder, \
        DtResupplyOfferInteraction, paramName, netType, inspector)

// class DtResupplyOfferEncoder:
//
// DtResupplyOfferEncoder is a DtInteractionEncoder that registers
// parameter encoders for all of the ResupplyOffer interaction
// parameters.

class DT_DLL_VL DtResupplyOfferEncoder : public DtInteractionEncoder
{
public:

    DtResupplyOfferEncoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtResupplyOfferEncoder();

    // copy constructor
    DtResupplyOfferEncoder(const DtResupplyOfferEncoder& orig);

    // assignment operator
    DtResupplyOfferEncoder& operator=(const DtResupplyOfferEncoder&
orig);

protected:

    // Individual parameter encoders

    DtDECLARE_RESOFF_PARAM_ENCODER(ReceivingObject);
    DtDECLARE_RESOFF_PARAM_ENCODER(SupplyingObject);
    DtDECLARE_RESOFF_PARAM_ENCODER(Supplies);
};
#endif
```

hResOffInter.h

```

#if DtHLA

#include "hIntWEncDec.h"
#include "hostStructs.h"
#include "mathTypes.h"
#include "packets/logisticsP.h"

#include "chooseDefines.h"

#define DtSUPPLY_VALID 1
#define DtSUPPLY_BAD_INDEX 0

// class DtResupplyOfferInteraction:
//
// Instances of DtResupplyOfferInteraction are used to represent
// ResupplyOffer Interactions in the RPR FOM.

// ResupplyOffer interaction callback function signature.
typedef void (*DtResupplyOfferInteractionCb)
    (DtResupplyOfferInteraction* inter, void* usr);

class DT_DLL_VL DtResupplyOfferInteraction : public DtInteraction-
WithEncDec
{
public:

    // default constructor
    DtResupplyOfferInteraction();

    // destructor
    virtual ~DtResupplyOfferInteraction();

    // copy constructor
    DtResupplyOfferInteraction(const DtResupplyOfferInteraction&
orig);

    // assignment operator
    DtResupplyOfferInteraction& operator=(
        const DtResupplyOfferInteraction& orig);

    // Print the interaction's data.
    virtual void printData() const;

    // Set/Get the receiving object.
    virtual void setReceivingId(const DtGlobalObjectDesignator
&receiver);
    virtual const DtGlobalObjectDesignator &receivingId() const;

```

```
// Set/Get the repairing object.
virtual void setSupplyingId(const DtGlobalObjectDesignator
&supplier);
virtual const DtGlobalObjectDesignator &supplyingId() const;

// Set/Get the supplies memory block all at once
virtual void setSupplies(const void* data, int numBytes);
virtual const void* supplies() const;

// Supporting functions to get/set the Supply Array
virtual void setNumSupplyTypes(int numTypes);
virtual int numSupplyTypes() const;

virtual void setNthSupply(int supplyIndex, const DtEntityType
&supply,
    float quantity, int *retCode = NULL);
virtual const DtEntityType &nthSupply(int supplyIndex,
    float *quantity = NULL, int *retCode = NULL) const;

virtual float supplyQuantity(const DtEntityType &supply) const;

virtual void changeSuppliesSize(int newSize);

virtual int indexInBounds(int ind) const;

// Name of the interaction (Not nec. the ClassName from the FOM).
virtual const char* name() const;

public:

    // Create a DtResupplyOfferInteraction.
    // Caller is responsible for deletion.
    static DtInteraction* create();

    // Add (register)/Remove (deregister) function to be called when
    // interaction occurs.
    static void addCallback(DtExerciseConn* conn,
        DtResupplyOfferInteractionCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtResupplyOfferInteractionCb cb, void* usr);

protected:

    DtGlobalObjectDesignator myReceivingObject;
    DtGlobalObjectDesignator mySupplyingObject;
    int myNumSupplyTypes;
    DtNetSupplyQuantity* mySupplies;
};
#endif
```

hResRecDec.h

```

#ifdef DtHLA

#include "interDecoder.h"

// This can be used to declare parameter processor functions
// which are static members of a class.

#define DtDECLARE_RESREC_PARAM_DECODER(paramName) \
    DtDECLARE_PARAM_DECODER(DtResupplyRecInteraction, paramName)

#define DtDEFINE_SIMPLE_RESREC_PARAM_DECODER( \
    paramName, netType, mutator) \
    DtDEFINE_SIMPLE_PARAM_DECODER( \
        DtResupplyReceivedDecoder, DtResupplyRecInteraction, \
        paramName, netType, mutator)

#define DtDEFINE_SIMPLE_RESREC_PARAM_DECODER_WITH_CAST( \
    paramName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_PARAM_DECODER_WITH_CAST( \
        DtResupplyReceivedDecoder, DtResupplyRecInteraction, \
        paramName, netType, mutator, castExpr)

// class DtResupplyReceivedDecoder:
//
// Instances of DtResupplyReceivedDecoder are used to decode Resup-
// plyReceived
// Interactions from the network.

class DT_DLL_VL DtResupplyReceivedDecoder : public DtInteractionDe-
coder
{
public:

    // constructor
    DtResupplyReceivedDecoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtResupplyReceivedDecoder();

    // copy constructor
    DtResupplyReceivedDecoder(const DtResupplyReceivedDecoder& orig);

    // assignment operator
    DtResupplyReceivedDecoder& operator=(const DtResupplyReceivedDe-
coder& orig);

```

protected:

```
    // Individual parameter decoders

    DtDECLARE_RESREC_PARAM_DECODER(ReceivingObject);
    DtDECLARE_RESREC_PARAM_DECODER(SupplyingObject);
    DtDECLARE_RESREC_PARAM_DECODER(Supplies);
};
#endif
```

hResRecEnc.h

```

#if DtHLA

#include "interEncoder.h"

#define DtDECLARE_RESREC_PARAM_ENCODER(paramName) \
    DtDECLARE_PARAM_ENCODER(DtResupplyRecInteraction, paramName)

#define DtDEFINE_SIMPLE_RESREC_PARAM_ENCODER(paramName, netType, \
    inspector) \
    DtDEFINE_SIMPLE_PARAM_ENCODER(DtResupplyReceivedEncoder, \
    DtResupplyRecInteraction, paramName, netType, inspector)

// class DtResupplyReceivedEncoder:
//
// DtResupplyReceivedEncoder is a DtInteractionEncoder that regis-
// ters
// parameter encoders for all of the ResupplyReceived interaction
// parameters.

class DT_DLL_VL DtResupplyReceivedEncoder : public DtInteractionEn-
coder
{
public:

    DtResupplyReceivedEncoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtResupplyReceivedEncoder();

    // copy constructor
    DtResupplyReceivedEncoder(const DtResupplyReceivedEncoder& orig);

    // assignment operator
    DtResupplyReceivedEncoder& operator=(const DtResupplyReceivedEn-
coder& orig);

protected:

    // Individual parameter encoders

    DtDECLARE_RESREC_PARAM_ENCODER(ReceivingObject);
    DtDECLARE_RESREC_PARAM_ENCODER(SupplyingObject);
    DtDECLARE_RESREC_PARAM_ENCODER(Supplies);
};
#endif

```

hResRecInter.h

```
#if DtHLA

#include "hIntWEncDec.h"
#include "hostStructs.h"
#include "mathTypes.h"
#include "packets/logisticsP.h"

#include "chooseDefines.h"

#define DtSUPPLY_VALID 1
#define DtSUPPLY_BAD_INDEX 0

// class DtResupplyRecInteraction:
//
// Instances of DtResupplyRecInteraction are used to represent
// ResupplyReceived Interactions in the RPR FOM.

// ResupplyReceived interaction callback function signature.
typedef void (*DtResupplyRecInteractionCb)
    (DtResupplyRecInteraction* inter, void* usr);

class DT_DLL_VL DtResupplyRecInteraction : public DtInteractionWith-
EncDec
{
public:

    // default constructor
    DtResupplyRecInteraction();

    // destructor
    virtual ~DtResupplyRecInteraction();

    // copy constructor
    DtResupplyRecInteraction(const DtResupplyRecInteraction& orig);

    // assignment operator
    DtResupplyRecInteraction& operator=(const DtResupplyRecInterac-
tion& orig);

    // Print the interaction's data.
    virtual void printData() const;

    // Set/Get the receiving object.
    virtual void setReceivingId(const DtGlobalObjectDesignator
&requester);
    virtual const DtGlobalObjectDesignator &receivingId() const;
```



```

    // Set/Get the repairing object.
    virtual void setSupplyingId(const DtGlobalObjectDesignator
&servicer);
    virtual const DtGlobalObjectDesignator &supplyingId() const;

    // Set/Get the supplies memory block all at once
    virtual void setSupplies(const void* data, int numBytes);
    virtual const void* supplies() const;

    // Supporting functions to get/set the Supply Array
    virtual void setNumSupplyTypes(int numTypes);
    virtual int numSupplyTypes() const;

    virtual void setNthSupply(int supplyIndex, const DtEntityType
&supply,
        float quantity, int *retCode = NULL);
    virtual const DtEntityType &nthSupply(int supplyIndex,
        float *quantity = NULL, int *retCode = NULL) const;

    virtual float supplyQuantity(const DtEntityType &supply) const;

    virtual void changeSuppliesSize(int newSize);

    virtual int indexInBounds(int ind) const;

    // Name of the interaction (Not nec. the ClassName from the FOM).
    virtual const char* name() const;

public:

    // Create a DtResupplyRecInteraction.
    // Caller is responsible for deletion.
    static DtInteraction* create();

    // Add (register)/Remove (deregister) function to be called when
    // interaction occurs.
    static void addCallback(DtExerciseConn* conn,
        DtResupplyRecInteractionCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtResupplyRecInteractionCb cb, void* usr);

protected:

    DtGlobalObjectDesignator myReceivingObject;
    DtGlobalObjectDesignator mySupplyingObject;
    int myNumSupplyTypes;
    DtNetSupplyQuantity* mySupplies;
};
#endif

```

hServReqDec.h

```
#if DtHLA

#include "interDecoder.h"

// This can be used to declare parameter processor functions
// which are static members of a class.

#define DtDECLARE_SERVREQ_PARAM_DECODER(paramName) \
    DtDECLARE_PARAM_DECODER(DtServiceRequestInteraction, paramName)

#define DtDEFINE_SIMPLE_SERVREQ_PARAM_DECODER( \
    paramName, netType, mutator) \
    DtDEFINE_SIMPLE_PARAM_DECODER( \
        DtServiceRequestDecoder, DtServiceRequestInteraction, \
        paramName, netType, mutator)

#define DtDEFINE_SIMPLE_SERVREQ_PARAM_DECODER_WITH_CAST( \
    paramName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_PARAM_DECODER_WITH_CAST( \
        DtServiceRequestDecoder, DtServiceRequestInteraction, \
        paramName, netType, mutator, castExpr)

// class DtServiceRequestDecoder:
//
// Instances of DtServiceRequestDecoder are used to decode Service-
// Request
// Interactions from the network.

class DT_DLL_VL DtServiceRequestDecoder : public DtInteractionDe-
coder
{
public:

    // constructor
    DtServiceRequestDecoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtServiceRequestDecoder();

    // copy constructor
    DtServiceRequestDecoder(const DtServiceRequestDecoder& orig);

    // assignment operator
    DtServiceRequestDecoder& operator=(const DtServiceRequestDecoder&
orig);
```

protected:

```
    // Individual parameter decoders

    DtDECLARE_SERVREQ_PARAM_DECODER(RequestingObject);
    DtDECLARE_SERVREQ_PARAM_DECODER(ServicingObject);
    DtDECLARE_SERVREQ_PARAM_DECODER(ServiceType);
    DtDECLARE_SERVREQ_PARAM_DECODER(Supplies);
};
#endif
```

hServReqEnc.h

```
#if DtHLA

#include "interEncoder.h"

#define DtDECLARE_SERVREQ_PARAM_ENCODER(paramName) \
    DtDECLARE_PARAM_ENCODER(DtServiceRequestInteraction, paramName)

#define DtDEFINE_SIMPLE_SERVREQ_PARAM_ENCODER(paramName, netType, \
    inspector) \
    DtDEFINE_SIMPLE_PARAM_ENCODER(DtServiceRequestEncoder, \
        DtServiceRequestInteraction, paramName, netType, inspector)

// class DtServiceRequestEncoder:
//
// DtServiceRequestEncoder is a DtInteractionEncoder that registers
// parameter encoders for all of the ServiceRequest interaction
// parameters.

class DT_DLL_VL DtServiceRequestEncoder : public DtInteractionEn-
coder
{
public:

    DtServiceRequestEncoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtServiceRequestEncoder();

    // copy constructor
    DtServiceRequestEncoder(const DtServiceRequestEncoder& orig);

    // assignment operator
    DtServiceRequestEncoder& operator=(const DtServiceRequestEncoder&
orig);

protected:

    // Individual parameter encoders

    DtDECLARE_SERVREQ_PARAM_ENCODER(RequestingObject);
    DtDECLARE_SERVREQ_PARAM_ENCODER(ServicingObject);
    DtDECLARE_SERVREQ_PARAM_ENCODER(ServiceType);
    DtDECLARE_SERVREQ_PARAM_ENCODER(Supplies);
};
#endif
```

hServReqInter.h

```

#if DtHLA

#include "hIntWEncDec.h"
#include "hostStructs.h"
#include "mathTypes.h"

#include "chooseDefines.h"
#include "packets/logisticsP.h"

#define DtSUPPLY_VALID 1
#define DtSUPPLY_BAD_INDEX 0

// class DtServiceRequestInteraction:
//
// Instances of DtServiceRequestInteraction are used to represent
// ServiceRequest Interactions in the RPR FOM.

// ServiceRequest interaction callback function signature.
typedef void (*DtServiceRequestInteractionCb)
    (DtServiceRequestInteraction* inter, void* usr);

class DT_DLL_VL DtServiceRequestInteraction : public DtInteraction-
WithEncDec
{
public:

    // default constructor
    DtServiceRequestInteraction();

    // destructor
    virtual ~DtServiceRequestInteraction();

    // copy constructor
    DtServiceRequestInteraction(const DtServiceRequestInteraction&
orig);

    // assignment operator
    DtServiceRequestInteraction& operator=(
        const DtServiceRequestInteraction& orig);

    // Print the interaction's data.
    virtual void printData() const;

    // Set/Get the receiving object.
    virtual void setRequestingId(const DtGlobalObjectDesignator
&requester);
    virtual const DtGlobalObjectDesignator &requestingId() const;

```

```
// Set/Get the repairing object.
virtual void setServicingId(const DtGlobalObjectDesignator
&servicer);
virtual const DtGlobalObjectDesignator &servicingId() const;

// Set/Get the service type.
virtual void setServiceType(DtServiceType type);
virtual DtServiceType serviceType() const;

// Set/Get the supplies memory block all at once
virtual void setSupplies(const void* data, int numBytes);
virtual const void* supplies() const;

// Supporting functions to get/set the Supply Array
virtual void setNumSupplyTypes(int numTypes);
virtual int numSupplyTypes() const;

virtual void setNthSupply(int supplyIndex, const DtEntityType
&supply,
    float quantity, int *retCode = NULL);
virtual const DtEntityType& nthSupply(int supplyIndex,
    float *quantity = NULL, int *retCode = NULL) const;

virtual float supplyQuantity(const DtEntityType &supply) const;

virtual void changeSuppliesSize(int newSize);

virtual int indexInBounds(int ind) const;

// Name of the interaction (Not nec. the ClassName from the FOM).
virtual const char* name() const;

public:

    // Create a DtServiceRequestInteraction.
    // Caller is responsible for deletion.
    static DtInteraction* create();

    // Add (register)/Remove (deregister) function to be called when
    // interaction occurs.
    static void addCallback(DtExerciseConn* conn,
        DtServiceRequestInteractionCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtServiceRequestInteractionCb cb, void* usr);

protected:
    DtGlobalObjectDesignator myRequestingObject;
    DtGlobalObjectDesignator myServicingObject;
    DtServiceType myServiceType;
```

```
    int myNumSupplyTypes;  
    DtNetSupplyQuantity* mySupplies;  
};  
#endif
```

hSetDataDec.h

```
#if DtHLA

#include "interDecoder.h"

// This can be used to declare parameter processor functions
// which are static members of a class.

#define DtDECLARE_SETDATA_PARAM_DECODER(paramName) \
    DtDECLARE_PARAM_DECODER(DtSetDataInteraction, paramName)

#define DtDEFINE_SIMPLE_SETDATA_PARAM_DECODER( \
    paramName, netType, mutator) \
    DtDEFINE_SIMPLE_PARAM_DECODER( \
        DtSetDataDecoder, DtSetDataInteraction, \
        paramName, netType, mutator)

#define DtDEFINE_SIMPLE_SETDATA_PARAM_DECODER_WITH_CAST( \
    paramName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_PARAM_DECODER_WITH_CAST( \
        DtSetDataDecoder, DtSetDataInteraction, \
        paramName, netType, mutator, castExpr)

// class DtSetDataDecoder:
//
// Instances of DtSetDataDecoder are used to decode SetData
// Interactions from the network.

class DT_DLL_VL DtSetDataDecoder : public DtInteractionDecoder
{
public:

    // constructor
    DtSetDataDecoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtSetDataDecoder();

    // copy constructor
    DtSetDataDecoder(const DtSetDataDecoder& orig);

    // assignment operator
    DtSetDataDecoder& operator=(const DtSetDataDecoder& orig);

protected:

    // Individual parameter decoders
```



```
DtDECLARE_SETDATA_PARAM_DECODER(OriginatingEntity);
DtDECLARE_SETDATA_PARAM_DECODER(ReceivingEntity);
DtDECLARE_SETDATA_PARAM_DECODER(RequestID);
DtDECLARE_SETDATA_PARAM_DECODER(FixedDatums);
DtDECLARE_SETDATA_PARAM_DECODER(VariableDatumSet);
};
#endif
```

hSetDataEnc.h

```
#if DtHLA

#include "interEncoder.h"

#define DtDECLARE_SETDATA_PARAM_ENCODER(paramName) \
    DtDECLARE_PARAM_ENCODER(DtSetDataInteraction, paramName)

#define DtDEFINE_SIMPLE_SETDATA_PARAM_ENCODER(paramName, netType, \
    inspector) \
    DtDEFINE_SIMPLE_PARAM_ENCODER(DtSetDataEncoder, \
        DtSetDataInteraction, paramName, netType, inspector)

// class DtSetDataEncoder:
//
// DtSetDataEncoder is a DtInteractionEncoder that registers
// parameter encoders for all of the SetData interaction parameters.

class DT_DLL_VL DtSetDataEncoder : public DtInteractionEncoder
{
public:

    DtSetDataEncoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtSetDataEncoder();

    // copy constructor
    DtSetDataEncoder(const DtSetDataEncoder& orig);

    // assignment operator
    DtSetDataEncoder& operator=(const DtSetDataEncoder& orig);

protected:

    // Individual parameter encoders

    DtDECLARE_SETDATA_PARAM_ENCODER(OriginatingEntity);
    DtDECLARE_SETDATA_PARAM_ENCODER(ReceivingEntity);
    DtDECLARE_SETDATA_PARAM_ENCODER(RequestID);
    DtDECLARE_SETDATA_PARAM_ENCODER(FixedDatums);
    DtDECLARE_SETDATA_PARAM_ENCODER(VariableDatumSet);
};
#endif
```

hSetDataInter.h

```

#if DtHLA

#include "hInterWData.h"
#include "hostStructs.h"
#include "mathTypes.h"

#include "chooseDefines.h"

// class DtSetDataInteraction:
//
// Instances of DtSetDataInteraction are used to represent
// SetData Interactions in the RPR FOM. The variable and fixed
// datum records are accessed through DtInteractionWithData member
// functions.

// SetData interaction callback function signature.
typedef void (*DtSetDataInteractionCb)
    (DtSetDataInteraction* inter, void* usr);

class DT_DLL_VL DtSetDataInteraction : public DtInteractionWithData
{
public:

    // default constructor
    DtSetDataInteraction();

    // destructor
    virtual ~DtSetDataInteraction();

    // copy constructor
    DtSetDataInteraction(const DtSetDataInteraction& orig);

    // assignment operator
    DtSetDataInteraction& operator=(
        const DtSetDataInteraction& orig);

    // Print the interaction's data.
    virtual void printData() const;

    // Set/Get sender ID
    virtual void setSenderId(const DtEntityIdentifier& id);
    virtual const DtEntityIdentifier& senderId() const;

    // Set/Get receiver ID
    virtual void setReceiverId(const DtEntityIdentifier& id);
    virtual const DtEntityIdentifier& receiverId() const;

```

```
// Set/Get request ID
virtual void setRequestId(unsigned long id);
virtual unsigned long requestId() const;

// Name of the interaction (Not nec. the ClassName from the FOM).
virtual const char* name() const;

public:

    // Create a DtSetDataInteraction.
    // Caller is responsible for deletion.
    static DtInteraction* create();

    // Add (register)/Remove (deregister) function to be called when
    // interaction occurs.
    static void addCallback(DtExerciseConn* conn,
        DtSetDataInteractionCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtSetDataInteractionCb cb, void* usr);

protected:

    DtEntityIdentifier myOrigEnt;
    DtEntityIdentifier myReceiveEnt;
    unsigned long myRequestId;
};
#endif
```

hSignalInter.h

```

#if DtHLA

#include "chooseDefines.h"
#include "hIntWEncDec.h"
#include "hostStructs.h"

// class DtSignalInteraction:
//
// Instances of DtSignalInteraction are

// Collision interaction callback function signature.
typedef void (*DtSignalInteractionCb)
    (DtSignalInteraction* inter, void* usr);

class DT_DLL_VL DtSignalInteraction : public DtInteractionWithEncDec
{
public:

    // default constructor
    DtSignalInteraction();

    // destructor
    virtual ~DtSignalInteraction();

    // copy constructor
    DtSignalInteraction(const DtSignalInteraction& orig);

    // assignment operator
    DtSignalInteraction& operator=(const DtSignalInteraction& orig);

    // Print the interaction's data.
    virtual void printData() const;

    // Set/Get Host radio id
    virtual void setRadioId(const DtGlobalObjectDesignator& id);
    virtual const DtGlobalObjectDesignator& radioId() const;

    // Set/Get data rate
    virtual void setSampleRate(DtU32 rate);
    virtual DtU32 sampleRate() const;

    // Set/Get sample count
    virtual void setNumSamples(DtU16 count);
    virtual DtU16 numSamples() const;

    // Set/Get signal data length in bytes or bits.
    virtual void setNumDataBytes(DtU16 len);

```

```
virtual DtU16 numDataBytes() const;
virtual void setNumDataBits(DtU16 len);
virtual DtU16 numDataBits() const;

// Set/Get signal data
virtual void setData(const void* data, int numBytes);
virtual const void* data() const;

// Set/Get the byteNum'th data byte
virtual DtU8 dataByte(int byteNum) const;
virtual void setDataByte(int byteNum, DtU8 data);

// Set/Get tactical data link
virtual void setTdlType(DtTacticalDataLinkType link);
virtual DtTacticalDataLinkType tdlType() const;

// Set/Get encoding type
virtual void setEncodingType(DtEncodingType encoding);
virtual DtEncodingType encodingType() const;

// Set/Get encoding class
virtual DtEncodingClass encodingClass() const;
virtual void setEncodingClass(DtEncodingClass encodingClass);

// Set/Get TDL message count
virtual void setTdlMessageCount(DtU16 count);
virtual DtU16 tdlMessageCount() const;

// Set/Get user protocol id
virtual void setProtocolId(DtUserProtocol id);
virtual DtUserProtocol protocolId() const;

// Set/Get database index
virtual void setDatabaseIndex(DtU32 index);
virtual DtU32 databaseIndex() const;

// Set/Get duration
virtual void setDuration(DtU32 duration);
virtual DtU32 duration() const;

// Set/Get start offset
virtual void setStartOffset(DtU32 offset);
virtual DtU32 startOffset() const;

// Name of the interaction (Not nec. the ClassName from the FOM).
virtual const char* name() const;

public:

    // Create a DtSignalInteraction.
```

```
// Caller is responsible for deletion.
static DtInteraction* create();

// Add (register)/Remove (deregister) function to be called when
// interaction occurs.
static void addCallback(DtExerciseConn* conn,
    DtSignalInteractionCb cb, void* usr);
static void removeCallback(DtExerciseConn* conn,
    DtSignalInteractionCb cb, void* usr);

protected:

    DtGlobalObjectDesignator    myHostRadioId;
    DtU16                       mySignalDataLength;    // bits
    DtU8*                       mySignalData;
    DtTacticalDataLinkType      myTacticalDataLink;
    DtU16                       myTDLMessageCount;
    DtUserProtocol              myUserProtocolId;
    DtU32                       myDatabaseIndex;
    DtU32                       myDuration;
    DtU32                       myStartOffset;
    DtEncodingType              myEncodingType;
    DtEncodingClass             myEncodingClass;
    DtU32                       mySampleRate;
    DtU16                       mySampleCount;
};
#endif    //DtHLA
```


protected:

```
    // Individual parameter decoders

    DtDECLARE_START_PARAM_DECODER(OriginatingEntity);
    DtDECLARE_START_PARAM_DECODER(ReceivingEntity);
    DtDECLARE_START_PARAM_DECODER(RequestID);
    DtDECLARE_START_PARAM_DECODER(RealWorldTime);
    DtDECLARE_START_PARAM_DECODER(SimulationTime);
};
#endif
```

hStartEnc.h

```
#if DtHLA

#include "interEncoder.h"

#define DtDECLARE_START_PARAM_ENCODER(paramName) \
    DtDECLARE_PARAM_ENCODER(DtStartResumeInteraction, paramName)

#define DtDEFINE_SIMPLE_START_PARAM_ENCODER(paramName, netType, \
    inspector) \
    DtDEFINE_SIMPLE_PARAM_ENCODER(DtStartResumeEncoder, \
        DtStartResumeInteraction, paramName, netType, inspector)

// class DtStartResumeEncoder:
//
// DtStartResumeEncoder is a DtInteractionEncoder that registers
// parameter encoders for all of the START/RESUME interaction param-
// eters.

class DT_DLL_VL DtStartResumeEncoder : public DtInteractionEncoder
{
public:

    DtStartResumeEncoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtStartResumeEncoder();

    // copy constructor
    DtStartResumeEncoder(const DtStartResumeEncoder& orig);

    // assignment operator
    DtStartResumeEncoder& operator=(const DtStartResumeEncoder&
orig);

protected:

    // Individual parameter encoders

    DtDECLARE_START_PARAM_ENCODER(OriginatingEntity);
    DtDECLARE_START_PARAM_ENCODER(ReceivingEntity);
    DtDECLARE_START_PARAM_ENCODER(RequestID);
    DtDECLARE_START_PARAM_ENCODER(RealWorldTime);
    DtDECLARE_START_PARAM_ENCODER(SimulationTime);
};
#endif
```

hStartInter.h

```

#if DtHLA

#include "hIntWEncDec.h"
#include "hostStructs.h"
#include "mathTypes.h"

#include "chooseDefines.h"

// class DtStartResumeInteraction:
//
// Instances of DtStartResumeInteraction are used to represent
// StartResume Interactions in the RPR ROM.

// StartResume interaction callback function signature.
typedef void (*DtStartResumeInteractionCb) (DtStartResumeInteraction* inter,
void* usr);

class DT_DLL_VL DtStartResumeInteraction : public DtInteractionWithEncDec
{
public:

    // default constructor
    DtStartResumeInteraction();

    // destructor
    virtual ~DtStartResumeInteraction();

    // copy constructor
    DtStartResumeInteraction(const DtStartResumeInteraction& orig);

    // assignment operator
    DtStartResumeInteraction& operator=(const DtStartResumeInteraction& orig);

    // Print the interaction's data.
    virtual void printData() const;

    // Set/Get sender ID
    virtual void setSenderId(const DtEntityIdentifier& id);
    virtual const DtEntityIdentifier& senderId() const;

    // Set/Get receiver ID
    virtual void setReceiverId(const DtEntityIdentifier& id);
    virtual const DtEntityIdentifier& receiverId() const;

```

```
// Set/Get real world time
virtual void setRealWorldTime(const DtTime time);
virtual DtTime realWorldTime() const;

// Set/Get simulation time
virtual void setSimulationTime(const DtTime time);
virtual DtTime simulationTime() const;

// Set/Get request ID
virtual void setRequestId(int id);
virtual int requestId() const;

// Name of the interaction (Not nec. the ClassName from the FOM).
virtual const char* name() const;

public:

    // Create a DtStartResumeInteraction.
    // Caller is responsible for deletion.
    static DtInteraction* create();

    // Add (register)/Remove (deregister) function to be called when
    // interaction occurs.
    static void addCallback(DtExerciseConn* conn,
        DtStartResumeInteractionCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtStartResumeInteractionCb cb, void* usr);

protected:

    DtEntityIdentifier myOrigEnt;
    DtEntityIdentifier myReceiveEnt;
    int myRequestId;
    DtTime myRealTime;
    DtTime mySimTime;
};
#endif
```

hStateDecoder.h

```

#if DtHLA

#include "RTI.hh"
#include "vfprintf.h"

#include "chooseDefines.h"

// class DtHlaStateDecoder:
//
// DtHlaStateDecoder is used to provide decoding of a state message
// into a state repository in an HLA (as opposed to DIS) environ-
// ment.

class DT_DLL_VL DtHlaStateDecoder
{
public:

    // constructor
    DtHlaStateDecoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtHlaStateDecoder();

    // copy constructor
    DtHlaStateDecoder(const DtHlaStateDecoder& orig);

    // assignment operator
    DtHlaStateDecoder& operator=(const DtHlaStateDecoder& orig);

    // Attribute decoding function declaration.
    typedef void (*DtAttributeDecoder) (
        DtStateRepository* stateRep,
        const RTI::AttributeHandleValuePairSet& avlist,
        int pairSetIndex);

    // Add/Remove a decoding function for the attribute specified
    // by handle.
    // Returns the decoding function which is replaced by the
    add/remove.
    // Note: This is non-virtual since it is typically called by
    // constructors.
    DtAttributeDecoder addDecoder(RTI::AttributeHandle h,
        DtAttributeDecoder func);
    DtAttributeDecoder removeDecoder(RTI::AttributeHandle h);

```

```
// Add/Remove a decoding function for the attribute specified
// by name. Complain flag determines whether to print a warning
// if the attribute does not exist in the FOM.
// Returns the decoding function which is replaced by the
add/remove.
// Note: This is non-virtual since it is typically called by
// constructors.
DtAttributeDecoder addDecoder(char* name,
    DtAttributeDecoder func, DtBoolean complain = DtTrue);
DtAttributeDecoder removeDecoder(char* name, DtBoolean complain =
DtTrue);

// Get the attribute decoder for the specified handle.
virtual DtAttributeDecoder decoder(int handle) const;

// Decode the state message into the state repository.
virtual void decode(const DtStateMsg& msg,
    DtStateRepository* stateRep) const;

// Get the object's class handle.
virtual RTI::ObjectClassHandle objectClassHandle() const;

// Returns a new-ed clone of this object
virtual DtHlaStateDecoder* clone() const;

// Set the class descriptor of the encoder
virtual void setClassDesc(DtObjClassDesc* desc);

public:

// Returns a pointer to a large static buffer that can be used to
// copy data obtained from a phvps.
static char* decodingBuffer();

protected:

// Decode the attribute at the specified index in the attribute
// handle pair set.
virtual void decodeAttribute(DtStateRepository* stateRep,
    const RTI::AttributeHandleValuePairSet& attrs,
    int pairSetIndex) const;

protected:

DtExerciseConn* myExConn;
DtObjClassDesc* myClassDesc;

DtVrIRtiAmbassador* myRtiAmb;
RTI::ObjectClassHandle myHandle;
```

```

    // myDecoderList is an array of theMaxAttributesPerObject items.
    DtAttributeDecoder* myDecoderList;

protected:

    static const unsigned int theMaxAttributesPerObject;
    static char theDecBuffer[2000];
};
// This macro is used to add attribute decoder functions
// (functions that decode individual attributes) to a DtHlaStateDe-
// coder
#define DtADD_ATTR_DECODER(attrName) \
    addDecoder(#attrName, (DtAttributeDecoder) decode##attrName);

// Same as above, but don't complain if name is not found.
#define DtADD_ATTR_DECODER_QUIET(attrName) \
    addDecoder(#attrName, (DtAttributeDecoder) decode##attrName,
DtFalse);

// This macro is used to declare attribute decoder functions
// (members of a derived DtHlaStateDecoder that decode
// individual attributes) to a DtHlaStateDecoder
#define DtDECLARE_ATTR_DECODER(stateRepClass, attrName) \
    static void decode##attrName(stateRepClass* stateRep, \
        const RTI::AttributeHandleValuePairSet& attrs, int pairSet-
Index)

// These macros are used to define simple attribute decoder func-
// tions
// (members of a derived DtHlaStateDecoder that decode
// individual attributes). The simple decoding is just to
// convert the value pulled out of the ahvps to a host type using
// the cast expression castExpr, and pass it to the mutator func-
// tion.
#define DtDEFINE_SIMPLE_ATTR_DECODER_WITH_CAST( \
    decClass, stateRepClass, attrName, netType, mutator, castExpr) \
void decClass::decode##attrName( \
    stateRepClass* stateRep, \
    const RTI::AttributeHandleValuePairSet& attrs, \
    int pairSetIndex) \
{ \
    RTI::ULong length = 0; \
    netType* netVal = (netType*) attrs.getValuePointer(pairSetIndex,
length); \
    if (length > sizeof(netType)) \
    { \
        DtWarn("Size of value decoded from RTI update for attribute %s
(%d)\n", \
            #attrName, length); \
    } \
}

```

```
        DtWarn(" is larger than size of %s (%d)\n", #netType,
sizeof(netType)); \
    } \
    stateRep->mutator(castExpr* netVal); \
}

#define DtDEFINE_SIMPLE_ATTR_DECODER( \
    decClass, stateRepClass, attrName, netType, mutator) \
    DtDEFINE_SIMPLE_ATTR_DECODER_WITH_CAST(decClass, stateRepClass, \
        attrName, netType, mutator, (netType))

#endif
```


hStateEncoder.h

```

#if DtHLA

#include "RTI.hh"
#include "MachineTypes.h"

#include "chooseDefines.h"

// class DtHlaStateEncoder:
//
// DtHlaStateEncoder is used to provide encoding of a state repository
// into a state message in an HLA (as opposed to DIS) environment.

class DT_DLL_VL DtHlaStateEncoder
{
public:

    // constructor
    DtHlaStateEncoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtHlaStateEncoder();

    // copy constructor
    DtHlaStateEncoder(const DtHlaStateEncoder& orig);

    // assignment operator
    DtHlaStateEncoder& operator=(const DtHlaStateEncoder& orig);

    // Attribute encoding function declaration.
    typedef void (*DtAttributeEncoder)(
        const DtStateRepository& stateRep,
        RTI::AttributeHandleValuePairSet* avList,
        RTI::AttributeHandle handle);

    // Attribute checking function declaration.
    typedef DtBoolean (*DtAttributeChecker)(
        const DtStateRepository& stateRep,
        const DtStateRepository& asSeenByRemote);

    // Add/Remove an encoding function for the attribute specified
    // by handle.
    // Returns the encoding function which is replaced by the
    add/remove.
    // Note: This is non-virtual since it is typically called by

```

```
//      constructors.
DtAttributeEncoder addEncoder(RTI::AttributeHandle hand,
    DtAttributeEncoder func);
DtAttributeEncoder removeEncoder(RTI::AttributeHandle hand);

// Add/Remove an encoding function for the attribute specified
// by name. Complain flag determines whether to print a warning
// if the attribute does not exist in the FOM.
// Returns the encoding function which is replaced by the
add/remove.
// Note: This is non-virtual since it is typically called by
//      constructors.
DtAttributeEncoder addEncoder(char* name,
    DtAttributeEncoder func, DtBoolean complain = DtTrue);
DtAttributeEncoder removeEncoder(char* name, DtBoolean complain =
DtTrue);

// Add/Remove a checker function for the attribute specified
// by handle.
// Returns the checker function which is replaced by the
add/remove.
// Note: This is non-virtual since it is typically called by
//      constructors.
DtAttributeChecker addChecker(RTI::AttributeHandle hand,
    DtAttributeChecker func);
DtAttributeChecker removeChecker(RTI::AttributeHandle hand);

// Add/Remove a checker function for the attribute specified
// by name. Complain flag determines whether to print a warning
// if the attribute does not exist in the FOM.
// Returns the checker function which is replaced by the
add/remove.
// Note: This is non-virtual since it is typically called by
//      constructors.
DtAttributeChecker addChecker(char* name,
    DtAttributeChecker func, DtBoolean complain = DtTrue);
DtAttributeChecker removeChecker(char* name, DtBoolean complain =
DtTrue);

// Get the attribute encoder for the specified handle.
virtual DtAttributeEncoder encoder(int handle) const;

// Get the attribute checker for the specified handle.
virtual DtAttributeChecker checker(int handle) const;

// Encode necessary values from the state repository.
// The second state repository represents what
// is seen by remote simulators and is used to determine when
// things need to be sent in the state message. includeThese
// is a set of attributes that should be included even if their
```

```

// thresholds have not been exceeded.
virtual void encode(const DtStateRepository& stateRep,
    const DtStateRepository& asSeenByRemote,
    const RTI::AttributeHandleSet& neededAttrs,
    const RTI::AttributeHandleSet& includeThese,
    DtStateMsg* msg);

// Get the object's class handle.
virtual RTI::ObjectClassHandle objectClassHandle();

// Returns a new-ed clone of this object
virtual DtHlaStateEncoder* clone() const;

// Set the class descriptor of the encoder
virtual void setClassDesc(DtObjClassDesc* desc);

public:

    // A checker function that always returns 1, meaning that the
    // attribute for which it is being used will be deemed
    // necessary to send on every tick.
    static DtBoolean trueChecker(const DtStateRepository& stateRep,
        const DtStateRepository& asSeenByRemote);

    // A checker function that always returns 0, meaning that the
    // attribute for which it is being used will never be deemed
    // necessary to send.
    static DtBoolean falseChecker(const DtStateRepository& stateRep,
        const DtStateRepository& asSeenByRemote);

protected:

    DtExerciseConn* myExConn;
    DtObjClassDesc* myClassDesc;

    DtVrLrtiAmbassador* myRtiAmb;
    RTI::InteractionClassHandle myHandle;

    // myEncoderList and myCheckerList are arrays
    // of theMaxParametersPerInteraction items.
    DtAttributeEncoder* myEncoderList;
    DtAttributeChecker* myCheckerList;

protected:

    static const int theMaxAttributesPerObject;
};
// This macro is used to add attribute encoder functions
// (functions that encode individual attributes) to a DtHlaStateEn-
coder

```

```
#define DtADD_ATTR_ENCODER(attrName) \
    addEncoder(#attrName, (DtAttributeEncoder) encode##attrName)

// Same as above, but don't complain if name is not found.
#define DtADD_ATTR_ENCODER_QUIET(attrName) \
    addEncoder(#attrName, (DtAttributeEncoder) encode##attrName,
DtFalse)

// This macro is used to add attribute checker functions
// (functions that determine whether particular attributes need to
// be included in an outgoing state update) to a DtHlaStateEncoder
#define DtADD_ATTR_CHECKER(attrName) \
    addChecker(#attrName, (DtAttributeChecker) need##attrName)

// Same as above, but don't complain if name is not found.
#define DtADD_ATTR_CHECKER_QUIET(attrName) \
    addChecker(#attrName, (DtAttributeChecker) need##attrName,
DtFalse)

// This macro is used to declare attribute encoder functions
// (members of a derived DtHlaStateEncoder that encode
// individual attributes) to a DtHlaStateEncoder
#define DtDECLARE_ATTR_ENCODER(stateRepClass, attrName) \
    static void encode##attrName(const stateRepClass& rep, \
        RTI::AttributeHandleValuePairSet* attrs, \
        RTI::AttributeHandle attrHandle)

// This is used to declare static member functions for determining
// whether or not an attribute needs to be encoded.
#define DtDECLARE_ATTR_CHECKER(stateRepClass, attrName) \
    static DtBoolean need##attrName(const stateRepClass& stateRep, \
        const stateRepClass& asSeenByRemote)

// This macro is used to define simple attribute encoder functions
// (members of a derived DtHlaStateEncoder that encode
// individual attributes). The simple encoding is just to implic-
// itly
// convert the value returned by the inspector to a netType,
// and add it to the ahvps.
#define DtDEFINE_SIMPLE_ATTR_ENCODER( \
    encClass, stateRepClass, attrName, netType, inspector) \
void encClass::encode##attrName(const stateRepClass& rep, \
    RTI::AttributeHandleValuePairSet* attrs, \
    RTI::AttributeHandle attrHandle) \
{ \
    netType netVal; \
    netVal = rep.inspector(); \
    attrs->add(attrHandle, (char*) &netVal, sizeof(netType)); \
}
```

```
// This macro is used to define simple attribute checker functions
// (members of a derived DtHlaStateEncoder that check whether
// a particular attribute is out of date. The simple implementation
// is to just compare the values returned by the two StateReps'
// inspector functions.
#define DtDEFINE_SIMPLE_ATTR_CHECKER( \
    encClass, stateRepClass, attrName, inspector) \
DtBoolean encClass::need##attrName(const stateRepClass& stateRep, \
    const stateRepClass& asSeenByRemote) \
{ \
    return (DtBoolean) !(stateRep.inspector() == asSeenByRe- \
mote.inspector()); \
}

#endif
```

hStateMsg.h

```
#if DtHLA

#include "RTI.hh"

#include "chooseDefines.h"
#include "vlTime.h"
#include "tStampType.h"
#include "str.h"

// class DtStateMsg:
//
// Instances of DtStateMsg are used to represent state data
// communicated via the network in an HLA (as opposed to DIS) envi-
// ronment.

#if DtRTI103
#define ObjectHandle ObjectID
#endif

class DT_DLL_VL DtStateMsg
{
public:

    // constructor
    DtStateMsg(RTI::ObjectHandle id,
               const RTI::AttributeHandleValuePairSet& attrs);

    // constructor
    DtStateMsg(RTI::ObjectHandle id,
               RTI::AttributeHandleValuePairSet* attrs);

    // destructor
    virtual ~DtStateMsg();

    // copy constructor
    DtStateMsg(const DtStateMsg& orig);

    // assignment operator
    DtStateMsg& operator=(const DtStateMsg& orig);

    // Get the attribute handle value pair set. (Const version)
    virtual const RTI::AttributeHandleValuePairSet& ahvps() const;

    // Get the attribute handle value pair set. (Non-const version)
    virtual RTI::AttributeHandleValuePairSet* ahvps();

    // Get the object identifier.
```

```

virtual RTI::ObjectHandle objectId() const;

// Prints prints header and attributes with values
virtual void print(DtExerciseConn* exConn,
    RTI::ObjectClassHandle h) const;

// Uses the object to print host representation of data
virtual void print(DtHlaObject *obj) const;

// prints the header information
virtual void printHeader(DtExerciseConn* exConn,
    RTI::ObjectClassHandle classHandle) const;

// Print the attributes (with hex values if withHex is 1)
virtual void printAttrs(DtExerciseConn* exConn,
    RTI::ObjectClassHandle classHandle, int withHex) const;

// Set/Get the user-defined tag.
virtual void setTag(const DtString& tag);
virtual const DtString& tag() const;

// Set the time of validity of the message, and whether this
// represents relative or absolute time. Time is in seconds, but
only
// time modulo an hour is stored.
virtual void setTimeStamp(DtTime time,
    DtTimeStampType type = DtTimeStampRelative);

// Return the time of validity of the message (in seconds past an
hour)
// as set by setTimeStamp.
virtual DtTime timeStamp() const;

// Return the type of the time stamp.
virtual DtTimeStampType timeStampType() const;

// Since timestamp only represents number of seconds past the
hour,
// guessTimeValid chooses the hour that would make the timestamp
// closest to reference time, and returns a time that is times-
tamp
// seconds past that hour.
virtual DtTime guessTimeValid(DtTime referenceTime) const;

public:

// Set/Get flag that tells us whether we're using the RPR FOM
// convention that the UserDefinedTag in updates and interactions
// represents a DIS-style timestamp.
static void setTagIsAsciiTime(DtBoolean val);

```

```
    static DtBoolean tagIsAsciiTime();

protected:

    DtString myTag;
    RTI::AttributeHandleValuePairSet* myAvList;
    RTI::ObjectHandle myId;
    DtTime myTime;
    DtTimeStampType myTimeType;

protected:

    static DtBoolean theTagIsAsciiTime;
};
#endif
```


hStopDec.h

```

#if DtHLA

#include "interDecoder.h"

#include "chooseDefines.h"

// This can be used to declare parameter processor functions
// which are static members of a class.

#define DtDECLARE_STOP_PARAM_DECODER(paramName) \
    DtDECLARE_PARAM_DECODER(DtStopFreezeInteraction, paramName)

#define DtDEFINE_SIMPLE_STOP_PARAM_DECODER( \
    paramName, netType, mutator) \
    DtDEFINE_SIMPLE_PARAM_DECODER( \
        DtStopFreezeDecoder, DtStopFreezeInteraction, \
        paramName, netType, mutator)

#define DtDEFINE_SIMPLE_STOP_PARAM_DECODER_WITH_CAST( \
    paramName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_PARAM_DECODER_WITH_CAST( \
        DtStopFreezeDecoder, DtStopFreezeInteraction, \
        paramName, netType, mutator, castExpr)

// class DtStopFreezeDecoder:
//
// Instances of DtStopFreezeDecoder are used to decode StopFreeze
// Interactions from the network.

class DT_DLL_VL DtStopFreezeDecoder : public DtInteractionDecoder
{
public:

    // constructor
    DtStopFreezeDecoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtStopFreezeDecoder();

    // copy constructor
    DtStopFreezeDecoder(const DtStopFreezeDecoder& orig);

    // assignment operator
    DtStopFreezeDecoder& operator=(const DtStopFreezeDecoder& orig);

protected:

```

```
// Individual parameter decoders

DtDECLARE_STOP_PARAM_DECODER(OriginatingEntity);
DtDECLARE_STOP_PARAM_DECODER(ReceivingEntity);
DtDECLARE_STOP_PARAM_DECODER(RequestID);
DtDECLARE_STOP_PARAM_DECODER(RealWorldTime);
DtDECLARE_STOP_PARAM_DECODER(Reason);
DtDECLARE_STOP_PARAM_DECODER(ReflectValues);
DtDECLARE_STOP_PARAM_DECODER(RunInternalSimulationClock);
DtDECLARE_STOP_PARAM_DECODER(UpdateAttributes);
};
#endif
```

hStopEnc.h

```

#if    DtHLA

#include "interEncoder.h"

#define DtDECLARE_STOP_PARAM_ENCODER(paramName) \
        DtDECLARE_PARAM_ENCODER(DtStopFreezeInteraction, paramName)

#define DtDEFINE_SIMPLE_STOP_PARAM_ENCODER(paramName, netType,
inspector) \
        DtDEFINE_SIMPLE_PARAM_ENCODER(DtStopFreezeEncoder, \
        DtStopFreezeInteraction, paramName, netType, inspector)

// class DtStopFreezeEncoder:
//
// DtStopFreezeEncoder is a DtInteractionEncoder that registers
// parameter encoders for all of the STOP/FREEZE interaction param-
// eters.

class DT_DLL_VL DtStopFreezeEncoder : public DtInteractionEncoder
{
public:

        DtStopFreezeEncoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

        // destructor
        virtual ~DtStopFreezeEncoder();

        // copy constructor
        DtStopFreezeEncoder(const DtStopFreezeEncoder& orig);

        // assignment operator
        DtStopFreezeEncoder& operator=(const DtStopFreezeEncoder& orig);

protected:

        // Individual parameter encoders

        DtDECLARE_STOP_PARAM_ENCODER(OriginatingEntity);
        DtDECLARE_STOP_PARAM_ENCODER(ReceivingEntity);
        DtDECLARE_STOP_PARAM_ENCODER(RequestID);
        DtDECLARE_STOP_PARAM_ENCODER(RealWorldTime);
        DtDECLARE_STOP_PARAM_ENCODER(Reason);
        DtDECLARE_STOP_PARAM_ENCODER(ReflectValues);
        DtDECLARE_STOP_PARAM_ENCODER(RunInternalSimulationClock);
        DtDECLARE_STOP_PARAM_ENCODER(UpdateAttributes);
};
#endif

```

hStopInter.h

```
#if DtHLA

#include "hIntWEncDec.h"
#include "hostStructs.h"
#include "mathTypes.h"

#include "chooseDefines.h"

// class DtStopFreezeInteraction:
//
// Instances of DtStopFreezeInteraction are used to represent
// StopFreeze Interactions in the RPR ROM.

// StopFreeze interaction callback function signature.
typedef void (*DtStopFreezeInteractionCb) (DtStopFreezeInteraction* inter,
void* usr);

class DT_DLL_VL DtStopFreezeInteraction : public DtInteractionWithEncDec
{
public:

    // default constructor
    DtStopFreezeInteraction();

    // destructor
    virtual ~DtStopFreezeInteraction();

    // copy constructor
    DtStopFreezeInteraction(const DtStopFreezeInteraction& orig);

    // assignment operator
    DtStopFreezeInteraction& operator=(const DtStopFreezeInteraction& orig);

    // Print the interaction's data.
    virtual void printData() const;

    // Set/Get sender ID
    virtual void setSenderId(const DtEntityIdentifier& id);
    virtual const DtEntityIdentifier& senderId() const;

    // Set/Get receiver ID
    virtual void setReceiverId(const DtEntityIdentifier& id);
    virtual const DtEntityIdentifier& receiverId() const;
```

```

// Set/Get real world time
virtual void setRealWorldTime(const DtTime time);
virtual DtTime realWorldTime() const;

// Set/Get request ID
virtual void setRequestId(int id);
virtual int requestId() const;

// Set/Get stopReason
virtual DtStopReason reason() const;
virtual void setReason(DtStopReason reason);

// Set/Get reflectWhenFrozen flag
virtual DtBoolean reflectWhenFrozen() const;
virtual void setReflectWhenFrozen(DtBoolean flag);

// Set/Get runClockWhenFrozen flag
virtual DtBoolean runClockWhenFrozen() const;
virtual void setRunClockWhenFrozen(DtBoolean flag);

// Set/Get updateWhenFrozen flag
virtual DtBoolean updateWhenFrozen() const;
virtual void setUpdateWhenFrozen(DtBoolean flag);

// Name of the interaction (Not nec. the ClassName from the FOM).
virtual const char* name() const;

```

public:

```

// Create a DtStopFreezeInteraction.
// Caller is responsible for deletion.
static DtInteraction* create();

// Add (register)/Remove (deregister) function to be called when
// interaction occurs.
static void addCallback(DtExerciseConn* conn,
    DtStopFreezeInteractionCb cb, void* usr);
static void removeCallback(DtExerciseConn* conn,
    DtStopFreezeInteractionCb cb, void* usr);

```

protected:

```

DtEntityIdentifier myOrigEnt;
DtEntityIdentifier myReceiveEnt;
int myRequestId;
DtTime myRealTime;
DtStopReason myReason;
DtBoolean myReflect;
DtBoolean myUpdate;
DtBoolean myRunClock;

```

```
};  
#endif
```

hUnknownInter.h

```

#ifdef DtHLA

#include "hInteraction.h"
#include "hostStructs.h"
#include "exerciseConn.h"

#include "chooseDefines.h"

// class DtUnknownInteraction:
//
// Instances of DtUnknownInteraction are used to represent
// Interactions for which there is no derived DtInteraction.

// Unknown interaction callback function signature.
typedef void (*DtUnknownInteractionCb)
    (DtUnknownInteraction* inter, void* usr);

class DT_DLL_VL DtUnknownInteraction : public DtInteraction
{
public:

    // default constructor
    DtUnknownInteraction();

    // constructor
    DtUnknownInteraction(RTI::InteractionClassHandle handle,
        const RTI::ParameterHandleValuePairSet& pvList);

    // destructor
    virtual ~DtUnknownInteraction();

    // copy constructor
    DtUnknownInteraction(const DtUnknownInteraction& orig);

    // assignment operator
    DtUnknownInteraction& operator=(const DtUnknownInteraction&
orig);

    // Get the parameter handle value pair set for the interaction.
    virtual const RTI::ParameterHandleValuePairSet& phvps() const;

    // Decode into this object from parameter value list.
    virtual void setFromPhvps(
        const RTI::ParameterHandleValuePairSet& pvList);

    // Sets the interaction's ExerciseConn and class descriptor. If
    // classDesc is NULL, we use the classDesc associated with the

```

```
// handle on which the interaction was created.
// Without being told what ExConn or class descriptor to use,
// many of the HLA-specific functions won't work properly.
virtual void setExConn(DtExerciseConn* exConn, DtInterClassDesc*
classDesc);

// Just prints the parameters
virtual void printData() const;

// Name of the interaction
virtual const char* name() const;

public:

// Create a DtUnknownInteraction.
// Caller is responsible for deletion.
static DtInteraction* create();

// Add (register)/Remove (deregister) function to be called when
// interaction with specified class handle occurs.
static void addCallback(RTI::InteractionClassHandle handle,
DtExerciseConn* conn,
DtUnknownInteractionCb cb, void* usr);
static void removeCallback(RTI::InteractionClassHandle handle,
DtExerciseConn* conn,
DtUnknownInteractionCb cb, void* usr);

protected:

// Decode into this object from parameter value list.
// Non-virtual since meant to be called from ctor.
void internalSetFromPhvps(
const RTI::ParameterHandleValuePairSet& pvList);

protected:

RTI::ParameterHandleValuePairSet* myParams;
RTI::InteractionClassHandle myHandle;
};
#endif
```


hVcDec.h

```

#if DtHLA

#include "interDecoder.h"

// This can be used to declare parameter processor functions
// which are static members of a class.

#define DtDECLARE_VIEW_CONTROL_PARAM_DECODER(paramName) \
    DtDECLARE_PARAM_DECODER(DtViewControlInteraction, paramName)

// class DtViewControlDecoder:
//
// Instances of DtViewControlDecoder are used to decode ViewControl
// Interactions
// from the network.

class DT_DLL_VL DtViewControlDecoder : public DtInteractionDecoder
{
public:

    // constructor
    DtViewControlDecoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtViewControlDecoder();

    // copy constructor
    DtViewControlDecoder(const DtViewControlDecoder& orig);

    // assignment operator
    DtViewControlDecoder& operator=(const DtViewControlDecoder&
orig);

protected:

    // Individual parameter decoders

    DtDECLARE_VIEW_CONTROL_PARAM_DECODER(ViewControlPdu);
};
#endif

```

hVcEnc.h

```
#if DtHLA

#include "interEncoder.h"

#include "chooseDefines.h"

#define DtDECLARE_VIEW_CONTROL_PARAM_ENCODER(paramName) \
    DtDECLARE_PARAM_ENCODER(DtViewControlInteraction, paramName)

#define DtDEFINE_SIMPLE_VIEW_CONTROL_PARAM_ENCODER(paramName, \
    netType, inspector) \
    DtDEFINE_SIMPLE_PARAM_ENCODER(DtViewControlEncoder, DtViewCon- \
    trolInteraction, \
                                paramName, netType, \
                                inspector)

// class DtViewControlEncoder:
//
// DtViewControlEncoder is a DtInteractionEncoder that registers
// parameter encoders for all of the VIEW_CONTROL interaction param-
// eters.

class DT_DLL_VL DtViewControlEncoder : public DtInteractionEncoder
{
public:

    DtViewControlEncoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtViewControlEncoder();

    // copy constructor
    DtViewControlEncoder(const DtViewControlEncoder& orig);

    // assignment operator
    DtViewControlEncoder& operator=(const DtViewControlEncoder&
orig);

protected:

    // Individual parameter encoders

    DtDECLARE_VIEW_CONTROL_PARAM_ENCODER(ViewControlPdu);
};
#endif
```

hVclInter.h

```

#if DtHLA

#include "hIntWEncDec.h"
#include "vcPdu.h"

#include "chooseDefines.h"

// class DtViewControlInteraction:
//
// Instances of DtViewControlInteraction are used to represent
// ViewControl Interactions in the RPR FOM.

// ViewControl interaction callback function signature.
typedef void (*DtViewControlInteractionCb)(DtViewControlInterac-
tion* inter,
void* usr);

class DT_DLL_VL DtViewControlInteraction : public DtInteractionWith-
EncDec
{
public:

    // default constructor
    DtViewControlInteraction();

    // destructor
    virtual ~DtViewControlInteraction();

    // copy constructor
    DtViewControlInteraction(const DtViewControlInteraction& orig);

    // assignment operator
    DtViewControlInteraction& operator=(const DtViewControlInterac-
tion& orig);

    // Print the interaction's data.
    virtual void printData() const;

    // Get/Set a view control PDU
    virtual const DtViewControlPdu* vcPdu() const;
    virtual void setVcPdu(const DtViewControlPdu& pdu);
    virtual void setFromNetPacket(const DtNetPacket* pack);

    // Name of the interaction
    virtual const char* name() const;

public:

```

```
// Create a DtViewControlInteraction.
// Caller is responsible for deletion.
static DtInteraction* create();

// Add (register)/Remove (deregister) function to be called when
// interaction occurs.
static void addCallback(DtExerciseConn* conn,
    DtViewControlInteractionCb cb, void* usr);
static void removeCallback(DtExerciseConn* conn,
    DtViewControlInteractionCb cb, void* usr);

protected:

    DtViewControlPdu* myVcPdu;
};
#endif
```

iffAtcNavBeam.h

```

#if !DtHLA
#if DtDISVERS >= 5

#ifndef DtIffAtcNav aidsBeamData_H_
#define DtIffAtcNav aidsBeamData_H_

#include "MachineTypes.h"
#include "disEnums.h"

#include "packets/iffAtcNavP.h"

// class DtIffAtcNav aidsBeamData:
//
// DtIffAtcNav aidsBeamData is a class which contains the antenna
// data
// for the IFF/ATC/NAVAIDS PDU.

class DT_DLL_VL DtIffAtcNav aidsBeamData
{
public:

    // Constructors
    DtIffAtcNav aidsBeamData();
    DtIffAtcNav aidsBeamData(const DtIffAtcNav aidsBeamData& orig);

    // Assignment operator
    DtIffAtcNav aidsBeamData& operator=(
        const DtIffAtcNav aidsBeamData& orig);
    DtIffAtcNav aidsBeamData& operator=(
        const DtNetIffAtcNav aidsBeamData& orig);

    // Destructor
    virtual ~DtIffAtcNav aidsBeamData();

    // Set/Get the Azimuth Center
    virtual void setAzimuthCenter(DtFloat32 value);
    virtual DtFloat32 azimuthCenter() const;

    // Set/Get the Azimuth Sweep
    virtual void setAzimuthSweep(DtFloat32 value);
    virtual DtFloat32 azimuthSweep() const;

    // Set/Get the Elevation Center
    virtual void setElevationCenter(DtFloat32 value);
    virtual DtFloat32 elevationCenter() const;

```

```
// Set/Get the Elevation Sweep
virtual void setElevationSweep(DtFloat32 value);
virtual DtFloat32 elevationSweep() const;

// Set/Get the Sweep Sync
virtual void setSweepSync(DtFloat32 value);
virtual DtFloat32 sweepSync() const;

// Print the information contained in the record
virtual void print() const;

// Load/Unload data from a packet record
virtual void unloadNetRecord(const DtNetIffAtcNavAidsBeamData&
record);
virtual void loadNetRecord(DtNetIffAtcNavAidsBeamData& record)
const;

protected:

// Data members:
DtFloat32          myAzimuthCenter;
DtFloat32          myAzimuthSweep;
DtFloat32          myElevationCenter;
DtFloat32          myElevationSweep;
DtFloat32          mySweepSync;

};
#endif
#endif
```

iffAtcNavLayr.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"

#include "packets/iffAtcNavP.h"

// class DtIffAtcNavAidsLayerHeader:
//
// DtIffAtcNavAidsLayerHeader is a class which contains the layer
// header
// information for a layer of the IFF/ATC/NAVAIDS PDU. These
// records are
// maintained automatically by the PDU class as layers are created
// and
// deleted from the PDU.

class DT_DLL_VL DtIffAtcNavAidsLayerHeader
{
public:

    // Constructors
    DtIffAtcNavAidsLayerHeader();
    DtIffAtcNavAidsLayerHeader(const DtIffAtcNavAidsLayerHeader&
orig);

    // Assignment operator
    DtIffAtcNavAidsLayerHeader& operator=(
        const DtIffAtcNavAidsLayerHeader& orig);
    DtIffAtcNavAidsLayerHeader& operator=(
        const DtNetIffAtcNavAidsLayerHeader& orig);

    // Destructor
    virtual ~DtIffAtcNavAidsLayerHeader();

    // Set/Get the layer number
    virtual void setLayerNumber(DtU8 num);
    virtual DtU8 layerNumber() const;

    // Set/Get the layer specific information
    virtual void setLayerInfo(DtIffAtcNavAidsLayerSpecificInfo info);
    virtual DtIffAtcNavAidsLayerSpecificInfo layerInfo() const;

    // Set/Get the layer length
    virtual void setLayerLength(DtU16 length);

```

```
virtual DtU16 layerLength() const;

// Print the information contained in the record
virtual void print() const;

// Load/Unload data from a packet record
virtual void unloadNetRecord(const DtNetIffAtcNavAidsLayerHeader&
record);
virtual void loadNetRecord(DtNetIffAtcNavAidsLayerHeader& record)
const;

protected:

// Data members:
DtU8                myLayerNumber;
DtU8                myLayerInfo;
DtU16               myLayerLength;

};
#endif
#endif
```


iffAtcNavOper.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"

#include "packets/iffAtcNavP.h"

// class DtIffAtcNavAidsOperational:
//
// DtIffAtcNavAidsOperational is a class which contains the funda-
// mental
// operational data for the IFF/ATC/NAVAIDS PDU.

class DT_DLL_VL DtIffAtcNavAidsOperational
{
public:

    // Constructors
    DtIffAtcNavAidsOperational();
    DtIffAtcNavAidsOperational(const DtIffAtcNavAidsOperational&
orig);

    // Assignment operator
    DtIffAtcNavAidsOperational& operator=(
        const DtIffAtcNavAidsOperational& orig);
    DtIffAtcNavAidsOperational& operator=(
        const DtNetIffAtcNavAidsOperational& orig);

    // Destructor
    virtual ~DtIffAtcNavAidsOperational();

    // Set/Get the system status
    virtual void setSystemStatus(DtU8 status);
    virtual DtU8 systemStatus() const;

    // Set/Get the system status by bit
    virtual void setSystemStatusBit(DtIffAtcNavAidsStatusBit bit,
        DtBoolean value);
    virtual DtBoolean systemStatusBit(
        DtIffAtcNavAidsStatusBit bit) const;

    // Set/Get alternate parameter 4
    virtual void setAltParameter4(DtIffAtcNavAidsAltParm4 mod);
    virtual DtIffAtcNavAidsAltParm4 altParameter4() const;

```

```
// Set/Get the information layers field (each layer has a corresponding
// bit in this field which indicates the presence of that layer)
virtual void setInformationLayers(DtU8 layers);
virtual DtU8 informationLayers() const;

// Set/Query an information layer field for a particular layer
virtual void setLayer(DtU8 layer, DtBoolean value);
virtual DtBoolean layerPresent(int layer) const;

// Set/Get modifier
virtual void setModifier(DtIffAtcNavAidsModifier mod);
virtual DtIffAtcNavAidsModifier modifier() const;

// The following functions allow you to set/get the fundamental
operational
// parameters. These parameters may be stored in different
formats
// depending on the system type field. These functions do not
check
// to make sure that the appropriate format is being used.

// Set/Get a parameter (unsigned 16-bit values)
virtual void setParameter(int index, DtU16 value);
virtual DtU16 parameter(int index) const;

// Set/Get parameter code elements. These are stored in 3-bit
integers
// which occupy bits 0-14 of the parameter.
virtual void setParameterCode(int index, int code, int value);
virtual int parameterCode(int index, int code) const;

// Set/Get parameter code bit
virtual void setParameterBit(int index,
    DtIffAtcNavAidsParameterBit bit, DtBoolean value);
virtual DtBoolean parameterBit(int index,
    DtIffAtcNavAidsParameterBit bit) const;

// Print the information contained in the record
virtual void print() const;

// Load/Unload data from a packet record
virtual void unloadNetRecord(const DtNetIffAtcNavAidsOperational&
record);
virtual void loadNetRecord(DtNetIffAtcNavAidsOperational& record)
const;

protected:

// Data members:
```

```
DtU8          myStatus;  
DtU8          myAltParameter4;  
DtU8          myLayers;  
DtU8          myModifier;  
DtU16         myParameter[6];  
  
};  
#endif  
#endif
```

iffAtcNavOpr2.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"

#include "packets/iffAtcNavP.h"

// class DtIffAtcNavAidsSecondOperational:
//
// DtIffAtcNavAidsSecondOperational is a class which contains the
// secondary
// operational data for a layer of the IFF/ATC/NAVAIDS PDU.

class DT_DLL_VL DtIffAtcNavAidsSecondOperational
{
public:

    // Constructors
    DtIffAtcNavAidsSecondOperational();
    DtIffAtcNavAidsSecondOperational(
        const DtIffAtcNavAidsSecondOperational& orig);

    // Assignment operators
    DtIffAtcNavAidsSecondOperational& operator=(
        const DtIffAtcNavAidsSecondOperational& orig);
    DtIffAtcNavAidsSecondOperational& operator=(
        const DtNetIffAtcNavAidsSecondOperational& orig);

    // Destructor
    virtual ~DtIffAtcNavAidsSecondOperational();

    // Set/Get parameter 1
    virtual void setParameter1(DtIffAtcNavAidsParm1 value);
    virtual DtIffAtcNavAidsParm1 parameter1() const;

    // Set/Get parameter 2
    virtual void setParameter2(DtIffAtcNavAidsParm2 value);
    virtual DtIffAtcNavAidsParm2 parameter2() const;

    // Get the number of parameter sets
    virtual void setNumParameterSets(DtU16);
    virtual DtU16 numParameterSets() const;

    // Print the information contained in the record
    virtual void print() const;
```

```
// Load/Unload data from a packet record
virtual void unloadNetRecord(
    const DtNetIffAtcNavAidsSecondOperational& record);
virtual void loadNetRecord(
    DtNetIffAtcNavAidsSecondOperational& record) const;

protected:

    // Data members:
    DtU8          myParameter1;
    DtU8          myParameter2;
    DtU16         myNumParameterSets;

};
#endif
#endif
```

iffAtcNavParm.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"

#include "packets/iffAtcNavP.h"

// class DtIffAtcNavAidsParameter:
//
// DtIffAtcNavAidsParameter is a class which contains the parameter
// data
// for the IFF/ATC/NAVAIDS PDU.

class DT_DLL_VL DtIffAtcNavAidsParameter
{
public:

    // Constructors
    DtIffAtcNavAidsParameter();
    DtIffAtcNavAidsParameter(const DtIffAtcNavAidsParameter& orig);

    // Assignment operator
    DtIffAtcNavAidsParameter& operator=(
        const DtIffAtcNavAidsParameter& orig);
    DtIffAtcNavAidsParameter& operator=(
        const DtNetIffAtcNavAidsParameter& orig);

    // Destructor
    virtual ~DtIffAtcNavAidsParameter();

    // Set/Get the ERP
    virtual void setERP(DtFloat32 value);
    virtual DtFloat32 ERP() const;

    // Set/Get the frequency
    virtual void setFrequency(DtFloat32 value);
    virtual DtFloat32 frequency() const;

    // Set/Get the PgRF
    virtual void setPgRF(DtFloat32 value);
    virtual DtFloat32 PgRF() const;

    // Set/Get the pulse width
    virtual void setPulseWidth(DtFloat32 value);
    virtual DtFloat32 pulseWidth() const;
```

```
// Set/Get the burst length
virtual void setBurstLength(DtFloat32 value);
virtual DtFloat32 burstLength() const;

// Set/Get the applicable modes
virtual void setApplicableModes(DtIffAtcNavAidsApplicableModes
value);
virtual DtIffAtcNavAidsApplicableModes applicableModes() const;

// Print the information contained in the record
virtual void print() const;

// Load/Unload data from a packet record
virtual void unloadNetRecord(const DtNetIffAtcNavAidsParameter&
record);
virtual void loadNetRecord(DtNetIffAtcNavAidsParameter& record)
const;

protected:

    // Data members:
    DtFloat32          myERP;
    DtFloat32          myFrequency;
    DtFloat32          myPgRF;
    DtFloat32          myPulseWidth;
    DtFloat32          myBurstLength;
    DtU8               myApplicableModes;

};
#endif
#endif
```

iffAtcNavPdu.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "hostStructs.h"
#include "pdu.h"
#include "exerciseConn.h"
#include "mathTypes.h"

#include "packets/iffAtcNavP.h"

#include "iffAtcNavBeam.h"
#include "iffAtcNavLayr.h"
#include "iffAtcNavOper.h"
#include "iffAtcNavOpr2.h"
#include "iffAtcNavParm.h"
#include "iffAtcNavSyID.h"

// class DtIffAtcNavAidsPdu:
//
// Instances of DtIffAtcNavAidsPdu are used to send information
// about simulated
// radio transponders and interrogators.

// DtIffAtcNavAidsPdu callback function signature.
typedef void (*DtIffAtcNavAidsPduCb)(DtIffAtcNavAidsPdu* pdu, void*
usr);

class DT_DLL_VL DtIffAtcNavAidsPdu : public DtPdu
{
public:

    // Constructors
    DtIffAtcNavAidsPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtIffAtcNavAidsPdu(const DtNetIffAtcNavAidsPduLayer1* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtIffAtcNavAidsPdu(const DtIffAtcNavAidsPdu& orig);

    // destructor
    virtual ~DtIffAtcNavAidsPdu();

    // Create a DtIffAtcNavAidsPdu.
    // Caller is responsible for deletion.
    static DtPdu* create(const DtNetPacket* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
```

```

// Add (register)/Remove (deregister) function to be called when
// pdu is received.
static void addCallback(DtExerciseConn* conn,
    DtIffAtcNavAidsPduCb cb, void* usr);
static void removeCallback(DtExerciseConn* conn,
    DtIffAtcNavAidsPduCb cb, void* usr);

// Assignment operator
DtIffAtcNavAidsPdu& operator=(const DtIffAtcNavAidsPdu& orig);

// Set/Get emitting entity identifier
virtual void setEmittingEntityID(const DtEntityIdentifier&
entity);
virtual const DtEntityIdentifier& emittingEntityID() const;

// Set/Get event identifier
virtual void setEventID(const DtEventID& event);
virtual const DtEventID& eventID() const;

// Set/Get location
virtual void setLocation(DtConstVector coords);
virtual DtConstVector location() const;

// Set/Get system ID
virtual void setSystemID(const DtIffAtcNavAidsSystemID& id);
virtual const DtIffAtcNavAidsSystemID& systemID() const;

// Set/Get fundamental operational data
virtual void setOperational(const DtIffAtcNavAidsOperational&
oper);
virtual const DtIffAtcNavAidsOperational& operational() const;

// Query the layers field of the fundamental operational data
record.
virtual DtBoolean layerPresent(int layer) const;

// Enable/Disable layer 2. This method is also used to select the
// number of fundamental parameter data records for the second
layer.
virtual DtBoolean enableLayer2(DtU8 layerSpecificInfo, int
numRecords);
virtual DtBoolean disableLayer2();

// Note: the following accessor functions are for records in
layer 2.
// If layer 2 is not enabled, the sets will do nothing and the
gets will
// return undefined results.

// Get layer header for layer 2

```

```
// This record is set automatically by the PDU class
virtual const DtIffAtcNavAidsLayerHeader& layer2Header() const;

// Set/Get beam data
virtual void setBeamData(const DtIffAtcNavAidsBeamData& beam-
Data);
virtual const DtIffAtcNavAidsBeamData& beamData() const;

// Set/Get secondary operational parameters
virtual void setSecondaryOperational(
    const DtIffAtcNavAidsSecondOperational& oper);
virtual const DtIffAtcNavAidsSecondOperational&
    secondaryOperational() const;

// Get the number of fundamental parameter data records from the
secondary
// operational parameters record
virtual DtU16 numParameterSets() const;

// Set/Get the fundamental parameter data records for the second
layer.
virtual DtBoolean setParmData(int parmIndex,
    const DtIffAtcNavAidsParameter& init);
virtual DtBoolean getParmData(int parmIndex,
    DtIffAtcNavAidsParameter& record) const;

// Print the data contained by this PDU
virtual void printData() const;

protected:

// Returns the internal PDU kind
virtual DtPduKind internalGetPduKind() const;

// Casts the netPdu element to our network struct
virtual DtNetIffAtcNavAidsPduLayer1* netPduStruct() const;

// Casts the netPdu element to our network struct
virtual DtNetIffAtcNavAidsPduLayer2* netPduStructL2() const;

// Casts the 'this' pointer for changing elements in a const
function
// Non virtual since you can't override based on return type
DtIffAtcNavAidsPdu* self() const;

// Returns a pointer to the first parameter record (after static
data).
virtual DtNetIffAtcNavAidsParameter* parameter() const;

protected:
```

```
// Size of PDU with zero parameter records.
static int theStaticSize;

protected:

    // Temporary objects for reference returns
    DtEntityIdentifier          myEntityID;
    DtEventID                  myEventID;
    DtVector                   myEntCoords;
    DtIffAtcNavAidsSystemID    mySystemID;
    DtIffAtcNavAidsOperational myOperational;
    DtIffAtcNavAidsLayerHeader myLayerHeader;
    DtIffAtcNavAidsBeamData    myBeamData;
    DtIffAtcNavAidsSecondOperational mySecondOperational;
    DtIffAtcNavAidsParameter   myParameter;

};
#endif
#endif
```

iffAtcNavSylID.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"

#include "packets/iffAtcNavP.h"

// class DtIffAtcNavAidsSystemID:
//
// DtIffAtcNavAidsSystemID is a class which contains information
// identifying
// the emitting system in the IFF/ATC/NAVAIDS PDU.

class DT_DLL_VL DtIffAtcNavAidsSystemID
{
public:

    // Constructors
    DtIffAtcNavAidsSystemID();
    DtIffAtcNavAidsSystemID(const DtIffAtcNavAidsSystemID& orig);

    // Assignment operator
    DtIffAtcNavAidsSystemID& operator=(const DtIffAtcNavAidsSystemID&
orig);
    DtIffAtcNavAidsSystemID& operator=(const DtNetIffAtcNavAidsSys-
temID& orig);

    // Destructor
    virtual ~DtIffAtcNavAidsSystemID();

    // Set/Get the system type
    virtual void setSystemType(DtIffAtcNavAidsSysType type);
    virtual DtIffAtcNavAidsSysType systemType() const;

    // Set/Get the system name
    virtual void setSystemName(DtIffAtcNavAidsSysName name);
    virtual DtIffAtcNavAidsSysName systemName() const;

    // Set/Get the system mode
    virtual void setSystemMode(DtIffAtcNavAidsSysMode mode);
    virtual DtIffAtcNavAidsSysMode systemMode() const;

    // Set/Get the system change/options
    virtual void setChangeOptions(DtU8 opts);
    virtual DtU8 changeOptions() const;
```

```
// Set/Get the system change/option by bit value
virtual void setChangeOptionsBit(DtIffAtcNavAidsChangeOptionsBit
bit,
    DtBoolean value);
virtual DtBoolean changeOptionsBit(
    DtIffAtcNavAidsChangeOptionsBit bit) const;

// Print the information contained in the record
virtual void print() const;

// Load/Unload data from a packet record
virtual void unloadNetRecord(const DtNetIffAtcNavAidsSystemID&
record);
virtual void loadNetRecord(DtNetIffAtcNavAidsSystemID& record)
const;

protected:

    // Data members:
    DtIffAtcNavAidsSysType      myType;
    DtIffAtcNavAidsSysName     myName;
    DtIffAtcNavAidsSysMode     myMode;
    DtU8                       myChangeOptions;

};
#endif
#endif
```

intClassDesc.h

```
#if DtHLA

#include "RTI.hh"
#include "MachineTypes.h"

#include "chooseDefines.h"

// class DtInterClassDesc:
//
// Instances of DtInterClassDesc are used to represent
// HLA Object and Interaction classes

class DT_DLL_VL DtInterClassDesc
{
public:

    // constructor
    DtInterClassDesc(const char* className,
        DtInterClassDesc* base, RTI::InteractionClassHandle handle,
        DtExerciseConn* exConn);

    // destructor
    virtual ~DtInterClassDesc();

    // copy constructor
    DtInterClassDesc(const DtInterClassDesc& orig);

    // assignment operator
    DtInterClassDesc& operator=(const DtInterClassDesc& orig);

    // Add a parameter (Used when reading FED file).
    virtual void addParam(const char* paramName);

    // Get the object's class handle.
    virtual RTI::InteractionClassHandle handle() const;

    // Get the object's class name
    virtual char* name() const;

    // Return a pointer to the base class's descriptor, NULL if none
    virtual const DtInterClassDesc* baseClass() const;

    // Returns 1 if the class is of class hand or a subclass
    // of hand, 0 otherwise
    virtual int isOfClass(RTI::InteractionClassHandle hand) const;

    // Get the number of parameters associated with this object.
```

```
virtual int numParameters() const;

// Get the set of valid parameter handles for this class.
Includes
// parameters inherited from base classes.
virtual const RTI::AttributeHandleSet& paramHandleSet() const;

// Get the set of non-inherited parameter handles for this class.
virtual const RTI::AttributeHandleSet& ownParamHandles() const;

// Publish the class, and register start and stopUpdates call-
backs.
virtual void publish();

// Subscribe to the class
virtual void subscribe();

// Unsubscribe to the class
virtual void unsubscribe();

// Returns DtTrue if the class is published, DtFalse otherwise.
virtual DtBoolean published() const;

// Returns DtTrue if the class is subscribed, DtFalse otherwise.
virtual DtBoolean subscribed() const;

// Get my RTI ambassador
virtual DtVrIrtiAmbassador* rtiAmb();

// Returns DtTrue if the interaction class is needed by the
federation
// false otherwise.
virtual DtBoolean neededByFederation() const;

// Return true if we have ever published the class using
// DtInterClassDesc::publish, false otherwise.
virtual DtBoolean everPublished() const;

// Return true if we have ever subscribed to the class using
// DtInterClassDesc::subscribe, false otherwise.
virtual DtBoolean everSubscribed() const;

public:

// Set/Get maximum number of allowable parameters for an interac-
tion
static void setMaxParameters(int num);
static int maxParameters();

protected:
```

```
// Callback for handling startInters RTI callback
static void startIntersCb(RTI::InteractionClassHandle hand,
    void* usr);

// Callback for handling stopInters RTI callback
static void stopIntersCb(RTI::InteractionClassHandle hand,
    void* usr);

// Callback for handling startInters RTI callback
virtual void processStartInters(RTI::InteractionClassHandle
hand);

// Callback for handling stopInters RTI callback
virtual void processStopInters(RTI::InteractionClassHandle hand);

protected:

    char* myName;
    RTI::InteractionClassHandle myHandle;
    RTI::AttributeHandleSet* myParams;
    RTI::AttributeHandleSet* myOwnParams;
    DtInterClassDesc* myBase;
    DtExerciseConn* myExConn;
    DtBoolean myNeededFlag;
    DtBoolean myPublished;
    DtBoolean mySubscribed;
    DtBoolean myEverPublished;
    DtBoolean myEverSubscribed;

protected:

    static int theMaxParams;
};
#endif
```


intDecFact.h

```

#if DtHLA

#include "interDecoder.h"
#include "MachineTypes.h"

// class DtInteractionDecoderFactory:
//
// Instances of DtInteractionDecoderFactory are used to create
// decoders which
// know how to decode an HLA interaction message (or network repre-
// sentation)
// into an interaction repository (or host representation).

class DT_DLL_VL DtInteractionDecoderFactory
{
public:

    // constructor
    DtInteractionDecoderFactory(DtExerciseConn* exConn);

    // destructor
    virtual ~DtInteractionDecoderFactory();

    // copy constructor
    // Copies across added decoder creation functions.
    DtInteractionDecoderFactory(const DtInteractionDecoderFactory&
orig);

    // assignment operator
    // Copies across added decoder creation functions.
    DtInteractionDecoderFactory& operator=(
        const DtInteractionDecoderFactory& orig);

    // Create a decoder for the specified interaction class.
    // Note: Caller is responsible for deleting.
    // Returns NULL if decoder creation function is not in the
factory.
    virtual DtInteractionDecoder* createDecoder(
        RTI::InteractionClassHandle hand);

    // Add a decoder prototype to the factory. The decoder prototype
    // will be cloned by createDecoder when called for the appro-
priate
    // handle. Caller deletes the decoder, but AFTER the decoder
    // factory no longer needs it. Returns any previous decoder
which
    // was replaced by the add.

```

```
virtual DtInteractionDecoder* addDecoder(
    RTI::InteractionClassHandle hand,
    DtInteractionDecoder* decoder);

// This function is used to add an parameter decoder function
// to several prototype decoders all at once - namely, those
// associated with the interaction class denoted by hand, as well
// as all of its subclasses. If applyToSubclasses is DtFalse,
// then the function is only registered with the one decoder
// prototype, making it equivalent to
// decoder(hand)->addDecoder(hand, decoder));
virtual void addParameterDecoder(
    RTI::InteractionClassHandle classHand,
    RTI::ParameterHandle paramHand,
    DtInteractionDecoder::DtParameterDecoder decoder,
    DtBoolean applyToSubclasses = DtTrue);

// Same as above, but identify class by name
void addParameterDecoder(
    char* className,
    RTI::ParameterHandle paramHand,
    DtInteractionDecoder::DtParameterDecoder func,
    DtBoolean applyToSubclasses = DtTrue);

// Same as above, but identify param by name
void addParameterDecoder(
    RTI::InteractionClassHandle classHand,
    char* paramName,
    DtInteractionDecoder::DtParameterDecoder func,
    DtBoolean applyToSubclasses = DtTrue);

// Same as above, but identify class and param by name
void addParameterDecoder(
    char* className,
    char* paramName,
    DtInteractionDecoder::DtParameterDecoder func,
    DtBoolean applyToSubclasses = DtTrue);

// Returns the decoder prototype associated with handle by a
previous
// call to addDecoder.
virtual DtInteractionDecoder* decoder(RTI::InteractionClassHandle
hand);

protected:

    // myProtos is an array of DtInteraction::maxInteractionClasses()
items.
    DtInteractionDecoder** myProtos;
    DtExerciseConn* myExConn;
```

```
};  
#endif
```

intEncFact.h

```
#if DtHLA

#include "interEncoder.h"
#include "MachineTypes.h"

// class DtInteractionEncoderFactory:
//
// Instances of DtInteractionEncoderFactory are used to create
// encoders
// which know how to encode an interaction repository (or host
// representation) into an HLA interaction message (or network
// representation).

class DT_DLL_VL DtInteractionEncoderFactory
{
public:

    // constructor
    DtInteractionEncoderFactory(DtExerciseConn* exConn);

    // destructor
    virtual ~DtInteractionEncoderFactory();

    // copy constructor
    DtInteractionEncoderFactory(const DtInteractionEncoderFactory&
orig);

    // assignment operator
    DtInteractionEncoderFactory& operator=(
        const DtInteractionEncoderFactory& orig);

    // Create an encoder for the specified interaction class.
    // Note: Caller is responsible for deleting.
    // Returns NULL if encoder creation function is not in the
factory.
    virtual DtInteractionEncoder* createEncoder(
        RTI::InteractionClassHandle hand);

    // Add a encoder prototype to the factory. The encoder prototype
    // will be cloned by createEncoder when called for the appro-
priate
    // handle. Caller deletes the encoder, but AFTER the encoder
    // factory no longer needs it. Returns any previous encoder
which was
    // replaced by the add.
    virtual DtInteractionEncoder* addEncoder(
        RTI::InteractionClassHandle hand,
```

```

    DtInteractionEncoder* encoder);

// This function is used to add an parameter encoder function
// to several prototype encoders all at once - namely, those
// associated with the interaction class denoted by hand, as well
// as all of its subclasses. If applyToSubclasses is DtFalse,
// then the function is only registered with the one encoder
// prototype, making it equivalent to
// encoder(hand)->addEncoder(hand, encoder));
virtual void addParameterEncoder(
    RTI::InteractionClassHandle classHand,
    RTI::ParameterHandle paramHand,
    DtInteractionEncoder::DtParameterEncoder encoder,
    DtBoolean applyToSubclasses = DtTrue);

// Same as above, but identify class by name
void addParameterEncoder(
    char* className,
    RTI::ParameterHandle paramHand,
    DtInteractionEncoder::DtParameterEncoder func,
    DtBoolean applyToSubclasses = DtTrue);

// Same as above, but identify param by name
void addParameterEncoder(
    RTI::InteractionClassHandle classHand,
    char* paramName,
    DtInteractionEncoder::DtParameterEncoder func,
    DtBoolean applyToSubclasses = DtTrue);

// Same as above, but identify class and param by name
void addParameterEncoder(
    char* className,
    char* paramName,
    DtInteractionEncoder::DtParameterEncoder func,
    DtBoolean applyToSubclasses = DtTrue);

// Returns the encoder prototype associated with handle by a
previous
// call to addEncoder.
virtual DtInteractionEncoder* encoder(RTI::InteractionClassHandle
hand);

protected:

    // myProtos is an array of DtInteraction::maxInteractionClasses()
items.
    DtInteractionEncoder** myProtos;
    DtExerciseConn* myExConn;
};
#endif

```

intercmCtlPdu.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "hostStructs.h"
#include "pdu.h"
#include "exerciseConn.h"

#include "packets/intercmCtlP.h"

#include "intercmParRec.h"
#include "intercmEntDst.h"
#include "intercmGrpAsg.h"
#include "intercmGrpDst.h"

// class DtIntercomControlPdu:
//
// Instances of DtIntercomControlPdu are used to communicate state
// information for simulated intercom devices.

// DtIntercomControlPdu callback function signature.
typedef void (*DtIntercomControlPduCb)(DtIntercomControlPdu* pdu,
void* usr);

class DT_DLL_VL DtIntercomControlPdu : public DtPdu
{
public:

    // Constructors
    DtIntercomControlPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtIntercomControlPdu(const DtNetIntercomControlPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtIntercomControlPdu(const DtIntercomControlPdu& orig);

    // destructor
    virtual ~DtIntercomControlPdu();

    // Assignment operator
    DtIntercomControlPdu& operator=(const DtIntercomControlPdu&
orig);

    // Set/Get intercom control type
    virtual void setControlType(DtIntercomCtlType controlType);
    virtual DtIntercomCtlType controlType() const;

    // Set/Get intercom communications channel class
```

```

virtual void setCommClass(DtIntercomCommClass commClass);
virtual DtIntercomCommClass commClass() const;

// Set/Get intercom communications channel type
virtual void setCommType(DtIntercomCommType commType);
virtual DtIntercomCommType commType() const;

// Set/Get source entity identifier
virtual void setSourceEntityID(const DtEntityIdentifier& entity);
virtual const DtEntityIdentifier& sourceEntityID() const;

// Set/Get source communications device ID
virtual void setSourceDeviceID(DtU16 deviceID);
virtual DtU16 sourceDeviceID() const;

// Set/Get source line ID
virtual void setSourceLineID(DtU8 lineID);
virtual DtU8 sourceLineID() const;

// Set/Get transmit priority
virtual void setTransmitPriority(DtU8 transmitPriority);
virtual DtU8 transmitPriority() const;

// Set/Get transmit line state
virtual void setTransmitLineState(DtIntercomTransLineState
lineState);
virtual DtIntercomTransLineState transmitLineState() const;

// Set/Get intercom command type
virtual void setCmdType(DtIntercomCmdType cmdType);
virtual DtIntercomCmdType cmdType() const;

// Set/Get master entity identifier
virtual void setMasterEntityID(const DtEntityIdentifier& entity);
virtual const DtEntityIdentifier& masterEntityID() const;

// Set/Get master communications device ID
virtual void setMasterDeviceID(DtU16 deviceID);
virtual DtU16 masterDeviceID() const;

// Set/Get master channel ID
virtual void setMasterChannelID(DtU16 channelID);
virtual DtU16 masterChannelID() const;

// Get the parameters length. This field is automatically
changed
// by the functions: setNumParmRecords(), and setParmRecord()
virtual DtU32 parmRecordsLength() const;

// Set/Get number of parameter records (this is not a PDU field).
```

```
// If the required space would exceed the maximum space permitted
for
// a PDU, then DtFalse may be returned.
virtual DtBoolean setNumParmRecords(int numRecords);
virtual int numParmRecords() const;

// Get the record type for parameter record parmIndex. If
parmIndex is
// out of range or the parameter is invalid, then DtFalse is
returned.
virtual DtBoolean getRecordType(int parmIndex,
DtIntercomCtlRecType& recordType) const;

// Set/Get the parameter record type and values of record
parmIndex to
// values specified by the initializer in the second argument.
If parmIndex
// is out of bounds, or if the record type does not match the
function
// (for a get), then DtFalse is returned. If the space would
exceed the
// maximum space permitted for a PDU (for a set) then DtFalse may
be
// returned.
virtual DtBoolean setParmRecord(int parmIndex,
const DtIntercomEntityDestRec& init);
virtual DtBoolean getParmRecord(int parmIndex,
DtIntercomEntityDestRec& record) const;
virtual DtBoolean setParmRecord(int parmIndex,
const DtIntercomGroupDestRec& init);
virtual DtBoolean getParmRecord(int parmIndex,
DtIntercomGroupDestRec& record) const;
virtual DtBoolean setParmRecord(int parmIndex,
const DtIntercomGroupAsgmtRec& init);
virtual DtBoolean getParmRecord(int parmIndex,
DtIntercomGroupAsgmtRec& record) const;

// Print the data contained by this PDU
virtual void printData() const;

public:

// Create a DtIntercomControlPdu.
// Caller is responsible for deletion.
static DtPdu* create(const DtNetPacket* initial,
DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

// Add (register)/Remove (deregister) function to be called when
// pdu is received.
static void addCallback(DtExerciseConn* conn,
```



```
DtIntercomControlPduCb cb, void* usr);
static void removeCallback(DtExerciseConn* conn,
    DtIntercomControlPduCb cb, void* usr);
```

protected:

```
// Returns the internal PDU kind
virtual DtPduKind internalGetPduKind() const;

// Casts the netPdu element to our network struct
virtual DtNetIntercomControlPdu* netPduStruct() const;

// Casts the 'this' pointer for changing elements in a const
function
// Non-virtual since you can't override based on return type
DtIntercomControlPdu* self() const;

// Returns a pointer to the first byte of variable length data
virtual char* var() const;

// Returns the offset (in bytes) of the given parameter record
virtual int recordOffset(int recordIndex) const;

// Resize a given parameter record to the size indicated
virtual void resizeRecord(DtNetIntercomParmRec* rec, int
newSize);
```

protected:

```
// Size of PDU with zero parameter records.
static int theStaticSize;
```

protected:

```
// Number of parameter records in this PDU
int myNumParmRecords;

// Temporary object for reference returns
DtEntityIdentifier myEntityID;
```

```
};
#endif
#endif
```

intercmEntDst.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"
#include "hostStructs.h"

#include "packets/intercmCtlP.h"
#include "intercmParRec.h"

// class DtIntercomEntityDestRec:
//
// Instances of DtIntercomEntityDestRec are used to contain the
// information
// in the entity destination parameter records used by the intercom
// control
// PDU class.

class DT_DLL_VL DtIntercomEntityDestRec : public DtIntercomParmRec
{
public:

    // Constructors
    DtIntercomEntityDestRec();
    DtIntercomEntityDestRec(const DtIntercomEntityDestRec& orig);

    // Destructor
    virtual ~DtIntercomEntityDestRec();

    // Assignment operators
    DtIntercomEntityDestRec& operator=(
        const DtIntercomEntityDestRec& orig);
    DtIntercomEntityDestRec& operator=(
        const DtNetIntercomEntityDestRec& orig);

    // Set/Get the destination entity identifier
    virtual void setDestEntityID(const DtEntityIdentifier& entityID);
    virtual const DtEntityIdentifier& destEntityID() const;

    // Set/Get the destination device ID
    virtual void setDestDeviceID(DtU16 deviceID);
    virtual DtU16 destDeviceID() const;

    // Set/Get the destination line ID
    virtual void setDestLineID(DtU8 lineID);
    virtual DtU8 destLineID() const;
```

```
// Set/Get the destination priority
virtual void setDestPriority(DtU8 priority);
virtual DtU8 destPriority() const;

// Set/Get the destination line state command
virtual void setDestLineStateCmd(DtIntercomDestLineStateCmd cmd);
virtual DtIntercomDestLineStateCmd destLineStateCmd() const;

// Print this record
virtual void print() const;

// Load/Unload data from a packet record
virtual void unloadNetRecord(const DtNetIntercomEntityDestRec&
record);
virtual void loadNetRecord(DtNetIntercomEntityDestRec& record)
const;

protected:

    DtEntityIdentifier      myDestEntityID;
    DtU16                   myDestDeviceID;
    DtU8                    myDestLineID;
    DtU8                    myDestPriority;
    DtIntercomDestLineStateCmd myDestLineStateCmd;

};
#endif
#endif
```

intercmGrpAsg.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"
#include "hostStructs.h"

#include "packets/intercmCtlP.h"
#include "intercmParRec.h"

// class DtIntercomGroupAsgmtRec:
//
// Instances of DtIntercomGroupAsgmtRec are used to contain the
// information
// in the group assignment parameter record used by the intercom
// control
// PDU class.

class DT_DLL_VL DtIntercomGroupAsgmtRec : public DtIntercomParmRec
{
public:

    // Constructors
    DtIntercomGroupAsgmtRec();
    DtIntercomGroupAsgmtRec(const DtIntercomGroupAsgmtRec& orig);

    // Destructor
    virtual ~DtIntercomGroupAsgmtRec();

    // Assignment operators
    DtIntercomGroupAsgmtRec& operator=(
        const DtIntercomGroupAsgmtRec& orig);
    DtIntercomGroupAsgmtRec& operator=(
        const DtNetIntercomGroupAsgmtRec& orig);

    // Set/Get the group bit field
    virtual void setGroupBitField(DtU32 bitField);
    virtual DtU32 groupBitField() const;

    // Set/Get the destination entity identifier
    virtual void setDestEntityID(const DtEntityIdentifier& entityID);
    virtual const DtEntityIdentifier& destEntityID() const;

    // Set/Get the destination equipment ID
    virtual void setDestEquipID(DtU16 equipID);
    virtual DtU16 destEquipID() const;
```

```
// Set/Get the destination line ID
virtual void setDestLineID(DtU8 lineID);
virtual DtU8 destLineID() const;

// Print this record
virtual void print() const;

// Load/Unload data from a packet record
virtual void unloadNetRecord(const DtNetIntercomGroupAsgmtRec&
record);
virtual void loadNetRecord(DtNetIntercomGroupAsgmtRec& record)
const;

protected:

    DtU32                myGroupBitField;
    DtEntityIdentifier    myDestEntityID;
    DtU16                myDestEquipID;
    DtU8                 myDestLineID;

};
#endif
#endif
```

intercmGrpDst.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"
#include "hostStructs.h"

#include "packets/intercmCtlP.h"
#include "intercmParRec.h"

// class DtIntercomGroupDestRec:
//
// Instances of DtIntercomGroupDestRec are used to contain the
// information
// in the group destination parameter record used by the intercom
// control
// PDU class.

class DT_DLL_VL DtIntercomGroupDestRec : public DtIntercomParmRec
{
public:

    // Constructors
    DtIntercomGroupDestRec();
    DtIntercomGroupDestRec(const DtIntercomGroupDestRec& orig);

    // Destructor
    virtual ~DtIntercomGroupDestRec();

    // Assignment operators
    DtIntercomGroupDestRec& operator=(
        const DtIntercomGroupDestRec& orig);
    DtIntercomGroupDestRec& operator=(
        const DtNetIntercomGroupDestRec& orig);

    // Set/Get the group bit field
    virtual void setDestGroupBitField(DtU32 bitField);
    virtual DtU32 destGroupBitField() const;

    // Set/Get the destination priority
    virtual void setDestPriority(DtU8 priority);
    virtual DtU8 destPriority() const;

    // Set/Get the destination line state command
    virtual void setDestLineStateCmd(DtIntercomDestLineStateCmd cmd);
    virtual DtIntercomDestLineStateCmd destLineStateCmd() const;
```

```
// Print this record
virtual void print() const;

// Load/Unload data from a packet record
virtual void unloadNetRecord(const DtNetIntercomGroupDestRec&
record);
virtual void loadNetRecord(DtNetIntercomGroupDestRec& record)
const;

protected:

    DtU32                myDestGroupBitField;
    DtU8                 myDestPriority;
    DtIntercomDestLineStateCmd myDestLineStateCmd;

};
#endif
#endif
```

intercmParRec.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"

// class DtIntercomParmRec:
//
// DtIntercomParmRec is an abstract base class for the parameter
// record
// classes which are used by the intercom control PDU (DtIntercom-
// ControlPdu).

class DT_DLL_VL DtIntercomParmRec
{
public:
    // Destructor
    virtual ~DtIntercomParmRec();

    // Get the record type
    virtual DtIntercomCtlRecType recordType() const;

    // Get the record length
    virtual DtU16 recordLength() const;

    // Print the record type and length
    virtual void print() const;

protected:
    // Constructors
    DtIntercomParmRec();
    DtIntercomParmRec(const DtIntercomParmRec& orig);
    DtIntercomParmRec(DtIntercomCtlRecType recordType,
        int recordLength);

    // Assignment operator
    DtIntercomParmRec& operator=(const DtIntercomParmRec& orig);

protected:
    // Data members:
    DtIntercomCtlRecType    myRecordType;
    DtU16                  myRecordLength;
```



```
};  
#endif  
#endif
```

intercmSigPdu.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "hostStructs.h"
#include "pdu.h"
#include "exerciseConn.h"

#include "packets/intercmSigP.h"

// class DtIntercomSignalPdu:
//
// Instances of DtIntercomSignalPdu are used to send simulated
// intercom
// signal data.

// DtIntercomSignalPdu callback function signature.
typedef void (*DtIntercomSignalPduCb)(DtIntercomSignalPdu* pdu,
void* usr);

class DT_DLL_VL DtIntercomSignalPdu : public DtPdu
{
public:

    // Constructors
    DtIntercomSignalPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtIntercomSignalPdu(const DtNetIntercomSignalPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtIntercomSignalPdu(const DtIntercomSignalPdu& orig);

    // destructor
    virtual ~DtIntercomSignalPdu();

    // Assignment operator
    DtIntercomSignalPdu& operator=(const DtIntercomSignalPdu& orig);

    // Set/get transmitting entity identifier
    virtual void setTransEntityID(const DtEntityIdentifier& entity);
    virtual const DtEntityIdentifier& transEntityID() const;

    // Set/get intercom ID
    virtual void setDeviceID(DtU16 intercomID);
    virtual DtU16 deviceID() const;

    // Set/get intercom encoding class
    virtual void setEncodingClass(DtEncodingClass encClass);
```

```

virtual DtEncodingClass encodingClass() const;

// Set/get intercom encoding type
virtual void setEncodingType(DtU16 encType);
virtual DtU16 encodingType() const;

// Set/get intercom TDL type
virtual void setTDL_Type(DtU16 TDL_Type);
virtual DtU16 TDL_Type() const;

// Set/get sample rate
virtual void setSampleRate(DtU32 rate);
virtual DtU32 sampleRate() const;

// Set/get data length in bits
virtual void setNumDataBits(DtU16 lengthInBits);
virtual DtU16 numDataBits() const;

// Set/get data length in bytes
virtual void setNumDataBytes(DtU16 lengthInBytes);
virtual DtU16 numDataBytes() const;

// Set/get number of samples
virtual void setNumSamples(DtU16 samples);
virtual DtU16 numSamples() const;

// Set/get a data byte
virtual void setDataByte(int byteNum, DtU8 data);
virtual DtU8 dataByte(int byteNum) const;

// Set/get multiple data bytes. Provide a pointer and data is
// copied to and from that memory. Check numDataBytes() to make
sure that
// you provide enough room in the buffer.
virtual void setData(const DtU8* data, int numBytes);
virtual void getData(DtU8* buff) const;

// Print the data contained by this PDU
virtual void printData() const;

public:

// Create a DtIntercomSignalPdu.
// Caller is responsible for deletion.
static DtPdu* create(const DtNetPacket* initial,
    DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

// Add (register)/Remove (deregister) function to be called when
// pdu is received.
static void addCallback(DtExerciseConn* conn,

```

```
DtIntercomSignalPduCb cb, void* usr);  
static void removeCallback(DtExerciseConn* conn,  
    DtIntercomSignalPduCb cb, void* usr);
```

protected:

```
    // Returns the internal PDU kind  
    virtual DtPduKind internalGetPduKind() const;  
  
    // Casts the netPdu element to our network struct  
    virtual DtNetIntercomSignalPdu* netPduStruct() const;  
  
    // Casts the 'this' pointer for changing elements in a const  
function  
    // Non virtual since you can't override based on return type  
    DtIntercomSignalPdu* self() const;  
  
    // Returns a pointer to the first byte of variable length data  
    virtual char* var() const;
```

protected:

```
    // Size of PDU with zero data bytes.  
    static int theStaticSize;
```

protected:

```
    // Temporary object for reference returns  
    DtEntityIdentifier myEntityID;
```

```
};  
#endif  
#endif
```

interDecoder.h

```

#if DtHLA

#include "RTI.hh"
#include "vfprintf.h"

#include "chooseDefines.h"

// DtInteractionRep is not a real class. It is a placeholder
// denoting
// any class that is a container for an interaction's values. For
// example,
// our derived DtInteraction classes currently are DtInteraction-
// Reps.
// class DtInteractionDecoder:
//
// Instances of DtInteractionDecoder are used to decode interactions
// from the network.

class DT_DLL_VL DtInteractionDecoder
{
public:

    // Parameter decoder function signature.
    typedef void (*DtParameterDecoder)(DtInteractionRep* inter,
        const RTI::ParameterHandleValuePairSet& pvlist,
        int i);

public:

    // constructor
    DtInteractionDecoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtInteractionDecoder();

    // copy constructor
    DtInteractionDecoder(const DtInteractionDecoder& orig);

    // assignment operator
    DtInteractionDecoder& operator=(const DtInteractionDecoder&
orig);

    // Add/Remove a decoding function for the parameter specified
    // by handle.
    // Returns the decoding function which is replaced by the
    add/remove.

```

```
// Note: This is non-virtual since it is typically called by
//       constructors.
DtParameterDecoder addDecoder(RTI::ParameterHandle handle,
    DtParameterDecoder proc);

// Add/Remove a decoding function for the parameter specified
// by name.
// Returns the decoding function which is replaced by the
add/remove.
// Note: This is non-virtual since it is typically called by
//       constructors.
DtParameterDecoder addDecoder(char* name,
    DtParameterDecoder proc);

// Get the parameter decoder for the specified handle.
virtual DtParameterDecoder decoder(int handle) const;

// Decode the interaction.
virtual void decode(DtInteractionRep* interaction,
    const RTI::ParameterHandleValuePairSet& pvList);

// Get the interaction's class handle.
virtual RTI::InteractionClassHandle interactionClassHandle();

// Returns a new-ed clone of this object
virtual DtInteractionDecoder* clone() const;

// Set the class descriptor of the decoder
virtual void setClassDesc(DtInterClassDesc* desc);

public:

    // Returns a pointer to a large static buffer that can be used to
    // copy data obtained from a phvps.
    static char* decodingBuffer();

protected:

    // Process (decode) the specified parameter.
    virtual void decodeParameter(DtInteractionRep* inter,
        const RTI::ParameterHandleValuePairSet& pvList, int i);

protected:

    DtExerciseConn* myExConn;
    DtInterClassDesc* myClassDesc;

    DtVrIRtiAmbassador* myRtiAmb;
    RTI::InteractionClassHandle myClassHandle;
```

```

    // myList is an array of theMaxParametersPerInteraction items.
    DtParameterDecoder* myList;

protected:

    static const int theMaxParametersPerInteraction;
    static char theDecBuffer[2000];
};
// This can be used to add a parameter decoder function from a
// routine.
#define DtADD_PARAM_DECODER(attrName) \
    addDecoder(#attrName, (DtParameterDecoder) decode##attrName)

// This can be used to declare attributeDecoder functions
// which are static members of a class.
#define DtDECLARE_PARAM_DECODER(RepT, paramName) \
    static void decode##paramName(RepT* inter, \
        const RTI::ParameterHandleValuePairSet& params, int pairSet-
Index)

#define DtDEFINE_SIMPLE_PARAM_DECODER_WITH_CAST(decClass, inter-
Class, \
    paramName, netType, mutator, castExpr) \
void decClass::decode##paramName(interClass* inter, \
    const RTI::ParameterHandleValuePairSet& params, \
    int index) \
{ \
    RTI::ULong length = 0; \
    netType* netVal = (netType*) params.getValuePointer(index,
length); \
    if (length > sizeof(netType)) \
    { \
        DtWarn("Size of value decoded for parameter %s (%d)\n", \
            #paramName, length); \
        DtWarn(" is larger than size of %s (%d)\n", #netType,
sizeof(netType)); \
    } \
    inter->mutator(castExpr *netVal); \
}

#define DtDEFINE_SIMPLE_PARAM_DECODER(decClass, interClass, \
    paramName, netType, mutator) \
    DtDEFINE_SIMPLE_PARAM_DECODER_WITH_CAST(decClass, interClass, \
        paramName, netType, mutator, (netType))

#endif

```

interEncoder.h

```
#if DtHLA

#include "RTI.hh"
#include "MachineTypes.h"
#include "chooseDefines.h"

// DtInteractionRep is not a real class. It is a placeholder
// denoting
// any class that is a container for an interaction's values. For
// example,
// our derived DtInteraction classes currently are DtInteraction-
// Reps.
// class DtInteractionEncoder:
//
// DtInteractionEncoder is used to provide encoding of an
// interaction into an RTI::ParameterHandleValuePairSet in an HLA
// (as opposed to DIS) environment.

class DT_DLL_VL DtInteractionEncoder
{
public:

    // constructor
    DtInteractionEncoder(DtExerciseConn* exConn,
        DtInterClassDesc* classDesc);

    // destructor
    virtual ~DtInteractionEncoder();

    // copy constructor
    DtInteractionEncoder(const DtInteractionEncoder& orig);

    // assignment operator
    DtInteractionEncoder& operator=(const DtInteractionEncoder&
orig);

    // Attribute encoding function declaration.
    typedef void (*DtParameterEncoder)
        (const DtInteractionRep& inter,
            RTI::ParameterHandleValuePairSet* avlist,
            RTI::ParameterHandle handle);

    // Add a parameter encoder function by parameter handle.
    // Returns any previous parameter encoder function which was
    // replaced by the add.
    virtual DtParameterEncoder addEncoder(RTI::ParameterHandle h,
        DtParameterEncoder func);

}
```



```

// Add a parameter encoder function by parameter name.
// Returns any previous parameter encoder function which was
// replaced by the add.
virtual DtParameterEncoder addEncoder(char* name,
    DtParameterEncoder proc);

// Get the parameter encoder for the specified handle.
virtual DtParameterEncoder encoder(int handle) const;

// Encode necessary values from the protocol-independent interac-
tion.
virtual void encode(const DtInteractionRep& inter,
    RTI::ParameterHandleValuePairSet* pvList);

// Get the object's class handle.
virtual RTI::InteractionClassHandle interClassHandle();

// Returns a new-ed clone of this object
virtual DtInteractionEncoder* clone() const;

// Set the class descriptor of the encoder
virtual void setClassDesc(DtInterClassDesc* desc);

protected:

    DtExerciseConn* myExConn;
    DtInterClassDesc* myClassDesc;

    DtVrIrtiAmbassador* myRtiAmb;
    RTI::InteractionClassHandle myHandle;

    // myList is an array of theMaxParametersPerInteraction items.
    DtParameterEncoder* myList;

protected:

    static const int theMaxParametersPerInteraction;
};
#define DtADD_PARAM_ENCODER(attrName) \
    addEncoder(#attrName, (DtParameterEncoder) encode##attrName)

#define DtDECLARE_PARAM_ENCODER(interClass, paramName) \
    static void encode##paramName(const interClass& inter, \
        RTI::ParameterHandleValuePairSet* params, \
        RTI::ParameterHandle paramHandle)

#define DtDEFINE_SIMPLE_PARAM_ENCODER(encClass, interClass, \
        paramName, netType, \
        inspector) \

```

```
void encClass::encode##paramName(const interClass& inter, \  
    RTI::ParameterHandleValuePairSet* params, \  
    RTI::ParameterHandle paramHandle) \  
{ \  
    netType netVal; \  
    netVal = inter.inspector(); \  
    params->add(paramHandle, (char*) &netVal, sizeof(netType)); \  
}  
  
#endif
```

interMgr.h

```

#if DtHLA

#include "hInterFactory.h"
#include "callbacks.h"

// class DtInteractionManager:
//
// Instances of DtInteractionManager are used to manage the creation
// and processing of HLA interactions.

class DT_DLL_VL DtInteractionManager
{
public:

    // constructor
    DtInteractionManager(DtInteractionFactory* factory);

    // destructor
    virtual ~DtInteractionManager();

    // copy constructor
    // Care needs to be taken since this copies the callbacks.
    // This means the manager will have a pointer to usr data supplied
    // to the other manager which the original owner may not be aware
    // of or anticipate.
    DtInteractionManager(const DtInteractionManager& orig);

    // assignment operator
    // Care needs to be taken since this copies the callbacks.
    // This means the manager will have a pointer to usr data supplied
    // to the other manager which the original owner may not be aware
    // of or anticipate.
    DtInteractionManager& operator=(const DtInteractionManager&
orig);

    // Add/remove a callback to be called when an interaction is
    // received from another federate.
    virtual void addInteractionCallback(RTI::InteractionClassHandle
h,
        DtReceiveInteractionCb cb,
        void* usr);
    virtual void removeInteractionCallback(RTI::InteractionClass-
Handle h,
        DtReceiveInteractionCb cb,
        void* usr);

    // Create an interaction object, call the registered callbacks,

```

```
// then delete the interaction object.  
virtual void processInteraction(RTI::InteractionClassHandle h,  
    const RTI::ParameterHandleValuePairSet& p,  
    const char* tag = NULL);
```

protected:

```
// Copy callback list.  
// Non-virtual since meant to be called from constructor.  
void copyCbList(const DtInteractionManager& orig);  
  
// Empty callback list.  
// Non-virtual since meant to be called from destructor.  
void emptyCbList();
```

protected:

```
    DtInteractionFactory* myFactory;  
    // myCbList is an array of DtInteraction::maxInteractionClasses()  
items.  
    DtList* myCbList;  
};  
#endif
```

intHashKey.h

```
#include "baseHashKey.h"

// class DtIntHashKey:
//
// Instances of DtIntHashKey are used as a hash key made from an
// integer.

class DT_DLL_VLUTIL DtIntHashKey : public DtBaseHashKey
{
public:

    // default constructor
    DtIntHashKey(unsigned long num);

    // destructor
    virtual ~DtIntHashKey();

    // copy constructor
    DtIntHashKey(const DtIntHashKey& orig);

    // assignment operator
    DtIntHashKey& operator=(const DtIntHashKey& orig);

    // make a copy of this hash key
    virtual DtBaseHashKey* clone() const;

protected:

    // get the string representation of this key
    virtual const DtString& stringRep() const;

    // compute the hashed value for this key
    virtual DtU32 computeHashValue();

protected:

    unsigned long myNum;
    DtString myStr;
};
```

isGroupOfPdu.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "hostStructs.h"
#include "pdu.h"
#include "mathTypes.h"
#include "exerciseConn.h"

#include "packets/isGroupOfP.h"

#include "gedBasFixWing.h"
#include "gedBasRotWing.h"
#include "gedBasSoldier.h"
#include "gedBasVehicle.h"
#include "gedEnhFixWing.h"
#include "gedEnhRotWing.h"
#include "gedEnhSoldier.h"
#include "gedEnhVehicle.h"
#include "gedLogVehicle.h"

// class DtIsGroupOfPdu:
//
// Instances of DtIsGroupOfPdu are used to communicate information
// about a
// particular group of entities (grouped together for the purposes
// of network
// bandwidth reduction).

// DtIsGroupOfPdu callback function signature.
typedef void (*DtIsGroupOfPduCb)(DtIsGroupOfPdu* pdu, void* usr);

class DT_DLL_VL DtIsGroupOfPdu : public DtPdu
{
public:

    // Constructors
    DtIsGroupOfPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtIsGroupOfPdu(const DtNetIsGroupOfPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtIsGroupOfPdu(const DtIsGroupOfPdu& orig);

    // destructor
    virtual ~DtIsGroupOfPdu();

    // Assignment operator
```

```
DtIsGroupOfPdu& operator=(const DtIsGroupOfPdu& orig);

// Set/Get group entity ID
virtual void setGroupEntityID(const DtEntityIdentifier& id);
virtual const DtEntityIdentifier& groupEntityID() const;

// Set/Get the grouped entity category (setCategory will also set
the number
// of entities to zero)
virtual void setCategory(DtGroupedEntityCategory value);
virtual DtGroupedEntityCategory category() const;

// Set/Get the number of entities
virtual DtBoolean setNumEntities(DtU8 num);
virtual DtU8 numEntities() const;

// Set/Get the reference latitude
virtual void setLatitude(DtFloat64 value);
virtual DtFloat64 latitude() const;

// Set/Get the reference longitude
virtual void setLongitude(DtFloat64 value);
virtual DtFloat64 longitude() const;

// Set/Get a BasicFixedWingAircraftRecord. Returns DtFalse if
the
// index is out of range, or if the entity category does not
match.
virtual DtBoolean setBasicFixedWingAircraftRecord(int index,
const DtBasicFixedWingAircraftRecord& init);
virtual DtBoolean getBasicFixedWingAircraftRecord(int index,
DtBasicFixedWingAircraftRecord& record) const;

// Set/Get an EnhancedFixedWingAircraftRecord. Returns DtFalse
if the
// index is out of range, or if the entity category does not
match.
virtual DtBoolean setEnhancedFixedWingAircraftRecord(int index,
const DtEnhancedFixedWingAircraftRecord& init);
virtual DtBoolean getEnhancedFixedWingAircraftRecord(int index,
DtEnhancedFixedWingAircraftRecord& record) const;

// Set/Get a BasicRotorWingAircraftRecord. Returns DtFalse if
the
// index is out of range, or if the entity category does not
match.
virtual DtBoolean setBasicRotorWingAircraftRecord(int index,
const DtBasicRotorWingAircraftRecord& init);
virtual DtBoolean getBasicRotorWingAircraftRecord(int index,
DtBasicRotorWingAircraftRecord& record) const;
```

```
// Set/Get an EnhancedRotorWingAircraftRecord. Returns DtFalse
if the
// index is out of range, or if the entity category does not
match.
virtual DtBoolean setEnhancedRotorWingAircraftRecord(int index,
    const DtEnhancedRotorWingAircraftRecord& init);
virtual DtBoolean getEnhancedRotorWingAircraftRecord(int index,
    DtEnhancedRotorWingAircraftRecord& record) const;

// Set/Get a BasicGroundCombatSoldierRecord. Returns DtFalse if
the
// index is out of range, or if the entity category does not
match.
virtual DtBoolean setBasicGroundCombatSoldierRecord(int index,
    const DtBasicGroundCombatSoldierRecord& init);
virtual DtBoolean getBasicGroundCombatSoldierRecord(int index,
    DtBasicGroundCombatSoldierRecord& record) const;

// Set/Get an EnhancedGroundCombatSoldierRecord. Returns DtFalse
if the
// index is out of range, or if the entity category does not
match.
virtual DtBoolean setEnhancedGroundCombatSoldierRecord(int index,
    const DtEnhancedGroundCombatSoldierRecord& init);
virtual DtBoolean getEnhancedGroundCombatSoldierRecord(int index,
    DtEnhancedGroundCombatSoldierRecord& record) const;

// Set/Get a BasicGroundCombatVehicleRecord. Returns DtFalse if
the
// index is out of range, or if the entity category does not
match.
virtual DtBoolean setBasicGroundCombatVehicleRecord(int index,
    const DtBasicGroundCombatVehicleRecord& init);
virtual DtBoolean getBasicGroundCombatVehicleRecord(int index,
    DtBasicGroundCombatVehicleRecord& record) const;

// Set/Get an EnhancedGroundCombatVehicleRecord. Returns DtFalse
if the
// index is out of range, or if the entity category does not
match.
virtual DtBoolean setEnhancedGroundCombatVehicleRecord(int index,
    const DtEnhancedGroundCombatVehicleRecord& init);
virtual DtBoolean getEnhancedGroundCombatVehicleRecord(int index,
    DtEnhancedGroundCombatVehicleRecord& record) const;

// Set/Get a GroundLogisticsVehicleRecord. Returns DtFalse if
the
// index is out of range, or if the entity category does not
match.
```



```

virtual DtBoolean setGroundLogisticsVehicleRecord(int index,
    const DtGroundLogisticsVehicleRecord& init);
virtual DtBoolean getGroundLogisticsVehicleRecord(int index,
    DtGroundLogisticsVehicleRecord& record) const;

// Print the data contained by this PDU
virtual void printData() const;

public:

    // Create a DtIsGroupOfPdu.
    // Caller is responsible for deletion.
    static DtPdu* create(const DtNetPacket* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // Add (register)/Remove (deregister) function to be called when
    // pdu is received.
    static void addCallback(DtExerciseConn* conn,
        DtIsGroupOfPduCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtIsGroupOfPduCb cb, void* usr);

protected:

    // Returns the internal PDU kind
    virtual DtPduKind internalGetPduKind() const;

    // Casts the netPdu element to our network struct
    virtual DtNetIsGroupOfPdu* netPduStruct() const;

    // Casts the 'this' pointer for changing elements in a const
    function
    // Non-virtual since you can't override based on return type
    DtIsGroupOfPdu* self() const;

    // Returns a pointer to the first byte of variable length data
    virtual char* var() const;

    // Returns the record size for the grouped entity category
    virtual int recordSize() const;

protected:

    // Size of PDU with zero parameter records.
    static int theStaticSize;

protected:

    // Temporary objects for reference returns
    DtEntityIdentifier myEntityID;

```

```
};  
#endif  
#endif
```

isPartOfPdu.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "hostStructs.h"
#include "pdu.h"
#include "exerciseConn.h"

#include "mathTypes.h"

#include "packets/isPartOfP.h"

// class DtIsPartOfPdu:
//
// Instances of DtIsPartOfPdu are used to join two simulation enti-
// ties to
// form a single entity.

// DtIsPartOfPdu callback function signature.
typedef void (*DtIsPartOfPduCb)(DtIsPartOfPdu* pdu, void* usr);

class DT_DLL_VL DtIsPartOfPdu : public DtPdu
{
public:

    // Constructors
    DtIsPartOfPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtIsPartOfPdu(const DtNetIsPartOfPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtIsPartOfPdu(const DtIsPartOfPdu& orig);

    // destructor
    virtual ~DtIsPartOfPdu();

    // Assignment operator
    DtIsPartOfPdu& operator=(const DtIsPartOfPdu& orig);

    // Set/Get originating entity identifier
    virtual void setOriginatingEntityID(const DtEntityIdentifier&
entity);
    virtual const DtEntityIdentifier& originatingEntityID() const;

    // Set/Get receiving entity identifier
    virtual void setReceivingEntityID(const DtEntityIdentifier&
entity);
    virtual const DtEntityIdentifier& receivingEntityID() const;

```

```
// Set/Get nature
virtual void setNature(DtNature& value);
virtual DtNature nature() const;

// Set/Get position
virtual void setPosition(DtPosition& value);
virtual DtPosition position() const;

// Set/Get part location
virtual void setLocation(DtConstVector coords);
virtual DtConstVector location() const;

// Set/Get station name
virtual void setStationName(DtStationName value);
virtual DtStationName stationName() const;

// Set/Get Station number
virtual void setStationNumber(DtU16 value);
virtual DtU16 stationNumber() const;

// Set/Get Part Entity Type
virtual void setEntityType(const DtEntityType& type);
virtual const DtEntityType& entityType() const;

// Print the data contained by this PDU
virtual void printData() const;
```

public:

```
// Create a DtIsPartOfPdu.
// Caller is responsible for deletion.
static DtPdu* create(const DtNetPacket* initial,
    DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

// Add (register)/Remove (deregister) function to be called when
// pdu is received.
static void addCallback(DtExerciseConn* conn,
    DtIsPartOfPduCb cb, void* usr);
static void removeCallback(DtExerciseConn* conn,
    DtIsPartOfPduCb cb, void* usr);
```

protected:

```
// Returns the internal PDU kind
virtual DtPduKind internalGetPduKind() const;

// Casts the netPdu element to our network struct
// Non-virtual since you can't override based on return type
virtual DtNetIsPartOfPdu* netPduStruct() const;
```

```
// Casts the 'this' pointer for changing elements in a const
function
// Non virtual since you can't override based on return type
DtIsPartOfPdu* self() const;

protected:

// Size of PDU with zero data bytes.
static int theStaticSize;

protected:

// Temporary objects for reference returns
DtEntityIdentifier myEntityID;
DtVector myVector;
DtEntityType myEntityType;

};
#endif
#endif
```

jammerBeamDec.h

```
#if DtHLA

#ifndef DtJammerBeamDecoder_H_
#define DtJammerBeamDecoder_H_

#include "chooseDefines.h"
#include "baseEmitttrDec.h"

// This is used to declare attribute decoding functions
// which are members of the class.
#define DtDECLARE_JAMMERBEAM_ATTR_DECODER(attrName) \
    DtDECLARE_ATTR_DECODER(DtEmitterBeamRepository, attrName)

// class DtJammerBeamDecoder:
//
// Instances of DtJammerBeamDecoder are

class DT_DLL_VL DtJammerBeamDecoder : public DtBaseEmitterBeamDe-
coder
{
public:

    // default constructor
    DtJammerBeamDecoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtJammerBeamDecoder();

    // copy constructor
    DtJammerBeamDecoder(const DtJammerBeamDecoder& orig);

    // assignment operator
    DtJammerBeamDecoder& operator=(const DtJammerBeamDecoder& orig);

public:

    // Create a DtBaseEntityDecoder. Caller is responsible for dele-
    // tion.
    static DtHlaStateDecoder* create(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

protected:

    // Add decoding functions for the attributes.
```

```

    // Note: Non-virtual since meant to be called from constructors
to
    //      add decoders.
    void addMyDecoders();

    // static decode<AttributeName> member functions.
    DtDECLARE_JAMMERBEAM_ATTR_DECODER(JammingModeSequence);
    DtDECLARE_JAMMERBEAM_ATTR_DECODER(JammedObjectIDs);
    DtDECLARE_JAMMERBEAM_ATTR_DECODER(HighDensityJam);
};
// These macros are used to define simple attribute decoder func-
tions
// (members of a derived DtHlaStateDecoder that decode
// individual attributes). The simple decoding is just to
// convert the value pulled out of the ahvps to a host type using
// the cast expression castExpr, and pass it to the mutator func-
tion.
#define DtDEFINE_SIMPLE_BASE_JAMMERBEAM_ATTR_DECODER_WITH_CAST( \
    attrName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_ATTR_DECODER_WITH_CAST( \
        DtJammerBeamDecoder, DtEmitterBeamRepository, \
        attrName, netType, mutator, castExpr)

#define DtDEFINE_SIMPLE_BASE_JAMMERBEAM_ATTR_DECODER( \
    attrName, netType, inspector) \
    DtDEFINE_SIMPLE_BASE_JAMMERBEAM_ATTR_DECODER_WITH_CAST( \
        attrName, netType, inspector, (netType))

#endif    //DtJammerBeamDecoder_H_

```

jammerBeamEnc.h

```
#if DtHLA

#ifndef DtJammerBeamEncoder_H_
#define DtJammerBeamEncoder_H_

#include "chooseDefines.h"
#include "baseEmitttrEnc.h"

// This is used to declare static member functions for determining
// whether or not an attribute needs to be encoded.
#define DtDECLARE_JAMMERBEAM_ATTR_CHECKER(attrName) \
    DtDECLARE_ATTR_CHECKER(DtEmitterBeamRepository, attrName)

// This is used to declare attribute encoding functions
// which are members of the class.
#define DtDECLARE_JAMMERBEAM_ATTR_ENCODER(attrName) \
    DtDECLARE_ATTR_ENCODER(DtEmitterBeamRepository, attrName)

// class DtJammerBeamEncoder:
//
// Instances of DtJammerBeamEncoder are

class DT_DLL_VL DtJammerBeamEncoder : public DtBaseEmitterBeamEncoder
{
public:

    // default constructor
    DtJammerBeamEncoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtJammerBeamEncoder();

    // copy constructor
    DtJammerBeamEncoder(const DtJammerBeamEncoder& orig);

    // assignment operator
    DtJammerBeamEncoder& operator=(const DtJammerBeamEncoder& orig);

public:

    // Create a DtBaseEntityEncoder. Caller is responsible for deletion.
    static DtHlaStateEncoder* create(
        DtExerciseConn* exConn,
```



```

        DtObjClassDesc* classDesc);

protected:

    DtDECLARE_JAMMERBEAM_ATTR_ENCODER(JammingModeSequence);
    DtDECLARE_JAMMERBEAM_ATTR_ENCODER(JammedObjectIDs);
    DtDECLARE_JAMMERBEAM_ATTR_ENCODER(HighDensityJam);

    DtDECLARE_JAMMERBEAM_ATTR_CHECKER(JammingModeSequence);
    DtDECLARE_JAMMERBEAM_ATTR_CHECKER(JammedObjectIDs);
    DtDECLARE_JAMMERBEAM_ATTR_CHECKER(HighDensityJam);
};
// This macro is used to define simple attribute encoder functions
// (members of a derived DtHlaStateEncoder that encode
// individual attributes). The simple encoding is just to implicitly
// convert the value returned by the inspector to a netType,
// and add it to the ahvps.
#define DtDEFINE_SIMPLE_JAMMERBEAM_ATTR_ENCODER( \
    attrName, netType, inspector) \
    DtDEFINE_SIMPLE_ATTR_ENCODER(DtJammerBeamEncoder, \
        DtEmitterBeamRepository, \
        attrName, netType, inspector)

// This macro is used to define simple attribute checker functions
// (members of a derived DtHlaStateEncoder that check whether
// a particular attribute is out of date. The simple implementation
// is to just compare the values returned by the two StateReps'
// inspector functions.
#define DtDEFINE_SIMPLE_JAMMERBEAM_ATTR_CHECKER( \
    attrName, inspector) \
    DtDEFINE_SIMPLE_ATTR_CHECKER(DtJammerBeamEncoder, \
        DtEmitterBeamRepository, \
        attrName, inspector)

#endif //DtJammerBeamEncoder_H_

```

laserPdu.h

```
#if !DtHLA

#if DtDISVERS == 3

#include "hostStructs.h"
#include "pdu.h"
#include "mathTypes.h"
#include "packets/laserP.h"
#define DtLaserStaticSize 72

class DT_DLL_VL DtLaserPdu;
typedef void (*DtLaserPduCb)(DtLaserPdu *pdu, void *usr);

class DT_DLL_VL DtLaserPdu : public DtPdu
{
public:

/***** STANDARD CONSTRUCTORS *****/

    DtLaserPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtLaserPdu(const DtNetLaserPdu *initial,
               DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    static DtPdu *create(const DtNetPacket *initial,
                        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    static void addCallback(DtExerciseConn *conn,
                           DtLaserPduCb cb, void *usr);
    static void removeCallback(DtExerciseConn *conn,
                               DtLaserPduCb cb, void *usr);

/***** INSPECTOR FUNCTIONS *****/

    const DtEntityIdentifier &lasingEntityId() const;
    DtLaserCodeName codeName() const;
    const DtEntityIdentifier &lasedEntityId() const;
    DtLaserCode laserCode() const;
    float power() const;
    float wavelength() const;
    DtConstVector lasedSpotRelative() const;
    DtConstVector lasedSpotWorld() const;

/***** MUTATOR FUNCTIONS *****/

    void setLasingEntityId(const DtEntityIdentifier &id);
    void setCodeName(DtLaserCodeName name);
    void setLasedEntityId(const DtEntityIdentifier &id);
    void setLaserCode(DtLaserCode code);
```

```
void setPower(float power);
void setWavelength(float wavelength);
void setLasedSpotRelative(DtConstVector loc);
void setLasedSpotWorld(DtConstVector loc);

void printData() const;
protected:
    DtPduKind internalGetPduKind() const {return DtLaserPduKind;}
    DtEntityIdentifier theId;
    DtVector loc;
};
#endif // DtDISVERS
#endif // !DtHLA
```

lcEnd.h

```
#include "lcPdu.h"

// class DtLgrEndPdu:
//
// Instances of DtLgrEndPdu are to tell remote logger to skip to end
// of file during playback.

class DT_DLL_VL DtLgrEndPdu : public DtLgrControlPdu
{
public:

    // default constructor
    DtLgrEndPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // construct from network representation
    DtLgrEndPdu(const DtNetLgrControlPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // construct and prepare for send
    DtLgrEndPdu(DtGlobalObjectDesignator& sender, DtGlobalObjectDes-
ignator& receiver,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // destructor
    virtual ~DtLgrEndPdu();

    // copy constructor
    DtLgrEndPdu(const DtLgrEndPdu& orig);

    // assignment operator
    DtLgrEndPdu& operator=(const DtLgrEndPdu& orig);
};
```

lcEofAction.h

```

#include "lcPdu.h"

// class DtLgrEofActionPdu:
//
// Instances of DtLgrEofActionPdu are to tell remote logger that
// it should pause, wrap, or stop at the end of playback.

class DT_DLL_VL DtLgrEofActionPdu : public DtLgrControlPdu
{
public:

    // default constructor
    DtLgrEofActionPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // construct from network representation
    DtLgrEofActionPdu(const DtNetLgrControlPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // construct and prepare for send
    DtLgrEofActionPdu(DtGlobalObjectDesignator& sender, DtGlobalOb-
        jectDesignator& receiver,
        DtLgrMode mode, DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // destructor
    virtual ~DtLgrEofActionPdu();

    // copy constructor
    DtLgrEofActionPdu(const DtLgrEofActionPdu& orig);

    // assignment operator
    DtLgrEofActionPdu& operator=(const DtLgrEofActionPdu& orig);

    // Set/Get value for what to do when Logger reaches EOF
    //
    virtual DtLgrMode action();

    // mutator functions
    virtual void setAction( DtLgrMode cmd );

    // return correctly casted parameters pointer; see lgrControlP.h
    virtual DtLgrIntParams* params();

private:
    // parameter
    DtLgrMode myAction;
};

```

lcHeartbeat.h

```
#include "lcPdu.h"

// class DtLgrHeartbeatPdu:
//
// Instances of DtLgrHeartbeatPdu are to tell remote logger to
// send packet for each playback entity at least every HEARTBEAT
// seconds. Important when in slow-motion.

class DT_DLL_VL DtLgrHeartbeatPdu : public DtLgrControlPdu
{
public:

    // default constructor
    DtLgrHeartbeatPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // construct from network representation
    DtLgrHeartbeatPdu(const DtNetLgrControlPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // construct and prepare for send
    DtLgrHeartbeatPdu(DtGlobalObjectDesignator& sender, DtGlobalOb-
    jectDesignator& receiver, DtReal hbeat,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // destructor
    virtual ~DtLgrHeartbeatPdu();

    // copy constructor
    DtLgrHeartbeatPdu(const DtLgrHeartbeatPdu& orig);

    // assignment operator
    DtLgrHeartbeatPdu& operator=(const DtLgrHeartbeatPdu& orig);

    // set/get the heartbeat. see class description.
    virtual void setHeartbeat( DtReal val);
    virtual DtReal heartbeat();

    // return correctly casted parameters pointer; see lgrControlP.h
    virtual DtLgrRealParams* params();

private:
    // parameter
    DtReal myHeartbeat;
};
```

lcPause.h

```
#include "lcPdu.h"

// class DtLgrPausePdu:
//
// Instances of DtLgrPausePdu are to tell remote logger to pause

class DT_DLL_VL DtLgrPausePdu : public DtLgrControlPdu
{
public:

    // default constructor
    DtLgrPausePdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // construct from network representation
    DtLgrPausePdu(const DtNetLgrControlPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // construct and prepare for send
    DtLgrPausePdu(DtGlobalObjectDesignator& sender, DtGlobalObject-
Designator& receiver,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // destructor
    virtual ~DtLgrPausePdu();

    // copy constructor
    DtLgrPausePdu(const DtLgrPausePdu& orig);

    // assignment operator
    DtLgrPausePdu& operator=(const DtLgrPausePdu& orig);
};
```

lcPdu.h

```
#ifndef DtLgrControlPdu_H_
#define DtLgrControlPdu_H_

#include "chooseDefines.h"
#include "MachineTypes.h"
#include "LibMatrix.h"
#include "printUtil.h"
#include "parseEnum.h"
#include "exerciseConn.h"
#include "hostStructs.h"
#include "pdu.h"
#include "packets/lgrControlP.h"
#include "mathTypes.h"
#include "interaction.h"

#if !DtHLA

class DT_DLL_VL DtExerciseConn;
class DT_DLL_VL DtLgrControlPdu;

typedef DtLgrControlPdu DtLgrControlInteraction;
typedef void (*DtLgrControlInteractionCb)
    (DtLgrControlInteraction* inter, void* usr);

#endif

// class DtLgrControlPdu:
//
// Instances of DtLgrControlPdu are a family of messgae used to
// control
// a remote MaK-Logger over the network. See lc*.h

#if DtHLA
class DT_DLL_VL DtLgrControlPdu : public DtPdu
#else
class DT_DLL_VL DtLgrControlPdu : public DtInteraction
#endif
{
public:

    // default constructor
    DtLgrControlPdu();

    //Create a PDU representation from a network packet.
    static DtPdu* create(const DtNetPacket* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
}
```

```

    //Create a PDU from DtLgrCommand;
    static DtLgrControlPdu* create(DtLgrCommand command);

#if !DtHLA
    //DIS -- install/remove handler to call when receiving a LC
    packet.
    static void addCallback(DtExerciseConn* conn, DtLgrControlInter-
    actionCb cb,
        void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtLgrControlInteractionCb cb, void* usr);
#endif

    // destructor
    virtual ~DtLgrControlPdu();

    // copy constructor
    DtLgrControlPdu(const DtLgrControlPdu& orig);

    // assignment operator
    DtLgrControlPdu& operator=(const DtLgrControlPdu& orig);

    // set/get Object Identifier of intended recipient (logger)
    virtual void setReceiverId(const DtGlobalObjectDesignator& id);
    const DtGlobalObjectDesignator& receiverId() const;

    // set/get Object Identifier of sender, if applicable.
    virtual void setSenderId(const DtGlobalObjectDesignator& id);
    virtual const DtGlobalObjectDesignator& senderId() const;

    // get message-type for proper casting to a subclass.
    DtLgrCommand lgrCommand() const;

    //print contents
    virtual void printData() const;

protected:
    // identity for use by toolkit.
    DtPduKind internalGetPduKind() const;

    // set is done by way of constructing the proper subclass.
    void setLgrCommand(DtLgrCommand com);

private:
    // parameter
    DtGlobalObjectDesignator mySenderId;
    DtGlobalObjectDesignator myReceiverId;
};

inline DtPduKind DtLgrControlPdu::internalGetPduKind() const

```

```
{
    return DtLgrControlPduKind;
}

//General operation modes

enum DtLgrMode
{
    DtLGR_PLAY,
    DtLGR_PAUSE,
    DtLGR_RECORD,
    DtLGR_STOP,
    DtLGR_EMPTY,
    DtLGR_LOOP
};

//Play forwards or backwards when in LGR_PLAY mode
enum DtLgrDirection
{
    DtLGR_STOPPED    =  0,
    DtLGR_NOPKT      =  0,
    DtLGR_FORWARDS   =  1,
    DtLGR_BACKWARDS  = -1
};

//Location status
enum DtLgrFilePosition
{
    DtLGR_NOFILE,
    DtLGR_START,
    DtLGR_MIDDLE,
    DtLGR_END
};

enum DtLgrStyle
{
    DtLGR_PRISTINE,
    DtLGR_RVA
};

//Receiving application may filter lcPdu based on this start-up
flag.
enum DtLgrProcess
{
    DtLGR_NONE,
    DtLGR_MINE,
    DtLGR_ALL
};

#endif
```

lcPlay.h

```
#include "lcPdu.h"

// class DtLgrPlayPdu:
//
// Instances of DtLgrPlayPdu are used to tell remote logger to begin
// recording

class DT_DLL_VL DtLgrPlayPdu : public DtLgrControlPdu
{
public:

    // default constructor
    DtLgrPlayPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // construct from network representation
    DtLgrPlayPdu(const DtNetLgrControlPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // construct and prepare for send
    DtLgrPlayPdu(DtGlobalObjectDesignator& sender, DtGlobalObjectDes-
ignator& receiver,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // destructor
    virtual ~DtLgrPlayPdu();

    // copy constructor
    DtLgrPlayPdu(const DtLgrPlayPdu& orig);

    // assignment operator
    DtLgrPlayPdu& operator=(const DtLgrPlayPdu& orig);
};
```

lcPlayAction.h

```
#include "lcPdu.h"

// class DtLgrPlayActionPdu:
//
// Instances of DtLgrPlayActionPdu are to tell remote logger
// what action to take initially when set to playback mode.

class DT_DLL_VL DtLgrPlayActionPdu : public DtLgrControlPdu
{
public:

    // default constructor
    DtLgrPlayActionPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // construct from network representation
    DtLgrPlayActionPdu(const DtNetLgrControlPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // construct and prepare for send
    DtLgrPlayActionPdu(DtGlobalObjectDesignator& sender, DtGlobalOb-
    jectDesignator& receiver,
        DtLgrMode mode, DtBufferPtr buffer =DtUSE_INTERNAL_BUFFER);

    // destructor
    virtual ~DtLgrPlayActionPdu();

    // copy constructor
    DtLgrPlayActionPdu(const DtLgrPlayActionPdu& orig);

    // assignment operator
    DtLgrPlayActionPdu& operator=(const DtLgrPlayActionPdu& orig);

    // set/get action to take place when logger switched to playback
    mode.
    virtual void setAction( DtLgrMode cmd );
    virtual DtLgrMode action();

    // return correctly casted parameters pointer; see lgrControlP.h
    virtual DtLgrIntParams* params();

private:
    // parameter
    DtLgrMode myAction;
};
```

lcPlayFile.h

```

#include "lcPdu.h"

// class DtLgrPbFilePdu:
//
// Instances of DtLgrPbFilePdu are used for switching to play-mode
// and defining
// which file to read from when sending traffic into the simulation.

class DT_DLL_VL DtLgrPbFilePdu : public DtLgrControlPdu
{
public:

    // default constructor
    DtLgrPbFilePdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // construct from network representation
    DtLgrPbFilePdu(const DtNetLgrControlPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // construct and prepare for send
    DtLgrPbFilePdu(DtGlobalObjectDesignator& sender, DtGlobalObject-
Designator& receiver, char* filename,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // destructor
    virtual ~DtLgrPbFilePdu();

    // copy constructor
    DtLgrPbFilePdu(const DtLgrPbFilePdu& orig);

    // assignment operator
    DtLgrPbFilePdu& operator=(const DtLgrPbFilePdu& orig);

    // set/get file to open for playback. (Path relative to logger)
    virtual void setFilename(const char* path);
    virtual const char* filename();

    // return correctly casted parameters pointer; see lgrControlP.h
    virtual DtLgrStringParams* params();

    // diagnostic output
    virtual void printData() const;

private:
    // parameter
    char* myFilename;
};

```

lcQuit.h

```
#include "lcPdu.h"

// class DtLgrQuitPdu:
//
// Instances of DtLgrQuitPdu are messages telling a remote logger to
// quit

class DT_DLL_VL DtLgrQuitPdu : public DtLgrControlPdu
{
public:

    // default constructor
    DtLgrQuitPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // construct from network representation
    DtLgrQuitPdu(const DtNetLgrControlPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // construct and prepare for send
    DtLgrQuitPdu(DtGlobalObjectDesignator& sender, DtGlobalObjectDes-
ignator& receiver,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // destructor
    virtual ~DtLgrQuitPdu();

    // copy constructor
    DtLgrQuitPdu(const DtLgrQuitPdu& orig);

    // assignment operator
    DtLgrQuitPdu& operator=(const DtLgrQuitPdu& orig);
};
```

lcRecAction.h

```

#include "lcPdu.h"

// class DtLgrRecordActionPdu:
//
// Instances of DtLgrRecordActionPdu are to tell remote logger to
// stop or begin recording when entering record-mode.

class DT_DLL_VL DtLgrRecordActionPdu : public DtLgrControlPdu
{
public:

    // default constructor
    DtLgrRecordActionPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // construct from network representation
    DtLgrRecordActionPdu(const DtNetLgrControlPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // construct and prepare for send
    DtLgrRecordActionPdu(DtGlobalObjectDesignator& sender, DtGlo-
balObjectDesignator& receiver,
        DtLgrMode mode, DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // destructor
    virtual ~DtLgrRecordActionPdu();

    // copy constructor
    DtLgrRecordActionPdu(const DtLgrRecordActionPdu& orig);

    // assignment operator
    DtLgrRecordActionPdu& operator=(const DtLgrRecordActionPdu&
orig);

    // set/get action to take when switching to record-mode
    virtual void setAction( DtLgrMode cmd );
    virtual DtLgrMode action();

    // return correctly casted parameters pointer; see lgrControlP.h
    virtual DtLgrIntParams* params();

private:
    // parameter
    DtLgrMode myAction;
};

```

lcRecFile.h

```
#include "lcPdu.h"

// class DtLgrRecFilePdu:
//
// Instances of DtLgrRecFilePdu are used to switching remote logger
// to
// recording-mode and to set the file into which to store captured
// simulation traffic

class DT_DLL_VL DtLgrRecFilePdu : public DtLgrControlPdu
{
public:

    // default constructor
    DtLgrRecFilePdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // construct from network representation
    DtLgrRecFilePdu(const DtNetLgrControlPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // construct and prepare for send
    DtLgrRecFilePdu(DtGlobalObjectDesignator& sender, DtGlobalObject-
Designator& receiver,
        char* filename, DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // destructor
    virtual ~DtLgrRecFilePdu();

    // copy constructor
    DtLgrRecFilePdu(const DtLgrRecFilePdu& orig);

    // assignment operator
    DtLgrRecFilePdu& operator=(const DtLgrRecFilePdu& orig);

    // set/get the name of file into which traffic will be recorded
    virtual void setFilename(const char* path);
    virtual const char* filename();

    // return correctly casted parameters pointer; see lgrControlP.h
    virtual DtLgrStringParams* params();

    // diagnostic output
    virtual void printData() const;

private:
    // parameter
    char* myFilename;
};
```


lcRecord.h

```
#include "lcPdu.h"

// class DtLgrRecordPdu:
//
// Instances of DtLgrRecordPdu are to tell remote logger to begin
// recording

class DT_DLL_VL DtLgrRecordPdu : public DtLgrControlPdu
{
public:

    // default constructor
    DtLgrRecordPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // construct from network representation
    DtLgrRecordPdu(const DtNetLgrControlPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // construct and prepare for send
    DtLgrRecordPdu(DtGlobalObjectDesignator& sender, DtGlobalObject-
Designator& receiver,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // destructor
    virtual ~DtLgrRecordPdu();

    // copy constructor
    DtLgrRecordPdu(const DtLgrRecordPdu& orig);

    // assignment operator
    DtLgrRecordPdu& operator=(const DtLgrRecordPdu& orig);
};
```

lcSetTime.h

```
#include "lcPdu.h"

// class DtLgrSetTimePdu:
//
// Instances of DtLgrSetTimePdu are to tell remote logger to
// go to set position to HH:MM:SS

class DT_DLL_VL DtLgrSetTimePdu : public DtLgrControlPdu
{
public:

    // default constructor
    DtLgrSetTimePdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // construct from network representation
    DtLgrSetTimePdu(const DtNetLgrControlPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // construct and prepare for send
    DtLgrSetTimePdu(DtGlobalObjectDesignator& sender, DtGlobalObject-
Designator& receiver,
        DtTime tm, DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // destructor
    virtual ~DtLgrSetTimePdu();

    // copy constructor
    DtLgrSetTimePdu(const DtLgrSetTimePdu& orig);

    // assignment operator
    DtLgrSetTimePdu& operator=(const DtLgrSetTimePdu& orig);

    // set/get time to jump to within logger playback file
    virtual void setTime(const DtTime time);
    virtual DtTime time();

    // return correctly casted parameters pointer; see lgrControlP.h
    virtual DtLgrTimeParams* params();

private:
    // parameter
    DtTime myTime;
};
```

lcSkipTime.h

```

#include "lcPdu.h"

// class DtLgrSkipTimePdu:
//
// Instances of DtLgrSetTimePdu are to tell remote logger to
// forward or backward HH:MM:SS

class DT_DLL_VL DtLgrSkipTimePdu : public DtLgrControlPdu
{
public:

    // default constructor
    DtLgrSkipTimePdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // construct from network representation
    DtLgrSkipTimePdu(const DtNetLgrControlPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // construct and prepare for send
    DtLgrSkipTimePdu(DtGlobalObjectDesignator& sender, DtGlobalOb-
jectDesignator& receiver,
        DtTime tm, DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // destructor
    virtual ~DtLgrSkipTimePdu();

    // copy constructor
    DtLgrSkipTimePdu(const DtLgrSkipTimePdu& orig);

    // assignment operator
    DtLgrSkipTimePdu& operator=(const DtLgrSkipTimePdu& orig);

    // set/get amount of time to skip forward or back in playback file
    virtual void setTime(DtTime dt);
    virtual DtTime time();

    // return correctly casted parameters pointer; see lgrControlP.h
    virtual DtLgrTimeParams* params();

private:
    // parameter
    DtTime myTime;
};

```

lcStart.h

```
#include "lcPdu.h"

// class DtLgrStartPdu:
//
// Instances of DtLgrStartPdu are used to tell remote logger to
// start recording

class DT_DLL_VL DtLgrStartPdu : public DtLgrControlPdu
{
public:

    // default constructor
    DtLgrStartPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // construct from network representation
    DtLgrStartPdu(const DtNetLgrControlPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // construct and prepare for send
    DtLgrStartPdu(DtGlobalObjectDesignator& sender, DtGlobalObject-
Designator& receiver,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // destructor
    virtual ~DtLgrStartPdu();

    // copy constructor
    DtLgrStartPdu(const DtLgrStartPdu& orig);

    // assignment operator
    DtLgrStartPdu& operator=(const DtLgrStartPdu& orig);
};
```

lcStop.h

```
#include "lcPdu.h"

// class DtLgrStopPdu:
//
// Instances of DtLgrStopPdu are to tell remote logger to stop

class DT_DLL_VL DtLgrStopPdu : public DtLgrControlPdu
{
public:

    // default constructor
    DtLgrStopPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // construct from network representation
    DtLgrStopPdu(const DtNetLgrControlPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // construct and prepare for send
    DtLgrStopPdu(DtGlobalObjectDesignator& sender, DtGlobalObjectDes-
ignator& receiver,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // destructor
    virtual ~DtLgrStopPdu();

    // copy constructor
    DtLgrStopPdu(const DtLgrStopPdu& orig);

    // assignment operator
    DtLgrStopPdu& operator=(const DtLgrStopPdu& orig);
};
```

lcTimeScale.h

```
#include "lcPdu.h"

// class DtLgrTimeScalePdu:
//
// Instances of DtLgrTimeScalePdu are to tell remote logger to
// slow down or speed up playback. 1.0 is normal speed

class DT_DLL_VL DtLgrTimeScalePdu : public DtLgrControlPdu
{
public:

    // default constructor
    DtLgrTimeScalePdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // construct from network representation
    DtLgrTimeScalePdu(const DtNetLgrControlPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // construct and prepare for send
    DtLgrTimeScalePdu(DtGlobalObjectDesignator& sender, DtGlobalOb-
        jectDesignator& receiver, double scale,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // destructor
    virtual ~DtLgrTimeScalePdu();

    // copy constructor
    DtLgrTimeScalePdu(const DtLgrTimeScalePdu& orig);

    // assignment operator
    DtLgrTimeScalePdu& operator=(const DtLgrTimeScalePdu& orig);

    // set/get playback speed
    virtual void setTimeScale( DtReal time);
    virtual double timeScale();

    // return correctly casted parameters pointer; see lgrControlP.h
    virtual DtLgrRealParams* params();

private:
    // parameter
    DtReal myScale;
};
```

lcTimeout.h

```

#include "lcPdu.h"

// class DtLgrTimeoutPdu:
//
// Instances of DtLgrTimeoutPdu are to tell remote logger how long
// to generate
// interim packets for entity before assuming no more packets in
// file.
// Useful during slow-motion playback.

class DT_DLL_VL DtLgrTimeoutPdu : public DtLgrControlPdu
{
public:

    // default constructor
    DtLgrTimeoutPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // construct from network representation
    DtLgrTimeoutPdu(const DtNetLgrControlPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // construct and prepare for send
    DtLgrTimeoutPdu(DtGlobalObjectDesignator& sender, DtGlobalObject-
Designator& receiver,
        DtReal tmout, DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // destructor
    virtual ~DtLgrTimeoutPdu();

    // copy constructor
    DtLgrTimeoutPdu(const DtLgrTimeoutPdu& orig);

    // assignment operator
    DtLgrTimeoutPdu& operator=(const DtLgrTimeoutPdu& orig);

    // set/get timeout.  see class description
    virtual void setTimeout( DtReal val);
    virtual DtReal timeout();

    // return correctly casted parameters pointer; see lgrControlP.h
    virtual DtLgrRealParams* params();

private:
    // parameter
    DtReal myTimeout;
};

```

leAppearPdu.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "hostStructs.h"
#include "pdu.h"
#include "mathTypes.h"
#include "exerciseConn.h"

#include "packets/leAppearP.h"

#include "leEntityID.h"

// class DtLeAppearancePdu:
//
// Instances of DtLeAppearancePdu are used to send appearance infor-
// mation
// for live entities.

// DtLeAppearancePdu callback function signature.
typedef void (*DtLeAppearancePduCb)(DtLeAppearancePdu* pdu, void*
usr);

class DT_DLL_VL DtLeAppearancePdu : public DtPdu
{
public:

    // Constructors
    DtLeAppearancePdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtLeAppearancePdu(const DtNetLeAppearancePdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtLeAppearancePdu(const DtLeAppearancePdu& orig);

    // destructor
    virtual ~DtLeAppearancePdu();

    // Assignment operator
    DtLeAppearancePdu& operator=(const DtLeAppearancePdu& orig);

    // Set/Get the sub-protocol number
    virtual void setSubProtocol(DtSubProtocol value);
    virtual DtSubProtocol subProtocol() const;

    // Set/Get entity ID
    virtual void setEntityID(const DtLeEntityID& id);
    virtual const DtLeEntityID& entityID() const;
```



```
// Get the first appearance flag field (this field is automati-
cally
// maintained by the PDU as other PDU fields are set)
virtual DtU8 appearanceField1() const;

// Get the value for a specific appearance flag from the first
// appearance field
virtual DtBoolean appearanceField1Flag(DtLE_AppearanceField1Flag
flag) const;

// The following accessor functions refer to records which are
optional.
// The set methods create the record in the PDU if it does not
already
// exist. The get methods return DtFalse if the record does not
exist.

// Get the second appearance flag field (this field is automati-
cally
// maintained by the PDU as other PDU fields are set)
virtual DtBoolean getAppearanceField2(DtU8& value) const;

// Get the value for a specific appearance flag from the second
// appearance field
virtual DtBoolean
getAppearanceField2Flag(DtLE_AppearanceField2Flag flag,
DtBoolean& value) const;

// Set/Get the force ID
virtual void setForceID(DtForceType force);
virtual DtBoolean getForceID(DtForceType& force) const;

// Set/Get entity type
virtual void setEntityType(const DtEntityType& type);
virtual DtBoolean getEntityType(DtEntityType& type) const;

// Set/Get alternative entity type
virtual void setAlternativeEntityType(const DtEntityType& type);
virtual DtBoolean getAlternativeEntityType(DtEntityType& type)
const;

// Set/Get marking character set
virtual void setMarkingCharacterSet(DtCharacterSet value);
virtual DtBoolean getMarkingCharacterSet(DtCharacterSet& value)
const;

// Set/Get marking text (copies 11 bytes)
virtual void setMarkingText(DtU8* dataPtr);
virtual DtBoolean getMarkingText(DtU8* dataPtr) const;
```

```
// Set/Get capabilities (as a 32-bit integer)
virtual void setCapabilities(DtU32 value);
virtual DtBoolean getCapabilities(DtU32& value) const;

// Set/Get a particular capability (as a Boolean)
virtual void setCapability(DtCapability flag, DtBoolean value);
virtual DtBoolean getCapability(DtCapability flag, DtBoolean&
value) const;

// Set/Get visual appearance (as a 32 bit integer)
virtual void setVisualAppearance(DtU32 value);
virtual DtBoolean getVisualAppearance(DtU32& value) const;

// Set/Get appearance by attribute
virtual void setVisualAppearanceAttribute(
    DtAppearAttribute attr, DtAppearAttrVal val);
virtual DtBoolean getVisualAppearanceAttribute(
    DtAppearAttribute attr, DtAppearAttrVal& val);

// Set/Get IR appearance (as a 32 bit integer)
virtual void setIR_Appearance(DtU32 value);
virtual DtBoolean getIR_Appearance(DtU32& value) const;

// Set/Get EM appearance (as a 32 bit integer)
virtual void setEM_Appearance(DtU32 value);
virtual DtBoolean getEM_Appearance(DtU32& value) const;

// Set/Get audio appearance (as a 32 bit integer)
virtual void setAudioAppearance(DtU32 value);
virtual DtBoolean getAudioAppearance(DtU32& value) const;

// Print the data contained by this PDU
virtual void printData() const;

// Print the header (override to include the new sub-protocol
field)
virtual void printHeader() const;

public:

    // Create a DtLeAppearancePdu.
    // Caller is responsible for deletion.
    static DtPdu* create(const DtNetPacket* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // Add (register)/Remove (deregister) function to be called when
    // pdu is received.
    static void addCallback(DtExerciseConn* conn,
        DtLeAppearancePduCb cb, void* usr);
```

```
static void removeCallback(DtExerciseConn* conn,
    DtLeAppearancePduCb cb, void* usr);
```

protected:

```
// Returns the internal PDU kind
virtual DtPduKind internalGetPduKind() const;

// Casts the netPdu element to our network struct
virtual DtNetLeAppearancePdu* netPduStruct() const;

// Casts the 'this' pointer for changing elements in a const
function
// Non-virtual since you can't override based on return type
DtLeAppearancePdu* self() const;

// Returns a pointer to the first byte of variable length data
virtual char* var() const;

// Set appearance field 1
virtual void setAppearanceField1(DtU8 value);

// Set a flag of appearance field 1
virtual void setAppearanceField1Flag(DtLE_AppearanceField1Flag
flag);

// Clear a flag and all following flags in appearance field 1
virtual void clearField1Flags(DtLE_AppearanceField1Flag flag);

// Set appearance field 2
virtual void setAppearanceField2(DtU8 value);

// Set a flag of appearance field 2
virtual void setAppearanceField2Flag(DtLE_AppearanceField2Flag
flag);

// Returns the offset (in bytes) of a field in the variable data
region.
virtual int appearanceField1Offset() const;
virtual int appearanceField2Offset() const;
virtual int forceID_Offset() const;
virtual int entityTypeOffset() const;
virtual int alternativeEntityTypeOffset() const;
virtual int entityMarkingOffset() const;
virtual int capabilitiesOffset() const;
virtual int visualAppearanceOffset() const;
virtual int IR_AppearanceOffset() const;
virtual int EM_AppearanceOffset() const;
virtual int audioAppearanceOffset() const;
```

protected:

```
// Size of PDU with zero parameter records.  
static int theStaticSize;
```

protected:

```
// Temporary objects for reference returns  
DtLeEntityID myEntityID;
```

```
};  
#endif  
#endif
```

leArtPartsPdu.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "hostStructs.h"
#include "pdu.h"
#include "exerciseConn.h"
#include "artPartList.h"
#include "attPartList.h"

#include "packets/leArtPartsP.h"

#include "leEntityID.h"

// class DtLeArticulatedPartsPdu:
//
// Instances of DtLeArticulatedPartsPdu are used to send information
// about
// articulated and attached parts for live entities.

// DtLeArticulatedPartsPdu callback function signature.
typedef void (*DtLeArticulatedPartsPduCb)(
    DtLeArticulatedPartsPdu* pdu, void* usr);

class DT_DLL_VL DtLeArticulatedPartsPdu : public DtPdu
{
public:
    // Constructors
    DtLeArticulatedPartsPdu(DtBufferPtr buffer =
DtUSE_INTERNAL_BUFFER);
    DtLeArticulatedPartsPdu(const DtNetLeArticulatedPartsPdu*
initial,
    DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtLeArticulatedPartsPdu(const DtLeArticulatedPartsPdu& orig);

    // destructor
    virtual ~DtLeArticulatedPartsPdu();

    // Assignment operator
    DtLeArticulatedPartsPdu& operator=(const DtLeArticulatedPartsPdu&
orig);

    // Set/Get the sub-protocol number
    virtual void setSubProtocol(DtSubProtocol value);
    virtual DtSubProtocol subProtocol() const;

```

```
// Set/Get entity ID
virtual void setEntityID(const DtLeEntityID& id);
virtual const DtLeEntityID& entityID() const;

// Get the number of articulation parameter records (this field
is
// automatically set by the PDU when the articulated part list or
attached
// part list is set)
virtual DtU8 numRecords() const;

// The articulated parts and attached parts information are both
optional.
// Either or both can be set.

// Set/Get articulated parts parameters. Parameters are passed
in the form
// of an articulated parts list. Returns DtFalse if the opera-
tion could
// not be performed.
virtual DtBoolean setArtPartParameters(const DtArtPartList&
partList);
virtual DtBoolean getArtPartParameters(DtArtPartList& partList)
const;

// Set/Get attached parts parameters. Parameters are passed in
the form
// of an attached parts list. Returns DtFalse if the operation
could
// not be performed.
virtual DtBoolean setAttPartParameters(const DtAttachedPartList&
partList);
virtual DtBoolean getAttPartParameters(DtAttachedPartList&
partList) const;

// Print the data contained by this PDU
virtual void printData() const;

// Print the header (override to include the new sub-protocol
field)
virtual void printHeader() const;

public:

// Create a DtLeArticulatedPartsPdu.
// Caller is responsible for deletion.
static DtPdu* create(const DtNetPacket* initial,
DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
```

```
// Add (register)/Remove (deregister) function to be called when
// pdu is received.
static void addCallback(DtExerciseConn* conn,
    DtLeArticulatedPartsPduCb cb, void* usr);
static void removeCallback(DtExerciseConn* conn,
    DtLeArticulatedPartsPduCb cb, void* usr);
```

protected:

```
// Returns the internal PDU kind
virtual DtPduKind internalGetPduKind() const;

// Casts the netPdu element to our network struct
virtual DtNetLeArticulatedPartsPdu* netPduStruct() const;

// Casts the 'this' pointer for changing elements in a const
function
// Non-virtual since you can't override based on return type
DtLeArticulatedPartsPdu* self() const;

// Returns a pointer to the first byte of variable length data
virtual char* var() const;

// Set number of records
virtual void setNumRecords(DtU8 value);

// Returns the offset (in bytes) of a field in the variable data
region.
virtual int numRecordsOffset() const;
virtual int artParmOffset() const;
virtual int attParmOffset() const;
```

protected:

```
// Size of PDU with zero parameter records.
static int theStaticSize;
```

protected:

```
// Number of articulated part parameters for this PDU
int myNumArtPartParams;

// Number of attached part parameters for this PDU
int myNumAttPartParams;

// Temporary objects for reference returns
DtLeEntityID myEntityID;
};
#endif
#endif
```

leDeadReck.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"

#include "packets/leTspiP.h"

// class DtLeDeadReckoningParameters:
//
// DtLeDeadReckoningParameters is a class which contains the entity
// dead reckoning parameter information for the live entity PDUs.

class DT_DLL_VL DtLeDeadReckoningParameters
{
public:

    // Constructors
    DtLeDeadReckoningParameters();
    DtLeDeadReckoningParameters(const DtLeDeadReckoningParameters&
orig);

    // Assignment operator
    DtLeDeadReckoningParameters& operator=(
        const DtLeDeadReckoningParameters& orig);
    DtLeDeadReckoningParameters& operator=(
        const DtNetLeDeadReckoningParameters& orig);

    // Destructor
    virtual ~DtLeDeadReckoningParameters();

    // Set/Get dead reckoning algorithm
    virtual void setDeadReckoningAlgorithm(DtDeadReckonTypes value);
    virtual DtDeadReckonTypes deadReckoningAlgorithm() const;

    // Set/Get entity linear acceleration. This PDU field is only
    24-bits wide.
    virtual void setLinearAcceleration(DtU32 value);
    virtual DtU32 linearAcceleration() const;

    // Set/Get entity angular velocity. This PDU field is only 24-
    bits wide.
    virtual void setAngularVelocity(DtU32 value);
    virtual DtU32 angularVelocity() const;

    // Print the information contained in the record
```



```
virtual void print() const;

// Load/Unload data from a packet record
virtual void unloadNetRecord(
    const DtNetLeDeadReckoningParameters& record);
virtual void loadNetRecord(
    DtNetLeDeadReckoningParameters& record) const;

protected:

    // Data members:
    DtDeadReckonTypes          myDeadReckoningAlgorithm;
    DtU32                      myLinearAcceleration;
    DtU32                      myAngularVelocity;

};
#endif
#endif
```

leDetonPdu.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "hostStructs.h"
#include "pdu.h"
#include "mathTypes.h"
#include "exerciseConn.h"

#include "packets/leDetonP.h"

#include "leEntityID.h"
#include "leLocWorld.h"
#include "leLocEntity.h"
#include "leEntityVel.h"
#include "leEntityOri.h"

// class DtLeDetonationPdu:

// Instances of DtLeDetonationPdu are used to send firing informa-
// tion for live
// entities.

// DtLeDetonationPdu callback function signature.
typedef void (*DtLeDetonationPduCb)(DtLeDetonationPdu* pdu, void*
usr);

class DT_DLL_VL DtLeDetonationPdu : public DtPdu
{
public:

    // Constructors
    DtLeDetonationPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtLeDetonationPdu(const DtNetLeDetonationPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtLeDetonationPdu(const DtLeDetonationPdu& orig);

    // destructor
    virtual ~DtLeDetonationPdu();

    // Assignment operator
    DtLeDetonationPdu& operator=(const DtLeDetonationPdu& orig);

    // Set/Get the sub-protocol number
    virtual void setSubProtocol(DtSubProtocol value);
    virtual DtSubProtocol subProtocol() const;
```

```
// Set/Get firing entity ID
virtual void setFiringEntityID(const DtLeEntityID& id);
virtual const DtLeEntityID& firingEntityID() const;

// Get flags field 1 (this field is automatically maintained by
the
// PDU as other PDU fields are set)
virtual DtU8 detonFlagsField() const;

// Get flags field 2 (this field is automatically maintained by
the
// PDU as other PDU fields are set). Returns DtFalse if this
field is
// not present.
virtual DtBoolean getDeton2FlagsField(DtU8& field) const;

// Get the value for a specific flag from the first flags field
virtual DtBoolean detonFlag(DtLE_DetonFieldFlag flag) const;

// Get the value for a specific flag from the second flags field.
// Returns DtFalse if the field is not present in the PDU.
virtual DtBoolean getDeton2Flag(
    DtLE_DetonField2Flag flag, DtBoolean& value) const;

// Some of following accessor functions refer to records which
are optional.
// The set methods create the record in the PDU if it does not
already
// exist. The get methods return DtFalse if the record does not
exist.

// Set/Get target entity ID (optional)
virtual void setTargetEntityID(const DtLeEntityID& id);
virtual DtBoolean getTargetEntityID(DtLeEntityID& id) const;

// Set/Get munition ID site (optional)
virtual void setMunitionID_Site(DtU8 value);
virtual DtBoolean getMunitionID_Site(DtU8& value) const;

// Set/Get munition ID application (optional)
virtual void setMunitionID_Application(DtU8 value);
virtual DtBoolean getMunitionID_Application(DtU8& value) const;

// Set/Get munition ID entity (optional)
virtual void setMunitionID_Entity(DtU16 value);
virtual DtBoolean getMunitionID_Entity(DtU16& value) const;

// Set/Get event ID site (optional)
virtual void setEventID_Site(DtU8 value);
```

```
virtual DtBoolean getEventID_Site(DtU8& value) const;

// Set/Get event ID application (optional)
virtual void setEventID_Application(DtU8 value);
virtual DtBoolean getEventID_Application(DtU8& value) const;

// Set/Get event ID event
virtual void setEventID_Event(DtU16 value);
virtual DtBoolean getEventID_Event(DtU16& value) const;

// Set/Get entity location (world coordinates). In this PDU,
entity
// location is stored in world coordinates or entity coordinates
but not
// both. Get returns DtFalse if the coordinates are stored in
entity
// coordinates.
virtual void setLocationWorld(
    const DtLeEntityLocationWorldCoordinates& value);
virtual DtBoolean getLocationWorld(
    DtLeEntityLocationWorldCoordinates& value) const;

// Set/Get entity location (entity coordinates). In this PDU,
entity
// location is stored in world coordinates or entity coordinates
but not
// both. Get returns DtFalse if the coordinates are stored in
world
// coordinates.
virtual void setLocationEntity(
    const DtLeEntityLocationEntityCoordinates& value);
virtual DtBoolean getLocationEntity(
    DtLeEntityLocationEntityCoordinates& value) const;

// Set/Get entity linear velocity
virtual void setVelocity(const DtLeEntityLinearVelocity& value);
virtual const DtLeEntityLinearVelocity& velocity() const;

// Set/Get entity orientation
virtual void setOrientation(const DtLeEntityOrientation& value);
virtual DtBoolean getOrientation(DtLeEntityOrientation& value)
const;

// Set/Get burst munition type
virtual void setBurstMunitionType(DtEntityType& type);
virtual const DtEntityType& burstMunitionType() const;

// Set/Get burst warhead (optional)
virtual void setBurstWarhead(DtU16 value);
virtual DtBoolean getBurstWarhead(DtU16& value) const;
```

```
// Set/Get burst fuse (optional)
virtual void setBurstFuse(DtU16 value);
virtual DtBoolean getBurstFuse(DtU16& value) const;

// Set/Get burst quantity (optional)
virtual void setBurstQuantity(DtU16 value);
virtual DtBoolean getBurstQuantity(DtU16& value) const;

// Set/Get burst rate (optional)
virtual void setBurstRate(DtU16 value);
virtual DtBoolean getBurstRate(DtU16& value) const;

// Set/Get detonation result
virtual void setDetonationResult(DtU8 value);
virtual DtU8 detonationResult() const;

// Print the data contained by this PDU
virtual void printData() const;

// Print the header (override to include the new sub-protocol
field)
virtual void printHeader() const;

public:

    // Create a DtLeDetonationPdu.
    // Caller is responsible for deletion.
    static DtPdu* create(const DtNetPacket* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // Add (register)/Remove (deregister) function to be called when
    // pdu is received.
    static void addCallback(DtExerciseConn* conn,
        DtLeDetonationPduCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtLeDetonationPduCb cb, void* usr);

protected:

    // Returns the internal PDU kind
    virtual DtPduKind internalGetPduKind() const;

    // Casts the netPdu element to our network struct
    virtual DtNetLeDetonationPdu* netPduStruct() const;

    // Casts the 'this' pointer for changing elements in a const
    function
    // Non-virtual since you can't override based on return type
    DtLeDetonationPdu* self() const;
```

```
// Returns a pointer to the first byte of variable length data
virtual char* var() const;

// Set first detonation flags field
virtual void setDetonFlagsField(DtU8 value);

// Set second detonation flags field
virtual void setDeton2FlagsField(DtU8 value);

// Set first detonation flag
virtual void setDetonFlag(DtLE_DetonFieldFlag flag);

// Set second detonation flag
virtual void setDeton2Flag(DtLE_DetonField2Flag flag);

// Returns the offset (in bytes) of a field from variable data
region.
// Some fields have subfields which are optional, so there are
multiple
virtual int detonFlagFieldOffset() const;
virtual int deton2FlagFieldOffset() const;
virtual int targetEntityID_Offset() const;
virtual int munitionID_Offset1() const;
virtual int munitionID_Offset2() const;
virtual int eventID_Offset1() const;
virtual int eventID_Offset2() const;
virtual int locationWorldOffset() const;
virtual int velocityOffset() const;
virtual int munitionOrientationOffset() const;
virtual int burstOffset1() const;
virtual int burstOffset2() const;
virtual int burstOffset3() const;
virtual int locationEntityOffset() const;
virtual int resultOffset() const;

protected:

// Size of PDU with zero parameter records.
static int theStaticSize;

protected:

// Temporary objects for reference returns
DtLeEntityID myEntityID;
DtLeEntityLinearVelocity myVelocity;
DtEntityType myEntityType;
};
#endif
#endif
```

leEntityID.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"

#include "packets/leTspiP.h"

// class DtLeEntityID:
//
// DtLeEntityID is a class which contains the entity identification
// information for the live entity PDUs.

class DT_DLL_VL DtLeEntityID
{
public:

    // Constructors
    DtLeEntityID();
    DtLeEntityID(const DtLeEntityID& orig);

    // Assignment operator
    DtLeEntityID& operator=(const DtLeEntityID& orig);
    DtLeEntityID& operator=(const DtNetLeEntityID& orig);

    // Destructor
    virtual ~DtLeEntityID();

    // Set/Get site
    virtual void setSite(DtU8 value);
    virtual DtU8 site() const;

    // Set/Get application
    virtual void setApplication(DtU8 value);
    virtual DtU8 application() const;

    // Set/Get entity
    virtual void setEntity(DtU16 value);
    virtual DtU16 entity() const;

    // Print the information contained in the record
    virtual void print() const;

    // Load/Unload data from a packet record
    virtual void unloadNetRecord(const DtNetLeEntityID& record);
    virtual void loadNetRecord(DtNetLeEntityID& record) const;

```

protected:

 // Data members:

DtU8	mySite;
DtU8	myApplication;
DtU16	myEntity;

};

#endif

#endif

leEntityOri.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"

#include "packets/leTspiP.h"

// class DtLeEntityOrientation:
//
// DtLeEntityOrientation is a class which contains the entity
// orientation information for the live entity PDUs. This record
// uses
// 8-bit values for the TSPI PDU and 16-bit values for the detona-
// tion
// PDU.

class DT_DLL_VL DtLeEntityOrientation
{
public:

    // Constructors
    DtLeEntityOrientation();
    DtLeEntityOrientation(const DtLeEntityOrientation& orig);

    // Assignment operator
    DtLeEntityOrientation& operator=(
        const DtLeEntityOrientation& orig);
    DtLeEntityOrientation& operator=(
        const DtNetLeEntityOrientation& orig);

    // Destructor
    virtual ~DtLeEntityOrientation();

    // Set/Get Psi
    virtual void setPsi(DtU16 value);
    virtual DtU16 psi() const;

    // Set/Get Theta
    virtual void setTheta(DtU16 value);
    virtual DtU16 theta() const;

    // Set/Get Phi
    virtual void setPhi(DtU16 value);
    virtual DtU16 phi() const;

```

```
// Print the information contained in the record
virtual void print() const;

// Load/Unload data from a packet record (for 8-bit fields)
virtual void unloadNetRecord(
    const DtNetLeEntityOrientation& record);
virtual void loadNetRecord(
    DtNetLeEntityOrientation& record) const;

// Load/Unload data from a packet record (for 16-bit fields)
virtual void unloadNetRecord16(
    const DtNetLeEntityOrientation16& record);
virtual void loadNetRecord16(
    DtNetLeEntityOrientation16& record) const;

protected:

    // Data members:
    DtU16                myPsi;
    DtU16                myTheta;
    DtU16                myPhi;

};
#endif
#endif
```

leEntityVel.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"

#include "packets/leTspiP.h"

// class DtLeEntityLinearVelocity:
//
// DtLeEntityLinearVelocity is a class which contains the entity
// linear
// velocity information for the live entity PDUs.

class DT_DLL_VL DtLeEntityLinearVelocity
{
public:

    // Constructors
    DtLeEntityLinearVelocity();
    DtLeEntityLinearVelocity(const DtLeEntityLinearVelocity& orig);

    // Assignment operator
    DtLeEntityLinearVelocity& operator=(
        const DtLeEntityLinearVelocity& orig);
    DtLeEntityLinearVelocity& operator=(
        const DtNetLeEntityLinearVelocity& orig);

    // Destructor
    virtual ~DtLeEntityLinearVelocity();

    // Set/Get X
    virtual void setX(DtU16 value);
    virtual DtU16 x() const;

    // Set/Get Y
    virtual void setY(DtU16 value);
    virtual DtU16 y() const;

    // Set/Get Z
    virtual void setZ(DtU16 value);
    virtual DtU16 z() const;

    // Print the information contained in the record
    virtual void print() const;

```

```
// Load/Unload data from a packet record
virtual void unloadNetRecord(
    const DtNetLeEntityLinearVelocity& record);
virtual void loadNetRecord(
    DtNetLeEntityLinearVelocity& record) const;

protected:

    // Data members:
    DtU16                myX;
    DtU16                myY;
    DtU16                myZ;

};
#endif
#endif
```

leFirePdu.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "hostStructs.h"
#include "pdu.h"
#include "mathTypes.h"
#include "exerciseConn.h"

#include "packets/leFireP.h"

#include "leDeadReck.h"
#include "leEntityID.h"
#include "leLocWorld.h"
#include "leEntityVel.h"

// class DtLeFirePdu:

// Instances of DtLeFirePdu are used to send firing information for
// live
// entities.

// DtLeFirePdu callback function signature.
typedef void (*DtLeFirePduCb)(DtLeFirePdu* pdu, void* usr);

class DT_DLL_VL DtLeFirePdu : public DtPdu
{
public:

    // Constructors
    DtLeFirePdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtLeFirePdu(const DtNetLeFirePdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtLeFirePdu(const DtLeFirePdu& orig);

    // destructor
    virtual ~DtLeFirePdu();

    // Assignment operator
    DtLeFirePdu& operator=(const DtLeFirePdu& orig);

    // Set/Get the sub-protocol number
    virtual void setSubProtocol(DtSubProtocol value);
    virtual DtSubProtocol subProtocol() const;

    // Set/Get firing entity ID

```

```
virtual void setFiringEntityID(const DtLeEntityID& id);
virtual const DtLeEntityID& firingEntityID() const;

// Get flags field (this field is automatically maintained by the
// PDU as other PDU fields are set)
virtual DtU8 fireFlagsField() const;

// Get the value for a specific flag from the flags field
virtual DtBoolean fireFlag(DtLE_FireFieldFlag flag) const;

// Some of following accessor functions refer to records which
are optional.
// The set methods create the record in the PDU if it does not
already
// exist. The get methods return DtFalse if the record does not
exist.

// Set/Get target entity ID (optional)
virtual void setTargetEntityID(const DtLeEntityID& id);
virtual DtBoolean getTargetEntityID(DtLeEntityID& id) const;

// Set/Get munition ID site (optional)
virtual void setMunitionID_Site(DtU8 value);
virtual DtBoolean getMunitionID_Site(DtU8& value) const;

// Set/Get munition ID application (optional)
virtual void setMunitionID_Application(DtU8 value);
virtual DtBoolean getMunitionID_Application(DtU8& value) const;

// Set/Get munition ID entity (optional)
virtual void setMunitionID_Entity(DtU16 value);
virtual DtBoolean getMunitionID_Entity(DtU16& value) const;

// Set/Get event ID site (optional)
virtual void setEventID_Site(DtU8 value);
virtual DtBoolean getEventID_Site(DtU8& value) const;

// Set/Get event ID application (optional)
virtual void setEventID_Application(DtU8 value);
virtual DtBoolean getEventID_Application(DtU8& value) const;

// Set/Get event ID event
virtual void setEventID_Event(DtU16 value);
virtual DtU16 eventID_Event() const;

// Set/Get entity location (optional)
virtual void setLocation(const DtLeEntityLocationWorldCoordi-
nates& value);
virtual DtBoolean getLocation(
    DtLeEntityLocationWorldCoordinates& value) const;
```

```

// Set/Get burst munition type
virtual void setBurstMunitionType(DtEntityType& type);
virtual const DtEntityType& burstMunitionType() const;

// Set/Get burst warhead (optional)
virtual void setBurstWarhead(DtU16 value);
virtual DtBoolean getBurstWarhead(DtU16& value) const;

// Set/Get burst fuse (optional)
virtual void setBurstFuse(DtU16 value);
virtual DtBoolean getBurstFuse(DtU16& value) const;

// Set/Get burst quantity (optional)
virtual void setBurstQuantity(DtU16 value);
virtual DtBoolean getBurstQuantity(DtU16& value) const;

// Set/Get burst rate (optional)
virtual void setBurstRate(DtU16 value);
virtual DtBoolean getBurstRate(DtU16& value) const;

// Set/Get entity linear velocity
virtual void setVelocity(const DtLeEntityLinearVelocity& value);
virtual const DtLeEntityLinearVelocity& velocity() const;

// Set/Get range
virtual void setRange(DtU16 value);
virtual DtU16 range() const;

// Print the data contained by this PDU
virtual void printData() const;

// Print the header (override to include the new sub-protocol
field)
virtual void printHeader() const;

public:

// Create a DtLeFirePdu.
// Caller is responsible for deletion.
static DtPdu* create(const DtNetPacket* initial,
    DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

// Add (register)/Remove (deregister) function to be called when
// pdu is received.
static void addCallback(DtExerciseConn* conn,
    DtLeFirePduCb cb, void* usr);
static void removeCallback(DtExerciseConn* conn,
    DtLeFirePduCb cb, void* usr);

```

protected:

```
// Returns the internal PDU kind
virtual DtPduKind internalGetPduKind() const;

// Casts the netPdu element to our network struct
virtual DtNetLeFirePdu* netPduStruct() const;

// Casts the 'this' pointer for changing elements in a const
function
// Non-virtual since you can't override based on return type
DtLeFirePdu* self() const;

// Returns a pointer to the first byte of variable length data
virtual char* var() const;

// Set Fire Flags
virtual void setFireFlagsField(DtU8 value);

// Set a flag of the flag field
virtual void setFireFlag(DtLE_FireFieldFlag flag);

// Clear the flag and all following flags in the fire flags field
virtual void clearFireFlags(DtLE_FireFieldFlag flag);

// Returns the offset (in bytes) of a field from variable data
region.
// Some fields have subfields which are optional, so there are
multiple
virtual int fireFlagFieldOffset() const;
virtual int targetEntityID_Offset() const;
virtual int munitionID_Offset1() const;
virtual int munitionID_Offset2() const;
virtual int eventID_Offset1() const;
virtual int eventID_Offset2() const;
virtual int locationOffset() const;
virtual int burstOffset1() const;
virtual int burstOffset2() const;
virtual int burstOffset3() const;
virtual int velocityOffset() const;
virtual int rangeOffset() const;
```

protected:

```
// Size of PDU with zero parameter records.
static int theStaticSize;
```

protected:

```
// Temporary objects for reference returns
```



```
DtLeEntityID myEntityID;  
DtLeEntityLinearVelocity myVelocity;  
DtEntityType myEntityType;  
  
};  
#endif  
#endif
```

leLocEntity.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"

#include "packets/leTspiP.h"

// class DtLeEntityLocationEntityCoordinates:
//
// DtLeEntityLocationEntityCoordinates is a class which contains the
// entity
// location information (in entity coordinates) for the live entity
// PDUs.

class DT_DLL_VL DtLeEntityLocationEntityCoordinates
{
public:

    // Constructors
    DtLeEntityLocationEntityCoordinates();
    DtLeEntityLocationEntityCoordinates(
        const DtLeEntityLocationEntityCoordinates& orig);

    // Assignment operator
    DtLeEntityLocationEntityCoordinates& operator=(
        const DtLeEntityLocationEntityCoordinates& orig);
    DtLeEntityLocationEntityCoordinates& operator=(
        const DtNetLeEntityLocationEntityCoordinates& orig);

    // Destructor
    virtual ~DtLeEntityLocationEntityCoordinates();

    // Set/Get X
    virtual void setX(DtU16 value);
    virtual DtU16 x() const;

    // Set/Get Y
    virtual void setY(DtU16 value);
    virtual DtU16 y() const;

    // Set/Get Z
    virtual void setZ(DtU16 value);
    virtual DtU16 z() const;

    // Print the information contained in the record
```

```
virtual void print() const;

// Load/Unload data from a packet record
virtual void unloadNetRecord(
    const DtNetLeEntityLocationEntityCoordinates& record);
virtual void loadNetRecord(
    DtNetLeEntityLocationEntityCoordinates& record) const;

protected:

    // Data members:
    DtU16          myX;
    DtU16          myY;
    DtU16          myZ;

};
#endif
#endif
```

leLocWorld.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"

#include "packets/leTspiP.h"

// class DtLeEntityLocationWorldCoordinates:
//
// DtLeEntityLocationWorldCoordinates is a class which contains the
// entity
// location information for the live entity PDUs.

class DT_DLL_VL DtLeEntityLocationWorldCoordinates
{
public:

    // Constructors
    DtLeEntityLocationWorldCoordinates();
    DtLeEntityLocationWorldCoordinates(
        const DtLeEntityLocationWorldCoordinates& orig);

    // Assignment operator
    DtLeEntityLocationWorldCoordinates& operator=(
        const DtLeEntityLocationWorldCoordinates& orig);
    DtLeEntityLocationWorldCoordinates& operator=(
        const DtNetLeEntityLocationWorldCoordinates& orig);

    // Destructor
    virtual ~DtLeEntityLocationWorldCoordinates();

    // Set/Get the reference point
    virtual void setReferencePoint(DtU16 value);
    virtual DtU16 referencePoint() const;

    // Set/Get delta X
    virtual void setDeltaX(DtU16 value);
    virtual DtU16 deltaX() const;

    // Set/Get delta Y
    virtual void setDeltaY(DtU16 value);
    virtual DtU16 deltaY() const;

    // Set/Get delta Z
    virtual void setDeltaZ(DtU16 value);
```

```
virtual DtU16 deltaZ() const;

// Print the information contained in the record
virtual void print() const;

// Load/Unload data from a packet record
virtual void unloadNetRecord(
    const DtNetLeEntityLocationWorldCoordinates& record);
virtual void loadNetRecord(
    DtNetLeEntityLocationWorldCoordinates& record) const;

protected:

    // Data members:
    DtU16                myReferencePoint;
    DtU16                myDeltaX;
    DtU16                myDeltaY;
    DtU16                myDeltaZ;

};
#endif
#endif
```

leOriError.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"

#include "packets/leTspiP.h"

// class DtLeOrientationError:
//
// DtLeOrientationError is a class which contains the entity
// orientation error information for the live entity PDUs.

class DT_DLL_VL DtLeOrientationError
{
public:

    // Constructors
    DtLeOrientationError();
    DtLeOrientationError(const DtLeOrientationError& orig);

    // Assignment operator
    DtLeOrientationError& operator=(
        const DtLeOrientationError& orig);
    DtLeOrientationError& operator=(
        const DtNetLeOrientationError& orig);

    // Destructor
    virtual ~DtLeOrientationError();

    // Set/Get azimuth error
    virtual void setAzimuthError(DtU16 value);
    virtual DtU16 azimuthError() const;

    // Set/Get elevation error
    virtual void setElevationError(DtU16 value);
    virtual DtU16 elevationError() const;

    // Set/Get rotation error
    virtual void setRotationError(DtU16 value);
    virtual DtU16 rotationError() const;

    // Print the information contained in the record
    virtual void print() const;

    // Load/Unload data from a packet record
```

```
virtual void unloadNetRecord(  
    const DtNetLeOrientationError& record);  
virtual void loadNetRecord(  
    DtNetLeOrientationError& record) const;  
  
protected:  
  
    // Data members:  
    DtU16                myAzimuthError;  
    DtU16                myElevationError;  
    DtU16                myRotationError;  
  
};  
#endif  
#endif
```

lePosError.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"

#include "packets/leTspiP.h"

// class DtLePositionError:
//
// DtLePositionError is a class which contains the entity location
// information for the live entity PDUs.

class DT_DLL_VL DtLePositionError
{
public:

    // Constructors
    DtLePositionError();
    DtLePositionError(const DtLePositionError& orig);

    // Assignment operator
    DtLePositionError& operator=(
        const DtLePositionError& orig);
    DtLePositionError& operator=(
        const DtNetLePositionError& orig);

    // Destructor
    virtual ~DtLePositionError();

    // Set/Get horizontal error
    virtual void setHorizontalError(DtU16 value);
    virtual DtU16 horizontalError() const;

    // Set/Get vertical error
    virtual void setVerticalError(DtU16 value);
    virtual DtU16 verticalError() const;

    // Print the information contained in the record
    virtual void print() const;

    // Load/Unload data from a packet record
    virtual void unloadNetRecord(
        const DtNetLePositionError& record);
    virtual void loadNetRecord(
        DtNetLePositionError& record) const;
```


protected:

```
    // Data members:
    DtU16          myHorizontalError;
    DtU16          myVerticalError;

};
#endif
#endif
```

leTspiPdu.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "hostStructs.h"
#include "pdu.h"
#include "mathTypes.h"
#include "exerciseConn.h"

#include "packets/leTspiP.h"

#include "leDeadReck.h"
#include "leEntityID.h"
#include "leLocWorld.h"
#include "leEntityOri.h"
#include "leEntityVel.h"
#include "leOriError.h"
#include "lePosError.h"

// class DtLeTspiPdu:
//
// Instances of DtLeTspiPdu are used to send state information for
// live
// entities.

// DtLeTspiPdu callback function signature.
typedef void (*DtLeTspiPduCb)(DtLeTspiPdu* pdu, void* usr);

class DT_DLL_VL DtLeTspiPdu : public DtPdu
{
public:

    // Constructors
    DtLeTspiPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtLeTspiPdu(const DtNetLeTspiPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtLeTspiPdu(const DtLeTspiPdu& orig);

    // destructor
    virtual ~DtLeTspiPdu();

    // Assignment operator
    DtLeTspiPdu& operator=(const DtLeTspiPdu& orig);

    // Set/Get the sub-protocol number
    virtual void setSubProtocol(DtSubProtocol value);
```

```
virtual DtSubProtocol subProtocol() const;

// Set/Get entity ID
virtual void setEntityID(const DtLeEntityID& id);
virtual const DtLeEntityID& entityID() const;

// Get TSPI Flag field (this field is automatically maintained by
the
// PDU as other PDU fields are set)
virtual DtU8 tspiField() const;

// Get the value for a specific TSPI flag from the flag field
virtual DtBoolean tspiFlag(DtLE_TSPI_FieldFlag flag) const;

// Set/Get entity location
virtual void setLocation(const DtLeEntityLocationWorldCoordinates& value);
virtual const DtLeEntityLocationWorldCoordinates& location()
const;

// The following accessor functions refer to records which are
optional.
// The set methods create the record in the PDU if it does not
already
// exist. The get methods return DtFalse if the record does not
exist.

// Set/Get entity linear velocity
virtual void setVelocity(const DtLeEntityLinearVelocity& value);
virtual DtBoolean getVelocity(DtLeEntityLinearVelocity& value)
const;

// Set/Get entity orientation
virtual void setOrientation(const DtLeEntityOrientation& value);
virtual DtBoolean getOrientation(DtLeEntityOrientation& value)
const;

// Set/Get position error
virtual void setPositionError(const DtLePositionError& value);
virtual DtBoolean getPositionError(DtLePositionError& value)
const;

// Set/Get orientation error
virtual void setOrientationError(const DtLeOrientationError&
value);
virtual DtBoolean getOrientationError(DtLeOrientationError&
value) const;

// Set/Get dead reckoning parameters
virtual void setDeadReckoningParameters(
```

```
    const DtLeDeadReckoningParameters& value);
virtual DtBoolean getDeadReckoningParameters(
    DtLeDeadReckoningParameters& value) const;

// Set/Get measured speed
virtual void setMeasuredSpeed(DtU16 value);
virtual DtBoolean getMeasuredSpeed(DtU16& value) const;

// Set/Get the system specific data length
virtual DtBoolean setSystemSpecificDataLength(DtU8 value);
virtual DtU8 systemSpecificDataLength() const;

// Set/Get the system specific data (will copy systemSpecific-
DataLength()
// bytes on set/get). Returns DtFalse if the length is zero.
virtual DtBoolean setSystemSpecificData(DtU8* dataPtr);
virtual DtBoolean getSystemSpecificData(DtU8* dataPtr);

// Print the data contained by this PDU
virtual void printData() const;

// Print the header (override to include the new sub-protocol
field)
virtual void printHeader() const;

public:

    // Create a DtLeTspiPdu.
    // Caller is responsible for deletion.
    static DtPdu* create(const DtNetPacket* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // Add (register)/Remove (deregister) function to be called when
    // pdu is received.
    static void addCallback(DtExerciseConn* conn,
        DtLeTspiPduCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtLeTspiPduCb cb, void* usr);

protected:

    // Returns the internal PDU kind
    virtual DtPduKind internalGetPduKind() const;

    // Casts the netPdu element to our network struct
    virtual DtNetLeTspiPdu* netPduStruct() const;

    // Casts the 'this' pointer for changing elements in a const
    function
    // Non-virtual since you can't override based on return type
```

```

    DtLeTspiPdu* self() const;

    // Returns a pointer to the first byte of variable length data
    virtual char* var() const;

    // Set TSPI Flags
    virtual void setTspiField(DtU8 value);

    // Set a flag of the TSPI field
    virtual void setTspiFlag(DtLE_TSPI_FieldFlag flag);

    // Clear the flag and all following flags in the TSPI field
    virtual void clearTspiFlags(DtLE_TSPI_FieldFlag flag);

    // Returns the offset (in bytes) of a field from variable data
    region.
    virtual int tspiFlagOffset() const;
    virtual int locationOffset() const;
    virtual int velocityOffset() const;
    virtual int orientationOffset() const;
    virtual int positionErrorOffset() const;
    virtual int orientationErrorOffset() const;
    virtual int deadReckoningParametersOffset() const;
    virtual int measuredSpeedOffset() const;
    virtual int systemSpecificOffset() const;

protected:

    // Size of PDU with zero parameter records.
    static int theStaticSize;

protected:

    // Temporary objects for reference returns
    DtLeEntityID myEntityID;
    DtLeEntityLocationWorldCoordinates myLocation;

};
#endif
#endif

```

LibMatrix.h

```
#include <math.h>

#include "MachineTypes.h"
#include "mathTypes.h"

#define DtRad2Deg(x)      (((x) / M_PI) * 180.0)
#define DtDeg2Rad(x)      (((x) * M_PI) / 180.0)

/* Returns the absolute value of x.      */
#define DtAbs(x)          (((x) >= 0) ? (x) : (-(x)))

/* Returns the sign of x.      */
#define DtSgn(x)          (((x) >= 0) ? 1 : -1)

/* Returns x if greater than zero.      */
#define DtPos(x)          (((x) >= 0) ? (x) : 0)

/* Returns x if x is less than zero.      */
#define DtNeg(x)          (((x) >= 0) ? 0 : (x))

/* Returns the square of x.      */
#define DtSquare(x)       ((x) * (x))

/* A selection of constants useful as tolerance and threshold
levels, etc. */
#define DtEPico            1.0E-12
#define DtENano            1.0E-09
#define DtEMicro           1.0E-06
#define DtEMilli           1.0E-03
#define DtECenti           1.0E-02
#define DtEDeci            1.0E-01
#define DtEHalf            5.0E-01
#define DtENineTenths      9.0E-01
#define DtEOne             1.0
#define DtEDeca            1.0E01
#define DtEHecto           1.0E02
#define DtEKilo            1.0E03
#define DtEMega            1.0E06
#define DtEGiga            1.0E09

/* A macro to make use of these tolerances. */
#define DtEq(x, y, e)      (((DtAbs((x) - (y)) <= (e)) ? DtTrue :
DtFalse)

/*
 * Vector routines.
 */
```

```

#define DtSetVec(v, x, y, z)v[DtX]=x;v[DtY]=y;v[DtZ]=z;

DT_DLL_MATRIX voidDtVecInit(DtVectorRef v);/* Initialize to all
zeros. */
DT_DLL_MATRIX voidDtVecCopy(DtConstVector from, DtVectorRef to);
DT_DLL_MATRIX voidDtVecNormalize(DtConstVector v, DtVectorRef
result);
DT_DLL_MATRIX intDtVecCheck(DtConstVector v);/* Is this vector
normalized? */
DT_DLL_MATRIX voidDtVecScale(DtConstVector v, DtReal scale_factor,
DtVectorRef result);
DT_DLL_MATRIX voidDtVecNeg(DtConstVector v1, DtVectorRef result);
/* result = -vec */
DT_DLL_MATRIX voidDtVecAdd(DtConstVector v1, DtConstVector v2,
DtVectorRef result);
DT_DLL_MATRIX voidDtVecSub(DtConstVector v1, DtConstVector v2,
DtVectorRef result); /* result = v1 - v2 */
DT_DLL_MATRIX DtRealDtVecCosProd(DtConstVector v1, DtConstVector
v2); /* cos of angle between vecs */
DT_DLL_MATRIX DtRealDtVecDotProd(DtConstVector v1, DtConstVector
v2);
DT_DLL_MATRIX voidDtVecDump(const char *str, DtConstVector v);
DT_DLL_MATRIX voidDtVecCrossProd(DtConstVector v1, DtConstVector
v2, DtVectorRef result);/* sweep v1 to v2 */
DT_DLL_MATRIX DtReal DtVecBoxProd(DtConstVector v1, DtConstVector
v2, DtConstVector v3);/* sweep v1 to v2 to v3*/

/*
 * Direction Cosine Matrix routines.
 */

DT_DLL_MATRIX double DtDcmDcmRotAngle(DtConstDcm a, DtConstDcm b);
DT_DLL_MATRIX void DtDcmDcmRotAxis(DtConstDcm a2ref, DtConstDcm
b2ref, DtVectorRef n /* a2b */);
DT_DLL_MATRIX voidDtDcmInit(DtDcmRef m);/* Initialize to all zeros.
*/
DT_DLL_MATRIX voidDtDcmIdentInit(DtDcmRef m);
DT_DLL_MATRIX intDtDcmCheck(DtConstDcm m);/* Is this matrix
orthonormal? */
DT_DLL_MATRIX voidDtDcmFixMatrix(DtDcmRef mat, int axis); /* make
orthonormal, pick most important. axis */
DT_DLL_MATRIX voidDtDcmCopy(DtConstDcm src, DtDcmRef dest);
DT_DLL_MATRIX DtRealDtDcmDeterminant(DtConstDcm m);
DT_DLL_MATRIX voidDtDcmAdd(DtConstDcm m1, DtConstDcm m2, DtDcmRef
result); /* result = m1 + m2 */
DT_DLL_MATRIX voidDtDcmSub(DtConstDcm m1, DtConstDcm m2, DtDcmRef
result); /* result = m1 - m2 */
DT_DLL_MATRIX voidDtDcmDcmMul(DtConstDcm m1, DtConstDcm m2, DtDc-
mRef result); /* result = m1 x m2 */

```

```
DT_DLL_MATRIX voidDtDcmTranspose(DtConstDcm m, DtDcmRef result);
DT_DLL_MATRIX voidDtDcmAdjugate(DtConstDcm m, DtDcmRef result);
DT_DLL_MATRIX voidDtDcmInverse(DtConstDcm m, DtDcmRef result);
DT_DLL_MATRIX voidDtDcmScale(DtConstDcm mat, DtReal scale_factor,
DtDcmRef result);
DT_DLL_MATRIX voidDtDcmDump(const char *str, DtConstDcm mat);

/* Rotation is counterclockwise when viewed a positive axis. */
DT_DLL_MATRIX voidDtDcmRotInit(DtDcmRef m, DtReal theta, int
rot_axis);
DT_DLL_MATRIX voidDtDcmRotInit2(DtDcmRef m, DtReal sin_theta, int
rot_axis); /* $$$ only pos rots? */
DT_DLL_MATRIX voidDtDcmTrigInit(DtDcmRef m, DtReal sin_theta,
DtReal cos_theta, int rot_axis);
DT_DLL_MATRIX voidDtDcmForm(DtReal *axis, DtReal angle, DtDcmRef C,
DtReal *sin_ang, DtReal *cos_ang);
DT_DLL_MATRIX voidDtDcmLevelInit (DtDcmRef m, DtConstVector v); /* y
axis given, x axis parallel to ground. */

/*
 * Combined vector/matrix routines.
 */

DT_DLL_MATRIX voidDtDcmVecMul(DtConstDcm m, DtConstVector v, DtVec-
torRef result);
DT_DLL_MATRIX voidDtVecDcmMul(DtConstVector v, DtConstDcm m, DtVec-
torRef result);

/*
 * Euler parameter routines.
 */

typedef DtReal DtEParm[4];

extern DT_DLL_MATRIX voidDtElrCopy (DtEParm from, DtEParm to);
extern DT_DLL_MATRIX voidDtElrElrCat (DtEParm A_e_B0, DtEParm
B0_e_B, DtEParm A_e_B);
extern DT_DLL_MATRIX voidDtElrForm (DtEParm axis, DtReal angle,
DtEParm result);
extern DT_DLL_MATRIX voidDtElrIdent (DtEParm e);
extern DT_DLL_MATRIX voidDtElrToMat (DtEParm A_e_B, DtDcmRef
A_c_B);
extern DT_DLL_MATRIX voidDtElrTranspose (DtEParm e, DtEParm
result);
extern DT_DLL_MATRIX voidDtDcmToElr (DtConstDcm C, DtEParm e);
extern DT_DLL_MATRIX voidDtVecElrTransform (DtEParm v_A, DtEParm
A_to_B, DtEParm v_B);

/*
 * Quaternion routines.
```



```

*/

typedef struct
{
    DtReal DtQ0;
    DtReal DtQ1;
    DtReal DtQ2;
    DtReal DtQ3;
} DtQuaternion;

DT_DLL_MATRIX voidDtQuatQuatMul(DtQuaternion *qresult, DtQuaternion *DtQ1, DtQuaternion *DtQ2);
DT_DLL_MATRIX voidDtDcmFromQuaternion(DtDcmRef m, DtQuaternion *q);
/* voidDtQuatScale(DtQuaternion *qresult, DtQuaternion *q, DtReal
delta_t); */
DT_DLL_MATRIX voidDtQuaternionDump(DtQuaternion *q);

DT_DLL_MATRIX voidDtQuaternionFromTaitBryan(DtQuaternion *q, const
DtTaitBryan &tb);
DT_DLL_MATRIX voidDtBodyToMat(DtBodyRates av, DtConstDcm HcW, DtDcmRef cDot);
DT_DLL_MATRIX voidDtTbToMat(const DtTaitBryan &tb, DtDcmRef HcW);

/*
 * Periodic modulus
 */

/* For x with period ABS(incl-excl),
 * return DtModPerLo(x, incl, excl) if incl < excl
 * return DtModPerHi(x, excl, incl) if incl > excl */
DT_DLL_MATRIX double DtModPer(double x, double incl, double excl);

/* For x with period per, return 0 <= x < per */
DT_DLL_MATRIX double DtModPerZero(double x, double per);

/* For x with period (max-min), return min <= x < max */
DT_DLL_MATRIX double DtModPerLo(double x, double min, double max);

/* For x with period (max-min), return min < x <= max */
DT_DLL_MATRIX double DtModPerHi(double x, double min, double max);

/* directed rotation rates */

DT_DLL_MATRIX DtDirectedRotRate DtDirectRotRate(DtConstVector pqr);
DT_DLL_MATRIX void DtMakeRotationDcm(DtDirectedRotRate drr, double
dt, DtDcmRef m);

/* Returns the trace of the rotation matrix, R, between between
   matricies A and B. If A is a DCM between frame body and world,
   and

```

```
    B is that same frame rotated through angle theta about an arbitrary
    axis, then trace of R [where B = A * R, or R = inv(A) * B] is
    2 * cos(theta) + 1. */
DT_DLL_MATRIX double DtDcmDcmRotTrace(DtConstDcm a, DtConstDcm b);

/* Generic DcmDcm Multiply routine */

enum generic_mm_multflag
{
    TRANSPPOSE_NONE = 0,
    TRANSPPOSE_M1 = 1,
    TRANSPPOSE_M2 = 2,
    TRANSPPOSE_M1_AND_M2 = 3
};
DT_DLL_MATRIX DtReal DtMM_Mult(DtConstDcm m1, DtConstDcm m2, DtDcmRef
result, generic_mm_multflag flag=TRANSPPOSE_NONE);

/* Generic Vector Sum :Ans=A+s*B*/
#define ADD 1.0
#define SUBTRACT -1.0
DT_DLL_MATRIX void DtSum(DtConstVector a, DtConstVector b, DtReal
s, DtVectorRef ans);

/* generic initialize matrix routines */
#define ZERO 0
#define IDENTITY 1;
DT_DLL_MATRIX void DtM_Init(DtConstDcm input, DtDcmRef result, int
flag);

DT_DLL_MATRIX void DtSetSym(DtDcmRef v, DtReal aa, DtReal ab, DtReal
ac, DtReal bb, DtReal bc, DtReal cc);

DT_DLL_MATRIX void DtVV_Product(DtConstVector a, DtConstVector b,
DtDcmRef ans);

DT_DLL_MATRIX void DtCreateCross(DtConstVector r, DtDcmRef ans);

DT_DLL_MATRIX void DtM(DtConstVector v, DtDcmRef m);

DT_DLL_MATRIX void DtRotate(DtDcmRef ans, DtConstVector direc-
tion, double theta);

DT_DLL_MATRIX double DtVecInfNorm(DtConstVector v1);

/* Unobvious Constants */
typedef DtReal DtThreeIndex[3][3][3];
typedef DtReal (*DtThreeIndexPtr)[3][3];

extern DT_DLL_MATRIX DtThreeIndexDtLevi_Civita;
```

```

/* additional Routines */
DT_DLL_MATRIX double DtDistance(DtConstVector p1, DtConstVector p2);
/* ||p1-p2||_{2} */
DT_DLL_MATRIX double DtDistSqr(DtConstVector p1, DtConstVector p2);

DT_DLL_MATRIX double DtMax(double x, double y);
DT_DLL_MATRIX double DtMin(double x, double y);
DT_DLL_MATRIX double DtLimit(double x, double hi, double lo);

class DT_DLL_MATRIX DtCoordTranslation
{
public:
    DtCoordTranslation();
    DtCoordTranslation(DtConstVector toOriginInFromCoord);

    void setOffset(DtConstVector toOriginInFromCoord);
    void setByInverse(DtCoordTranslation &to2from);

    void getOffset(DtVectorRef toOriginInFromCoord);
    void coordTrans(DtConstVector from, DtVectorRef to);

private:
    DtVector offset; /* "to" origin in base coords */
};

class DT_DLL_MATRIX DtCoordTransform
{
public:
    DtCoordTransform();
    DtCoordTransform(DtConstVector toOriginInFromCoord, DtConstDcm
from2to);
    void setOffset(DtConstVector toOriginInFromCoord);
    void setRotation(DtConstDcm from2to);
    void setByInverse(const DtCoordTransform &to2from);

    void getOffset(DtVectorRef toOriginInFromCoord) const;
    void getOrientation(DtDcmRef from_to) const;

    void coordTrans(DtConstVector from, DtVectorRef to) const;
    void vecTrans(DtConstVector from, DtVectorRef to) const;
    void eulerTrans(const DtTaitBryan &from, DtTaitBryan *to) const;

    /* these were protected, but it is often more efficient to set
these
    by passing them to another function */
    DtVector offset; /* "to" origin in base coords */
    DtDcm rotation;
};

```

linObjStPdu.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "hostStructs.h"
#include "pdu.h"
#include "mathTypes.h"
#include "exerciseConn.h"

#include "packets/linObjStP.h"

#include "linSegParam.h"
#include "objectType.h"

// class DtLinearObjectStatePdu:
//
// Instances of DtLinearObjectStatePdu are used to communicate
// information
// about the addition/modification of a synthetic object which is
// geometrically
// anchored to the terrain with one point.

// DtLinearObjectStatePdu callback function signature.
typedef void (*DtLinearObjectStatePduCb)(
    DtLinearObjectStatePdu* pdu, void* usr);

class DT_DLL_VL DtLinearObjectStatePdu : public DtPdu
{
public:
    // Constructors
    DtLinearObjectStatePdu(DtBufferPtr buffer =
DtUSE_INTERNAL_BUFFER);
    DtLinearObjectStatePdu(const DtNetLinearObjectStatePdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtLinearObjectStatePdu(const DtLinearObjectStatePdu& orig);

    // destructor
    virtual ~DtLinearObjectStatePdu();

    // Assignment operator
    DtLinearObjectStatePdu& operator=(const DtLinearObjectStatePdu&
orig);

    // Set/Get object ID
    virtual void setObjectID(const DtEntityIdentifier& id);
```

```

virtual const DtEntityIdentifier& objectID() const;

// Set/Get referenced object ID
virtual void setReferencedObjectID(const DtEntityIdentifier& id);
virtual const DtEntityIdentifier& referencedObjectID() const;

// Set/Get update number
virtual void setUpdateNumber(DtU16 field);
virtual DtU16 updateNumber() const;

// Set/Get the force ID
virtual void setForceID(DtForceType force);
virtual DtForceType forceID() const;

// Set/Get the number of segments
virtual DtBoolean setNumSegments(DtU8 num);
virtual DtU8 numSegments() const;

// Set/Get the requester ID
virtual void setRequesterID(const DtSimulationAddress& type);
virtual const DtSimulationAddress& requesterID() const;

// Set/Get the receiving ID
virtual void setReceivingID(const DtSimulationAddress& type);
virtual const DtSimulationAddress& receivingID() const;

// Set/Get the object type
virtual void setObjectType(const DtObjectType& type);
virtual const DtObjectType& objectType() const;

// Set/Get a linear segment parameter record. Returns DtFalse if
the
// index is out of range.
virtual DtBoolean setLinearSegmentParameter(int index,
const DtLinearSegmentParameter& init);
virtual DtBoolean getLinearSegmentParameter(int index,
DtLinearSegmentParameter& record) const;

// Print the data contained by this PDU
virtual void printData() const;

public:

// Create a DtLinearObjectStatePdu.
// Caller is responsible for deletion.
static DtPdu* create(const DtNetPacket* initial,
DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

// Add (register)/Remove (deregister) function to be called when
// pdu is received.

```

```
static void addCallback(DtExerciseConn* conn,
    DtLinearObjectStatePduCb cb, void* usr);
static void removeCallback(DtExerciseConn* conn,
    DtLinearObjectStatePduCb cb, void* usr);
```

protected:

```
// Returns the internal PDU kind
virtual DtPduKind internalGetPduKind() const;
```

```
// Casts the netPdu element to our network struct
virtual DtNetLinearObjectStatePdu* netPduStruct() const;
```

// Casts the 'this' pointer for changing elements in a const function

```
// Non-virtual since you can't override based on return type
DtLinearObjectStatePdu* self() const;
```

```
// Returns a pointer to the first byte of variable length data
virtual char* var() const;
```

protected:

```
// Size of PDU with zero parameter records.
static int theStaticSize;
```

protected:

```
// Temporary objects for reference returns
DtEntityIdentifier myEntityID;
DtObjectType myObjectType;
DtSimulationAddress mySimulationAddress;
```

```
};
#endif
#endif
```

linSegParam.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "mathTypes.h"
#include "MachineTypes.h"
#include "disEnums.h"

#include "packets/linObjStP.h"

// class DtLinearSegmentParameter:
//
// DtLinearSegmentParameter is a class which contains the linear
// segment
// parameters for the linear object state PDU.

class DT_DLL_VL DtLinearSegmentParameter
{
public:

    // Constructors
    DtLinearSegmentParameter();
    DtLinearSegmentParameter(const DtLinearSegmentParameter& orig);

    // Assignment operator
    DtLinearSegmentParameter& operator=(
        const DtLinearSegmentParameter& orig);
    DtLinearSegmentParameter& operator=(
        const DtNetLinearSegmentParameter& orig);

    // Destructor
    virtual ~DtLinearSegmentParameter();

    // Set/Get the segment number
    virtual void setSegmentNumber(DtU8 value);
    virtual DtU8 segmentNumber() const;

    // Set/Get the modifications (by 8-bit integer)
    virtual void setModifications(DtU8 value);
    virtual DtU8 modifications() const;

    // Set/Get the modifications (by field)
    virtual void setModificationsField(
        DtModificationField field, DtBoolean value);
    virtual DtBoolean modificationsField(
        DtModificationField field) const;

```

```
// Set/Get appearance, the lower 16 bits (as an integer)
virtual void setAppearance(DtU16 app);
virtual DtU16 appearance() const;

// Set/Get appearance by attribute, the lower 16 bits
virtual void setAppearanceAttribute(
    DtAppearAttribute attr, DtAppearAttrVal val);
virtual DtAppearAttrVal appearanceAttribute(
    DtAppearAttribute attr) const;

// Set/Get specific appearance, the upper 32 bits (as a 32 bit
integer)
virtual void setSpecificAppearance(DtU32 app);
virtual DtU32 specificAppearance() const;

// Set/Get specific appearance by attribute, the upper 32 bits
virtual void setSpecificAppearanceAttribute(
    DtAppearAttribute attr, DtAppearAttrVal val);
virtual DtAppearAttrVal specificAppearanceAttribute(
    DtAppearAttribute attr) const;

// Set/Get location
virtual void setLocation(DtConstVector coords);
virtual DtConstVector location() const;

// Set/Get orientation
virtual void setOrientation(const DtTaitBryan& orientation);
virtual const DtTaitBryan& orientation() const;

// Set/Get the segment length
virtual void setSegmentLength(DtU16 value);
virtual DtU16 segmentLength() const;

// Set/Get the segment width
virtual void setSegmentWidth(DtU16 value);
virtual DtU16 segmentWidth() const;

// Set/Get the segment height
virtual void setSegmentHeight(DtU16 value);
virtual DtU16 segmentHeight() const;

// Set/Get the segment depth
virtual void setSegmentDepth(DtU16 value);
virtual DtU16 segmentDepth() const;

// Print the information contained in the record
virtual void print() const;

// Load/Unload data from a packet record
virtual void unloadNetRecord(
```



```
        const DtNetLinearSegmentParameter& record);  
virtual void loadNetRecord(  
    DtNetLinearSegmentParameter& record) const;  
  
protected:  
  
    // Data members:  
    DtU8          mySegmentNumber;  
    DtU8          myModifications;  
    DtU32         mySpecificAppearance;  
    DtU16         myAppearance;  
    DtVector      myLocation;  
    DtTaitBryan   myOrientation;  
    DtU16         mySegmentLength;  
    DtU16         mySegmentWidth;  
    DtU16         mySegmentHeight;  
    DtU16         mySegmentDepth;  
};  
#endif  
#endif
```

locObjMgr.h

```
#if DtHLA

#include "hlaObj.h"
#include "objIdHashLst.h"

// class DtLocalHlaObjectManager:
//
// Instances of DtLocalHlaObjectManager are used to manage
// local HLA objects (objects which are being published by this
// application). This manager handles providing updates to the
// network.

class DT_DLL_VL DtLocalHlaObjectManager
{
public:

    // default constructor
    DtLocalHlaObjectManager();

    // destructor
    virtual ~DtLocalHlaObjectManager();

    // Get the list of all HLA objects.
    virtual const DtList& allHlaObjects() const;

    // Get the HLA object based on its identifier.
    virtual DtHlaObject* hlaObject(RTI::ObjectHandle id) const;

    // process RTI's provideUpdate service.
    virtual void processProvideUpdate(RTI::ObjectHandle id,
        const RTI::AttributeHandleSet& theAttributes);

    // add the object to the list of local objects. This enables it
    to
    // get RTI callbacks destined for it.
    virtual void manage(DtHlaObject* obj);

    // remove the object from the list of local objects. Indicates
    // that the object is no longer being published.
    virtual void unManage(DtHlaObject* obj);

protected:

    // Note: The copy constructor and assignment operator are
    protected
    // because there should only be one object manager per exercise
    // connection.

}
```

```
// These should not be called.

// copy constructor
DtLocalHlaObjectManager(const DtLocalHlaObjectManager& orig);

// assignment operator
DtLocalHlaObjectManager& operator=(const DtLocalHlaObjectManager&
orig);

protected:

    DtObjectIdHashList myObjectList;
};
#endif
```

MachineTypes.h

```
#ifndef _MachineTypes_h
#define _MachineTypes_h

#include "vlconfig.h"

/* defined(__cplusplus), not __cplusplus,
 * because __SUNPRO_CC defines it to be empty */
#if DT_HAVE_STD_H && defined(__cplusplus)
#include <std.h>
#endif

#include <sys/types.h>

typedef unsigned charDtU8;
typedef unsigned shortDtU16;

#if DT_HAVE_64BIT_LONGS
typedef unsigned intDtU32;
typedef intDtInt32;
#else
typedef unsigned longDtU32;
typedef long intDtInt32;
#endif

typedef charDtInt8;
typedef short intDtInt16;
typedef floatDtFloat32;
typedef doubleDtFloat64;
typedef DtFloat64DtReal; /* For convenience */

enum DtBoolean { DtFalse, DtTrue } ;

/* Windows requires extra work to export from DLLs */
#if DT_NEED_DLL_DECLARATIONS

#ifdef DT_DLL_MTL
#undef DT_DLL_MTL
#define DT_DLL_MTL      __declspec( dllexport )
#else
#define DT_DLL_MTL      __declspec( dllimport )
#endif

#ifdef DT_DLL_MATRIX
#undef DT_DLL_MATRIX
#define DT_DLL_MATRIX   __declspec( dllexport )
#else
#define DT_DLL_MATRIX   __declspec( dllimport )
#endif


```

```
#endif

#ifdef DT_DLL_VL
#undef DT_DLL_VL
#define DT_DLL_VL      __declspec( dllexport )
#else
#define DT_DLL_VL      __declspec( dllimport )
#endif

#ifdef DT_DLL_VLUTIL
#undef DT_DLL_VLUTIL
#define DT_DLL_VLUTIL  __declspec( dllexport )
#else
#define DT_DLL_VLUTIL  __declspec( dllimport )
#endif

/* Default -- Unix or Win32 static */

#define DT_DLL_MTL
#define DT_DLL_MATRIX
#define DT_DLL_VL
#define DT_DLL_VLUTIL
#endif

#ifdef WIN32
#ifdef __cplusplus
extern "C"
{
    DT_DLL_VLUTIL int getopt(int argc, char** argv, char* opts);
    extern DT_DLL_VLUTIL char* optarg;
    extern DT_DLL_VLUTIL int optind;
    extern DT_DLL_VLUTIL int opterr;
}
#endif //__cplusplus
#endif //WIN32
```

mathTypes.h

```
#include <math.h>
#ifndef SQR
#define SQR(x) ((x) * (x))
#endif
#ifndef MIN
#define MIN(x, y) ((x)<(y) ? (x) : (y))
#endif
#ifndef MAX
#define MAX(x, y) ((x)>(y) ? (x) : (y))
#endif
#ifndef LIMIT
#define LIMIT(low, x, high) MAX(MIN(x, high), low)
#endif

// For DtVector
#include "vlVector.h"

// For DtDcm
#include "dcm.h"

// For DtTaitBryan
#include "taitBryan.h"

// Angular rotation about body X, Y, Z axes
typedef DtVector DtBodyRates;

// For DtDirectedRotRate and DtBodyRates
#include "directRotRate.h"
```

messagePdu.h

```

#if !DtHLA

#include "hostStructs.h"
#include "pduWithData.h"
#include "datumIdType.h"

#if DtDISVERS <= 4

#define DtMessageStaticSize 32

class DT_DLL_VL DtMessagePdu;
typedef void (*DtMessagePduCb)(DtMessagePdu *pdu, void *usr);

class DT_DLL_VL DtMessagePdu : public DtSimanPdu
{
public:

/***** STANDARD CONSTRUCTORS *****/

    DtMessagePdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtMessagePdu(const DtNetMessagePdu *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtMessagePdu(const DtEntityIdentifier &sender,
        const DtEntityIdentifier &receiver,
        const char *message,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    static DtPdu *create(const DtNetPacket *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    static void addCallback(DtExerciseConn *conn,
        DtMessagePduCb cb, void *usr);
    static void removeCallback(DtExerciseConn *conn,
        DtMessagePduCb cb, void *usr);

/***** INSPECTOR FUNCTIONS *****/

    int numVarFields() const;
    DtDatumParam varDatumId(int index) const;
    int varDatumSizeBits(int index) const;
    int varDatumSizeBytes(int index) const;
    int paramIndex(DtDatumParam param, int *retcode = NULL) const;

/***** INSPECTING DATUM VALUES BY INDEX *****/

    DtFloat32 datumValFloat32(int index, int *retcode = NULL)
const;

```

```
DtFloat64      datumValFloat64(int index, int *retcode = NULL)
const;
unsigned long datumValUnsigned32(int index, int *retcode = NULL)
const;
long           datumValInt32(int index, int *retcode = NULL)
const;
const char*    datumValByteArray(int index) const;

/***** INSPECTING DATUM VALUES BY PARAM ID *****/

DtFloat32      datumValFloat32(DtDatumParam param,
                                int *retcode = NULL) const;
DtFloat64      datumValFloat64(DtDatumParam param,
                                int *retcode = NULL) const;
unsigned long datumValUnsigned32(DtDatumParam param,
                                int *retcode = NULL) const;
long           datumValInt32(DtDatumParam param,
                                int *retcode = NULL) const;
const char*    datumValByteArray(DtDatumParam param,
                                int *retcode = NULL) const;

/***** MUTATOR FUNCTIONS *****/

void initDataSection(const DtVarDatumSpec *specList);
void setNumVarFields(int newNum);
void setDatumParam(int index, DtDatumParam param, int *retcode =
NULL);
void setVarDataBytes(int index, int length, int *retcode =
NULL);
void setMessage(const char *message, int size);

/***** SETTING DATA VALUES BY INDEX *****/

void setDatumValFloat64(int index, DtFloat64 val, int *retcode =
NULL);
void setDatumValFloat32(int index, DtFloat32 val, int *retcode =
NULL);
void setDatumValInt32(int index, long val, int *retcode = NULL);
void setDatumValUnsigned32(int index, unsigned long val,
                            int *retcode = NULL);
void setDatumValByteArray(int index, const char *val,
                            int *retcode = NULL);

/***** SETTING DATA VALUES BY PARAM ID *****/

void setDatumValFloat64(DtDatumParam param, DtFloat64 val,
                        int *retcode = NULL);
void setDatumValFloat32(DtDatumParam param, DtFloat32 val,
                        int *retcode = NULL);
void setDatumValInt32(DtDatumParam param, long val,
```



```
        int *retcode = NULL);
void setDatumValUnsigned32(DtDatumParam param, unsigned long
val,
        int *retcode = NULL);
void setDatumValByteArray(DtDatumParam param, const char *val,
        int *retcode = NULL);

void printData() const;

protected:

    DtPduKind internalGetPduKind() const {return DtMessagePduKind;}
    DtNetVarDatum *nthVarDataRecord(int index) const;
    void *datumValPtr(int index) const;
};
#endif // DtDISVERS
#endif // !DtHLA
```

milEntDec.h

```
#if DtHLA

#include "physEntDec.h"

// class DtMilitaryEntityDecoder:
//
// Instances of DtMilitaryEntityDecoder are used to decode a
// military entity state message into a state repository in an HLA
// (as opposed to DIS) environment.

class DT_DLL_VL DtMilitaryEntityDecoder : public DtPhysicalEntityDecoder
{
public:

    // constructor
    DtMilitaryEntityDecoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtMilitaryEntityDecoder();

    // copy constructor
    DtMilitaryEntityDecoder(const DtMilitaryEntityDecoder& orig);

    // assignment operator
    DtMilitaryEntityDecoder& operator=(const DtMilitaryEntityDecoder&
orig);

public:

    // Create a DtMilitaryEntityDecoder.
    // User is responsible for deletion.
    static DtHlaStateDecoder* create(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

protected:

    // Add decoding functions for the attributes.
    // Note: Non-virtual since meant to be called from constructors
    to
    //      add decoders.
    void addMyDecoders();

    // static decode<AttributeName> member functions.
```


milEntEnc.h

```
#if DtHLA

#include "physEntEnc.h"
#include "entitySR.h"

// class DtMilitaryEntityEncoder:
//
// Instances of DtMilitaryEntityEncoder are used to encode a
// military entity's state repository into a state message in an
// HLA environment.

class DT_DLL_VL DtMilitaryEntityEncoder : public DtPhysicalEntityEncoder
{
public:

    // constructor
    DtMilitaryEntityEncoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtMilitaryEntityEncoder();

    // copy constructor
    DtMilitaryEntityEncoder(const DtMilitaryEntityEncoder& orig);

    // assignment operator
    DtMilitaryEntityEncoder& operator=(const DtMilitaryEntityEncoder&
orig);

public:

    // Create a DtMilitaryEntityEncoder.
    // Caller is responsible for deletion.
    static DtHlaStateEncoder* create(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

protected:

    // static need<Attribute> member functions.
    DtDECLARE_ENTITY_ATTR_CHECKER(AlternateEntityType);
    DtDECLARE_ENTITY_ATTR_CHECKER(CamouflageType);
    DtDECLARE_ENTITY_ATTR_CHECKER(FirePowerDisabled);
    DtDECLARE_ENTITY_ATTR_CHECKER(ForceID);
    DtDECLARE_ENTITY_ATTR_CHECKER(IsConcealed);
```

```

    // static decode<Attribute> member functions.
    DtDECLARE_ENTITY_ATTR_ENCODER(AlternateEntityType);
    DtDECLARE_ENTITY_ATTR_ENCODER(CamouflageType);
    DtDECLARE_ENTITY_ATTR_ENCODER(FirePowerDisabled);
    DtDECLARE_ENTITY_ATTR_ENCODER(ForceID);
    DtDECLARE_ENTITY_ATTR_ENCODER(IsConcealed);
};
// This macro is used to define simple attribute encoder functions
// (members of a derived DtHlaStateEncoder that encode
// individual attributes). The simple encoding is just to implicitly
// convert the value returned by the inspector to a netType,
// and add it to the ahvps.
#define DtDEFINE_SIMPLE_MIL_ENTITY_ATTR_ENCODER( \
    attrName, netType, inspector) \
    DtDEFINE_SIMPLE_ATTR_ENCODER(DtMilitaryEntityEncoder, \
        DtEntityStateRepository, \
        attrName, netType, inspector)

// This macro is used to define simple attribute checker functions
// (members of a derived DtHlaStateEncoder that checker whether
// a particular attribute is out of date. The simple implementation
// is to just compare the values returned by the two StateReps'
// inspector functions.
#define DtDEFINE_SIMPLE_MIL_ENTITY_ATTR_CHECKER( \
    attrName, inspector) \
    DtDEFINE_SIMPLE_ATTR_CHECKER(DtMilitaryEntityEncoder, \
        DtEntityStateRepository, \
        attrName, inspector)

#endif

```

milPlatDec.h

```
#if DtHLA

#include "milEntDec.h"

// class DtMilitaryPlatformEntityDecoder:
//
// Instances of DtMilitaryPlatformEntityDecoder are used to decode a
// military platform entity state message into a state repository in
// an HLA
// (as opposed to DIS) environment.

class DT_DLL_VL DtMilitaryPlatformEntityDecoder : public DtMilitary-
EntityDecoder
{
public:

    // constructor
    DtMilitaryPlatformEntityDecoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtMilitaryPlatformEntityDecoder();

    // copy constructor
    DtMilitaryPlatformEntityDecoder(
        const DtMilitaryPlatformEntityDecoder& orig);

    // assignment operator
    DtMilitaryPlatformEntityDecoder& operator=(
        const DtMilitaryPlatformEntityDecoder& orig);

public:

    // Create a DtMilitaryPlatformEntityDecoder.
    // User is responsible for deletion.
    static DtHlaStateDecoder* create(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

protected:

    // Add decoding functions for the attributes.
    // Note: Non-virtual since meant to be called from constructors
    to
    //      add decoders.
    void addMyDecoders();
}
```

```
// static decode<AttributeName> member functions.
DtDECLARE_ENTITY_ATTR_DECODER(AfterburnerOn);
DtDECLARE_ENTITY_ATTR_DECODER(HasAmmunitionSupplyCap);
DtDECLARE_ENTITY_ATTR_DECODER(LauncherRaised);
};
// These macros are used to define simple attribute decoder func-
// tions
// (members of a derived DtHlaStateDecoder that decode
// individual attributes). The simple decoding is just to
// convert the value pulled out of the ahvps to a host type using
// the cast expression castExpr, and pass it to the mutator func-
// tion.
#define DtDEFINE_SIMPLE_MIL_PLAT_ENTITY_ATTR_DECODER_WITH_CAST( \
    attrName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_ATTR_DECODER_WITH_CAST( \
        DtMilitaryPlatformEntityDecoder, DtEntityStateRepository, \
        attrName, netType, mutator, castExpr)

#define DtDEFINE_SIMPLE_MIL_PLAT_ENTITY_ATTR_DECODER( \
    attrName, netType, inspector) \
    DtDEFINE_SIMPLE_MIL_PLAT_ENTITY_ATTR_DECODER_WITH_CAST( \
        attrName, netType, inspector, (netType))

#endif
```

milPlatEnc.h

```
#if DtHLA

#include "milEntEnc.h"
#include "entitySR.h"

// class DtMilitaryPlatformEntityEncoder:
//
// Instances of DtMilitaryPlatformEntityEncoder are used to encode a
// military platform entity's state repository into a state message
// in an
// HLA environment.

class DT_DLL_VL DtMilitaryPlatformEntityEncoder : public DtMilitary-
EntityEncoder
{
public:

    // constructor
    DtMilitaryPlatformEntityEncoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtMilitaryPlatformEntityEncoder();

    // copy constructor
    DtMilitaryPlatformEntityEncoder(
        const DtMilitaryPlatformEntityEncoder& orig);

    // assignment operator
    DtMilitaryPlatformEntityEncoder& operator=(
        const DtMilitaryPlatformEntityEncoder& orig);

public:

    // Create a DtMilitaryPlatformEntityEncoder.
    // Caller is responsible for deletion.
    static DtHlaStateEncoder* create(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

protected:

    // static need<Attribute> member functions.
    DtDECLARE_ENTITY_ATTR_CHECKER(AfterburnerOn);
    DtDECLARE_ENTITY_ATTR_CHECKER(HasAmmunitionSupplyCap);
    DtDECLARE_ENTITY_ATTR_CHECKER(LauncherRaised);
```



```

    // static decode<Attribute> member functions.
    DtDECLARE_ENTITY_ATTR_ENCODER(AfterburnerOn);
    DtDECLARE_ENTITY_ATTR_ENCODER(HasAmmunitionSupplyCap);
    DtDECLARE_ENTITY_ATTR_ENCODER(LauncherRaised);
};
// This macro is used to define simple attribute encoder functions
// (members of a derived DtHlaStateEncoder that encode
// individual attributes). The simple encoding is just to implicitly
// convert the value returned by the inspector to a netType,
// and add it to the ahvps.
#define DtDEFINE_SIMPLE_MIL_PLAT_ENTITY_ATTR_ENCODER( \
    attrName, netType, inspector) \
    DtDEFINE_SIMPLE_ATTR_ENCODER(DtMilitaryPlatformEntityEncoder, \
        DtEntityStateRepository, \
        attrName, netType, inspector)

// This macro is used to define simple attribute checker functions
// (members of a derived DtHlaStateEncoder that checker whether
// a particular attribute is out of date. The simple implementation
// is to just compare the values returned by the two StateReps'
// inspector functions.
#define DtDEFINE_SIMPLE_MIL_PLAT_ENTITY_ATTR_CHECKER( \
    attrName, inspector) \
    DtDEFINE_SIMPLE_ATTR_CHECKER(DtMilitaryPlatformEntityEncoder, \
        DtEntityStateRepository, \
        attrName, inspector)

#endif

```

mineDataPdu.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "hostStructs.h"
#include "pdu.h"
#include "mathTypes.h"
#include "exerciseConn.h"

#include "packets/mineDataP.h"

// class DtMinefieldDataPdu:
//
// Instances of DtMinefieldDataPdu are used to communicate minefield
// data
// information.

// DtMinefieldDataPdu callback function signature.
typedef void (*DtMinefieldDataPduCb)(DtMinefieldDataPdu* pdu, void*
usr);

class DT_DLL_VL DtMinefieldDataPdu : public DtPdu
{
public:

    // Constructors
    DtMinefieldDataPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtMinefieldDataPdu(const DtNetMinefieldDataPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtMinefieldDataPdu(const DtMinefieldDataPdu& orig);

    // destructor
    virtual ~DtMinefieldDataPdu();

    // Assignment operator
    DtMinefieldDataPdu& operator=(const DtMinefieldDataPdu& orig);

    // Set/Get minefield ID
    virtual void setMinefieldID(const DtEntityIdentifier& id);
    virtual const DtEntityIdentifier& minefieldID() const;

    // Set/Get requesting entity ID
    virtual void setRequestingEntityID(const DtEntityIdentifier& id);
    virtual const DtEntityIdentifier& requestingEntityID() const;

    // Set/Get minefield sequence number
```

```

virtual void setMinefieldSequenceNumber(DtU16 field);
virtual DtU16 minefieldSequenceNumber() const;

// Set/Get the request ID
virtual void setRequestID(DtU8 value);
virtual DtU8 requestID() const;

// Set/Get the PDU sequence number
virtual void setPDU_SequenceNumber(DtU8 value);
virtual DtU8 PDU_SequenceNumber() const;

// Set/Get the number of PDUs
virtual void setNumberOfPdus(DtU8 value);
virtual DtU8 numberOfPdus() const;

// Set/Get the number of mines (this call erases any data which
was set
// in the variable length fields)
virtual DtBoolean setNumberOfMines(DtU8 value);
virtual DtU8 numberOfMines() const;

// Set/Get the number of sensor types (this call erases any data
which
// was set in the variable length fields)
virtual DtBoolean setNumberOfSensorTypes(DtU8 value);
virtual DtU8 numberOfSensorTypes() const;

// Set/Get the data filter as a 32-bit value (this call erases any
data
// which was set in the variable length fields)
virtual void setDataFilter(DtU32 value);
virtual DtU32 dataFilter() const;

// Set/Get a data filter bit (this call erases any data which was
// set in the variable length fields)
virtual void setDataFilterBit(
    DtMinefieldDataFilterField field, DtBoolean value);
virtual DtBoolean dataFilterBit(DtMinefieldDataFilterField field)
const;

// Set/Get the mine type record.
virtual void setMineType(const DtEntityType& value);
virtual const DtEntityType& mineType() const;

// Set/Get a sensor type category. Returns DtFalse if the index
is out
// of range.
virtual DtBoolean setSensorTypeCategory(
    int sensorIndex, DtSensorTypeCategory value);
virtual DtBoolean getSensorTypeCategory(

```

```
    int sensorIndex, DtSensorTypeCategory& value) const;

    // Set/Get a sensor type sub-category. Returns DtFalse if the
index is out
    // of range.
    virtual DtBoolean setSensorTypeSubcategory(
        int sensorIndex, DtU16 value);
    virtual DtBoolean getSensorTypeSubcategory(
        int sensorIndex, DtU16& value) const;

    // Set/Get a mine location. Returns DtFalse if the index is out
of range.
    virtual DtBoolean setMineLocation(int mineIndex, DtConstVector
value);
    virtual DtBoolean getMineLocation(int mineIndex, DtVectorRef
value) const;

    // Set/Get a ground burial depth offset. Returns DtFalse if the
index
    // is out of range or if an appropriate flag bit is not set.
    virtual DtBoolean setGroundBurialDepthOffset(
        int mineIndex, DtFloat32 value);
    virtual DtBoolean getGroundBurialDepthOffset(
        int mineIndex, DtFloat32& value) const;

    // Set/Get a water burial depth offset. Returns DtFalse if the
index is
    // out of range or if the appropriate flag bit is not set.
    virtual DtBoolean setWaterBurialDepthOffset(
        int mineIndex, DtFloat32 value);
    virtual DtBoolean getWaterBurialDepthOffset(
        int mineIndex, DtFloat32& value) const;

    // Set/Get a snow burial depth offset. Returns DtFalse if the
index is out
    // of range or if the appropriate flag bit is not set.
    virtual DtBoolean setSnowBurialDepthOffset(
        int mineIndex, DtFloat32 value);
    virtual DtBoolean getSnowBurialDepthOffset(
        int mineIndex, DtFloat32& value) const;

    // Set/Get mine orientation. Returns DtFalse if the index is out
of range
    // or if the appropriate flag bit is not set.
    virtual DtBoolean setMineOrientation(
        int mineIndex, const DtTaitBryan& value);
    virtual DtBoolean getMineOrientation(
        int mineIndex, DtTaitBryan& value) const;
```

```
// Set/Get a thermal contrast value. Returns DtFalse if the
index
// is out of range or if the appropriate flag bit is not set.
virtual DtBoolean setThermalContrast(int mineIndex, DtFloat32
value);
virtual DtBoolean getThermalContrast(int mineIndex, DtFloat32&
value) const;

// Set/Get a reflectance value. Returns DtFalse if the index
// is out of range or if the appropriate flag bit is not set.
virtual DtBoolean setReflectance(int mineIndex, DtFloat32 value);
virtual DtBoolean getReflectance(int mineIndex, DtFloat32& value)
const;

// Set/Get a mine emplacement time. Returns DtFalse if the index
// is out of range or if the appropriate flag bit is not set.
virtual DtBoolean setMineEmplacementTime(int mineIndex, DtTime
value);
virtual DtBoolean getMineEmplacementTime(int mineIndex, DtTime&
value) const;

// Set/Get a mine entity ID. Returns DtFalse if the index is out
of range.
virtual DtBoolean setMineEntityID(int mineIndex, DtU16 value);
virtual DtBoolean getMineEntityID(int mineIndex, DtU16& value)
const;

// Set/Get primary fusing. Returns DtFalse if the index is out of
range
// or if the appropriate flag bit is not set.
virtual DtBoolean setPrimaryFusing(int mineIndex,
DtMinefieldFusing value);
virtual DtBoolean getPrimaryFusing(int mineIndex,
DtMinefieldFusing& value) const;

// Set/Get secondary fusing. Returns DtFalse if the index is out
of range
// or if the appropriate flag bit is not set.
virtual DtBoolean setSecondaryFusing(int mineIndex,
DtMinefieldFusing value);
virtual DtBoolean getSecondaryFusing(int mineIndex,
DtMinefieldFusing& value) const;

// Set/Get anti-handling device. Returns DtFalse if the index is
out of
// range or if the appropriate flag bit is not set.
virtual DtBoolean setAntiHandlingDevice(int mineIndex,
DtMinefieldAntiHandlingDevice value);
virtual DtBoolean getAntiHandlingDevice(int mineIndex,
DtMinefieldAntiHandlingDevice& value) const;
```

```
// Set/Get scalar detection coefficient. Returns DtFalse if the
index
// is out of range or if the appropriate flag bit is not set.
virtual DtBoolean setScalarDetectionCoefficient(
    int sensorIndex, int mineIndex, DtU8 value);
virtual DtBoolean getScalarDetectionCoefficient(
    int sensorIndex, int mineIndex, DtU8& value) const;

// Set/Get minefield algae. Returns DtFalse if the index
// is out of range or if the appropriate flag bit is not set.
virtual DtBoolean setMinefieldAlgae(
    int mineIndex, DtMinefieldAlgae value);
virtual DtBoolean getMinefieldAlgae(
    int mineIndex, DtMinefieldAlgae& value) const;

// Set/Get paint scheme. Returns DtFalse if the index
// is out of range or if the appropriate flag bit is not set.
virtual DtBoolean setPaintScheme(
    int mineIndex, DtMinefieldPaintScheme value);
virtual DtBoolean getPaintScheme(
    int mineIndex, DtMinefieldPaintScheme& value) const;

// Set/Get the number of trip/detonation wires. Returns DtFalse
if the
// index is out of range or if the appropriate flag bit is not
set.
virtual DtBoolean setNumTripWires(int mineIndex, DtU8 num);
virtual DtBoolean getNumTripWires(int mineIndex, DtU8& num)
const;

// Set/Get the number of vertices. Returns DtFalse if the index
is out
// of range or if the appropriate flag bit is not set.
virtual DtBoolean setNumVertices(
    int mineIndex, int wireIndex, DtU8 num);
virtual DtBoolean getNumVertices(
    int mineIndex, int wireIndex, DtU8& num) const;

// Set/Get a vertex. Returns DtFalse if the index is out of range
or if
// the appropriate flag bit is not set.
virtual DtBoolean setVertex(int mineIndex, int wireIndex, int
vertexIndex,
    DtConstVector value);
virtual DtBoolean getVertex(int mineIndex, int wireIndex, int
vertexIndex,
    DtVectorRef value) const;

// Print the data contained by this PDU
```

```

    virtual void printData() const;

public:
    // Create a DtMinefieldDataPdu.
    // Caller is responsible for deletion.
    static DtPdu* create(const DtNetPacket* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // Add (register)/Remove (deregister) function to be called when
    // pdu is received.
    static void addCallback(DtExerciseConn* conn,
        DtMinefieldDataPduCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtMinefieldDataPduCb cb, void* usr);

protected:
    // Returns the internal PDU kind
    virtual DtPduKind internalGetPduKind() const;

    // Casts the netPdu element to our network struct
    virtual DtNetMinefieldDataPdu* netPduStruct() const;

    // Casts the 'this' pointer for changing elements in a const
    function
    // Non-virtual since you can't override based on return type
    DtMinefieldDataPdu* self() const;

    // Returns a pointer to the first byte of variable length data
    virtual char* var() const;

    // Returns a buffer offset to the sensor type records
    virtual int sensorTypeRecordOffset(int sensorIndex) const;

    // Returns a buffer offset to the padding following sensor type
    information
    virtual int sensorTypePaddingOffset() const;

    // Returns the number of padding bytes following sensor type
    information
    virtual int sensorTypePaddingBytes() const;

    // Returns a buffer offset to the mine location records
    virtual int mineLocationRecordOffset(int mineIndex) const;

    // Returns a buffer offset to the ground burial depth records
    virtual int groundBurialDepthRecordOffset(int mineIndex) const;

    // Returns a buffer offset to the water burial depth records

```

```
virtual int waterBurialDepthRecordOffset(int mineIndex) const;

// Returns a buffer offset to the snow burial depth records
virtual int snowBurialDepthRecordOffset(int mineIndex) const;

// Returns a buffer offset to the mine orientation records
virtual int mineOrientationRecordOffset(int mineIndex) const;

// Returns a buffer offset to the thermal contrast records
virtual int thermalContrastRecordOffset(int mineIndex) const;

// Returns a buffer offset to the reflectance records
virtual int reflectanceRecordOffset(int mineIndex) const;

// Returns a buffer offset to the mine emplacement time records
virtual int mineEmplacementTimeRecordOffset(int mineIndex) const;

// Returns a buffer offset to the mine entity ID records
virtual int mineEntityID_RecordOffset(int mineIndex) const;

// Returns a buffer offset to the fusing records
virtual int fusingRecordOffset(int mineIndex) const;

// Returns a buffer offset to the scalar detection coefficient
records
virtual int scalarDetectionCoefficientRecordOffset(
    int sensorIndex, int mineIndex) const;

// Returns a buffer offset to the paint scheme records
virtual int paintSchemeRecordOffset(int mineIndex) const;

// Returns a buffer offset to the padding following paint scheme
records
virtual int paintSchemePaddingOffset() const;

// Returns the number of padding bytes following paint scheme
records
virtual int paintSchemePaddingBytes() const;

// Returns a buffer offset to the number of wires records
virtual int numberOfTripWiresRecordOffset(int mineIndex) const;

// Returns a buffer offset to the padding following number of
wires records
virtual int numberOfTripWiresPaddingOffset() const;

// Returns the number of padding bytes following number of wires
records
virtual int numberOfTripWiresPaddingBytes() const;
```

```

    // Returns the length of the variable data section in the PDU
    after the
    // variable data has been reset. For each mine, the number of
    trip wires
    // has been set to zero so there is no variable length data
    following.
    virtual int variableDataLengthAfterReset() const;

    // Returns a buffer offset to the number of vertices records
    virtual int numberOfVerticesRecordOffset(
        int mineIndex, int wireIndex) const;

    // Returns a buffer offset to the padding following number of
    // vertices records
    virtual int numberOfVerticesPaddingOffset() const;

    // Returns the number of padding bytes following number of
    vertices records
    virtual int numberOfVerticesPaddingBytes() const;

    // Returns a buffer offset to the vertex records
    virtual int vertexRecordOffset(int mineIndex, int wireIndex,
        int vertexIndex) const;

    // Returns the length of the variable data section in the PDU
    virtual int variableDataLength() const;

protected:

    // Size of PDU with zero parameter records.
    static int theStaticSize;

protected:

    // Temporary objects for reference returns
    DtEntityIdentifier myEntityID;
    DtEntityType myEntityType;

};
#endif
#endif

```

minePerimeter.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"

#include "packets/mineStateP.h"

// class DtMinefieldPerimeterRecord:
//
// DtMinefieldPerimeterRecord is a class which contains coordinates
// for the
// minefield perimeter points. This record is used in the minefield
// state
// and minefield query PDUs.

class DT_DLL_VL DtMinefieldPerimeterRecord
{
public:

    // Constructors
    DtMinefieldPerimeterRecord();
    DtMinefieldPerimeterRecord(const DtMinefieldPerimeterRecord&
orig);

    // Assignment operator
    DtMinefieldPerimeterRecord& operator=(
        const DtMinefieldPerimeterRecord& orig);
    DtMinefieldPerimeterRecord& operator=(
        const DtNetMinefieldPerimeterRecord& orig);

    // Destructor
    virtual ~DtMinefieldPerimeterRecord();

    // Set/Get the X Component
    virtual void setX(DtFloat32 value);
    virtual DtFloat32 x() const;

    // Set/Get the Y Component
    virtual void setY(DtFloat32 value);
    virtual DtFloat32 y() const;

    // Print the information contained in the record
    virtual void print() const;

    // Load/Unload data from a packet record
```

```
virtual void unloadNetRecord(  
    const DtNetMinefieldPerimeterRecord& record);  
virtual void loadNetRecord(  
    DtNetMinefieldPerimeterRecord& record) const;  
  
protected:  
  
    // Data members:  
    DtFloat32          myX;  
    DtFloat32          myY;  
  
};  
#endif  
#endif
```

mineQueryPdu.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "hostStructs.h"
#include "pdu.h"
#include "exerciseConn.h"

#include "packets/mineQueryP.h"

#include "minePerimeter.h"

// class DtMinefieldQueryPdu:
//
// Instances of DtMinefieldQueryPdu are used to communicate mine-
// field state
// information.

// DtMinefieldQueryPdu callback function signature.
typedef void (*DtMinefieldQueryPduCb)(DtMinefieldQueryPdu* pdu,
void* usr);

class DT_DLL_VL DtMinefieldQueryPdu : public DtPdu
{
public:

    // Constructors
    DtMinefieldQueryPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtMinefieldQueryPdu(const DtNetMinefieldQueryPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtMinefieldQueryPdu(const DtMinefieldQueryPdu& orig);

    // destructor
    virtual ~DtMinefieldQueryPdu();

    // Assignment operator
    DtMinefieldQueryPdu& operator=(const DtMinefieldQueryPdu& orig);

    // Set/Get minefield ID
    virtual void setMinefieldID(const DtEntityIdentifier& id);
    virtual const DtEntityIdentifier& minefieldID() const;

    // Set/Get requesting entity ID
    virtual void setRequestingEntityID(const DtEntityIdentifier& id);
    virtual const DtEntityIdentifier& requestingEntityID() const;
```

```
// Set/Get request ID
virtual void setRequestID(DtU8 id);
virtual DtU8 requestID() const;

// Set/Get the number of perimeter points
virtual DtBoolean setNumPerimeterPoints(DtU8 num);
virtual DtU8 numPerimeterPoints() const;

// Set/Get the number of sensor types
virtual DtBoolean setNumSensorTypes(DtU8 num);
virtual DtU8 numSensorTypes() const;

// Set/Get the data filter (as a 32-bit integer)
virtual void setDataFilter(DtU32 value);
virtual DtU32 dataFilter() const;

// Set/Get data filter attributes (DtTrue indicates that the data
for that
// field is requested, DtFalse indicates that the data for that
field is
// not requested.
virtual void setDataFilterField(
    DtMinefieldDataFilterField field, DtBoolean value);
virtual DtBoolean getDataFilterField(DtMinefieldDataFilterField
field);

// Set/Get requested mine type
virtual void setRequestedMineType(const DtEntityType& type);
virtual const DtEntityType& requestedMineType() const;

// Set/Get a perimeter point coordinate record. Returns DtFalse
if the
// index is out of range.
virtual DtBoolean setPerimeterPoint(int index,
    const DtMinefieldPerimeterRecord& init);
virtual DtBoolean getPerimeterPoint(int index,
    DtMinefieldPerimeterRecord& record) const;

// Get the sensor type category for a record. Returns DtFalse if
the
// index is out of range. The category is set by setSensorTypeR-
ecord.
virtual DtBoolean getSensorTypeCategory(int index,
    DtSensorTypeCategory& value);

// Set a sensor type subcategory for a record. The correct cate-
gory is
// automatically set by this function according to type of the
enumeration
```

```
// in the argument list. DtFalse is returned if the index is out
of range.
virtual DtBoolean setSensorTypeSubcategory(int index,
    DtOpticalSensorTypeSubcategory value);
virtual DtBoolean setSensorTypeSubcategory(int index,
    DtFLIR_SensorTypeSubcategory value);
virtual DtBoolean setSensorTypeSubcategory(int index,
    DtRADAR_SensorTypeSubcategory value);
virtual DtBoolean setSensorTypeSubcategory(int index,
    DtMagneticSensorTypeSubcategory value);
virtual DtBoolean setSensorTypeSubcategory(int index,
    DtLaserSensorTypeSubcategory value);
virtual DtBoolean setSensorTypeSubcategory(int index,
    DtSONARSensorTypeSubcategory value);
virtual DtBoolean setSensorTypeSubcategory(int index,
    DtPhysicalSensorTypeSubcategory value);
virtual DtBoolean setSensorTypeSubcategory(int index,
    DtMultispectralSensorTypeSubcategory value);

// Get a sensor type subcategory for a record. DtFalse is
returned if the
// index is out of range or if the category of the record does not
match
// the category indicated by the type of the enumeration in the
argument
// list.
virtual DtBoolean getSensorTypeSubcategory(int index,
    DtOpticalSensorTypeSubcategory& value);
virtual DtBoolean getSensorTypeSubcategory(int index,
    DtFLIR_SensorTypeSubcategory& value);
virtual DtBoolean getSensorTypeSubcategory(int index,
    DtRADAR_SensorTypeSubcategory& value);
virtual DtBoolean getSensorTypeSubcategory(int index,
    DtMagneticSensorTypeSubcategory& value);
virtual DtBoolean getSensorTypeSubcategory(int index,
    DtLaserSensorTypeSubcategory& value);
virtual DtBoolean getSensorTypeSubcategory(int index,
    DtSONARSensorTypeSubcategory& value);
virtual DtBoolean getSensorTypeSubcategory(int index,
    DtPhysicalSensorTypeSubcategory& value);
virtual DtBoolean getSensorTypeSubcategory(int index,
    DtMultispectralSensorTypeSubcategory& value);

// Print the data contained by this PDU
virtual void printData() const;

public:

    // Create a DtMinefieldQueryPdu.
    // Caller is responsible for deletion.
```

```

static DtPdu* create(const DtNetPacket* initial,
    DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

// Add (register)/Remove (deregister) function to be called when
// pdu is received.
static void addCallback(DtExerciseConn* conn,
    DtMinefieldQueryPduCb cb, void* usr);
static void removeCallback(DtExerciseConn* conn,
    DtMinefieldQueryPduCb cb, void* usr);

```

protected:

```

// Returns the internal PDU kind
virtual DtPduKind internalGetPduKind() const;

// Casts the netPdu element to our network struct
virtual DtNetMinefieldQueryPdu* netPduStruct() const;

// Casts the 'this' pointer for changing elements in a const
function
// Non-virtual since you can't override based on return type
DtMinefieldQueryPdu* self() const;

// Returns a pointer to the first byte of variable length data
virtual char* var() const;

// Returns a buffer offset to a perimeter point record
virtual int perimeterPointRecordOffset(int recordIndex) const;

// Returns a buffer offset to a sensor type record
virtual int sensorTypeRecordOffset(int recordIndex) const;

// Set/Get the sensor type. (As a 16-bit integer)
virtual void setSensorType(int index, DtU16 value);
virtual DtU16 sensorType(int index) const;

```

protected:

```

// Size of PDU with zero parameter records.
static int theStaticSize;

```

protected:

```

// Temporary objects for reference returns
DtEntityIdentifier myEntityID;
DtEntityType myEntityType;
DtMinefieldPerimeterRecord myPerimeterRecord;
};
#endif
#endif

```

mineRsNackPdu.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "hostStructs.h"
#include "pdu.h"
#include "exerciseConn.h"

#include "packets/mineRsNackP.h"

// class DtMinefieldResponseNackPdu:
//
// Instances of DtMinefieldResponseNackPdu are used to request a
// retransmission of minefield data PDUs.

// DtMinefieldResponseNackPdu callback function signature.
typedef void (*DtMinefieldResponseNackPduCb)
    (DtMinefieldResponseNackPdu* pdu, void* usr);

class DT_DLL_VL DtMinefieldResponseNackPdu : public DtPdu
{
public:

    // Constructors
    DtMinefieldResponseNackPdu(DtBufferPtr buffer =
DtUSE_INTERNAL_BUFFER);
    DtMinefieldResponseNackPdu(const DtNetMinefieldResponseNackPdu*
initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtMinefieldResponseNackPdu(const DtMinefieldResponseNackPdu&
orig);

    // destructor
    virtual ~DtMinefieldResponseNackPdu();

    // Assignment operator
    DtMinefieldResponseNackPdu& operator=(
        const DtMinefieldResponseNackPdu& orig);

    // Set/Get minefield ID
    virtual void setMinefieldID(const DtEntityIdentifier& id);
    virtual const DtEntityIdentifier& minefieldID() const;

    // Set/Get requesting entity ID
    virtual void setRequestingEntityID(const DtEntityIdentifier& id);
    virtual const DtEntityIdentifier& requestingEntityID() const;
```



```

// Set/Get request ID
virtual void setRequestID(DtU8 id);
virtual DtU8 requestID() const;

// Set/Get the number of missing PDUs
virtual DtBoolean setNumMissingPDUs(DtU8 num);
virtual DtU8 numMissingPDUs() const;

// Set/Get a PDU sequence number. Returns DtFalse if the
// index is out of range.
virtual DtBoolean setPDU_SequenceNumber(int index, DtU8 num);
virtual DtBoolean getPDU_SequenceNumber(int index, DtU8& num)
const;

// Print the data contained by this PDU
virtual void printData() const;

public:

// Create a DtMinefieldResponseNackPdu.
// Caller is responsible for deletion.
static DtPdu* create(const DtNetPacket* initial,
    DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

// Add (register)/Remove (deregister) function to be called when
// pdu is received.
static void addCallback(DtExerciseConn* conn,
    DtMinefieldResponseNackPduCb cb, void* usr);
static void removeCallback(DtExerciseConn* conn,
    DtMinefieldResponseNackPduCb cb, void* usr);

protected:

// Returns the internal PDU kind
virtual DtPduKind internalGetPduKind() const;

// Casts the netPdu element to our network struct
virtual DtNetMinefieldResponseNackPdu* netPduStruct() const;

// Returns a pointer to the first byte of variable length data
virtual char* var() const;

// Casts the 'this' pointer for changing elements in a const
function
// Non-virtual since you can't override based on return type
DtMinefieldResponseNackPdu* self() const;

protected:

```

```
// Size of PDU with zero parameter records.
static int theStaticSize;

protected:

    // Temporary objects for reference returns
    DtEntityIdentifier myEntityID;

};
#endif
#endif
```

mineStatePdu.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "hostStructs.h"
#include "pdu.h"
#include "mathTypes.h"
#include "exerciseConn.h"

#include "packets/mineStateP.h"

#include "minePerimeter.h"

// class DtMinefieldStatePdu:
//
// Instances of DtMinefieldStatePdu are used to communicate mine-
// field state
// information.

// DtMinefieldStatePdu callback function signature.
typedef void (*DtMinefieldStatePduCb)(DtMinefieldStatePdu* pdu,
void* usr);

class DT_DLL_VL DtMinefieldStatePdu : public DtPdu
{
public:

    // Constructors
    DtMinefieldStatePdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtMinefieldStatePdu(const DtNetMinefieldStatePdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtMinefieldStatePdu(const DtMinefieldStatePdu& orig);

    // destructor
    virtual ~DtMinefieldStatePdu();

    // Assignment operator
    DtMinefieldStatePdu& operator=(const DtMinefieldStatePdu& orig);

    // Set/Get minefield ID
    virtual void setMinefieldID(const DtEntityIdentifier& id);
    virtual const DtEntityIdentifier& minefieldID() const;

    // Set/Get minefield sequence number
    virtual void setSequenceNumber(DtU16 field);
    virtual DtU16 sequenceNumber() const;

```

```
// Set/Get the force ID
virtual void setForceID(DtForceType force);
virtual DtForceType forceID() const;

// Set/Get the number of perimeter points
virtual DtBoolean setNumPerimeterPoints(DtU8 num);
virtual DtU8 numPerimeterPoints() const;

// Set/Get minefield type
virtual void setMinefieldType(const DtEntityType& type);
virtual const DtEntityType& minefieldType() const;

// Set/Get the number of mine types
virtual DtBoolean setNumMineTypes(DtU16 num);
virtual DtU16 numMineTypes() const;

// Set/Get location of minefield
virtual void setLocation(DtConstVector coords);
virtual DtConstVector location() const;

// Set/Get orientation
virtual void setOrientation(const DtTaitBryan& orientation);
virtual const DtTaitBryan& orientation() const;

// Set/Get appearance (as a 16 bit integer)
virtual void setAppearance(DtU16 app);
virtual DtU16 appearance() const;

// Set/Get appearance by attribute
virtual void setAppearanceAttribute(
    DtAppearAttribute attr, DtAppearAttrVal val);
virtual DtAppearAttrVal appearanceAttribute(
    DtAppearAttribute attr) const;

// Set/Get which network protocol mode is being used in the
// minefield transmission.
virtual void setMinefieldProtocolMode(DtMinefieldProtocol mode);
virtual DtMinefieldProtocol minefieldProtocolMode() const;

// Set/Get a perimeter point coordinate record, returns DtFalse
if the
// index is out of range.
virtual DtBoolean setPerimeterPoint(int index,
    const DtMinefieldPerimeterRecord& init);
virtual DtBoolean getPerimeterPoint(int index,
    DtMinefieldPerimeterRecord& record) const;

// Set/Get a mine type record. Returns DtFalse if the index is
// out of range.
```

```

virtual DtBoolean setMineType(int index,
    const DtEntityType& init);
virtual DtBoolean getMineType(int index,
    DtEntityType& record) const;

// Print the data contained by this PDU
virtual void printData() const;

public:

    // Create a DtMinefieldStatePdu.
    // Caller is responsible for deletion.
    static DtPdu* create(const DtNetPacket* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // Add (register)/Remove (deregister) function to be called when
    // pdu is received.
    static void addCallback(DtExerciseConn* conn,
        DtMinefieldStatePduCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtMinefieldStatePduCb cb, void* usr);

protected:

    // Returns the internal PDU kind
    virtual DtPduKind internalGetPduKind() const;

    // Casts the netPdu element to our network struct
    virtual DtNetMinefieldStatePdu* netPduStruct() const;

    // Casts the 'this' pointer for changing elements in a const
    function
    // Non-virtual since you can't override based on return type
    DtMinefieldStatePdu* self() const;

    // Returns a pointer to the first byte of variable length data
    virtual char* var() const;

    // Returns a buffer offset to a perimeter point record
    virtual int perimeterPointRecordOffset(int recordIndex) const;

    // Returns a buffer offset to a mine type record
    virtual int mineTypeRecordOffset(int recordIndex) const;

protected:

    // Size of PDU with zero parameter records.
    static int theStaticSize;

protected:

```

```
// Temporary objects for reference returns
DtEntityIdentifier myEntityID;
DtEntityType myEntityType;
DtTaitBryan myEulerAngles;
DtVector myVector;
DtMinefieldPerimeterRecord myPerimeterRecord;

};
#endif
#endif
```

munEntDec.h

```

#if DtHLA

#include "milEntDec.h"

// class DtMunitionEntityDecoder:
//
// Instances of DtMunitionEntityDecoder are used to decode a
// munition entity state message into a state repository in an HLA
// (as opposed to DIS) environment.

class DT_DLL_VL DtMunitionEntityDecoder : public DtMilitaryEntityDe-
coder
{
public:

    // constructor
    DtMunitionEntityDecoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtMunitionEntityDecoder();

    // copy constructor
    DtMunitionEntityDecoder(const DtMunitionEntityDecoder& orig);

    // assignment operator
    DtMunitionEntityDecoder& operator=(const DtMunitionEntityDecoder&
orig);

public:

    // Create a DtMunitionEntityDecoder.
    // User is responsible for deletion.
    static DtHlaStateDecoder* create(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

protected:

    // Add decoding functions for the attributes.
    // Note: Non-virtual since meant to be called from constructors
to
    //      add decoders.
    void addMyDecoders();

    // static decode<AttributeName> member functions.

```

```
DtDECLARE_ENTITY_ATTR_DECODER(LauncherFlashPresent);
};
// These macros are used to define simple attribute decoder func-
// tions
// (members of a derived DtHlaStateDecoder that decode
// individual attributes). The simple decoding is just to
// convert the value pulled out of the ahvps to a host type using
// the cast expression castExpr, and pass it to the mutator func-
// tion.
#define DtDEFINE_SIMPLE_MUN_ENTITY_ATTR_DECODER_WITH_CAST( \
    attrName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_ATTR_DECODER_WITH_CAST( \
        DtMunitionEntityDecoder, DtEntityStateRepository, \
        attrName, netType, mutator, castExpr)

#define DtDEFINE_SIMPLE_MUN_ENTITY_ATTR_DECODER( \
    attrName, netType, inspector) \
    DtDEFINE_SIMPLE_MUN_ENTITY_ATTR_DECODER_WITH_CAST( \
        attrName, netType, inspector, (netType))

#endif
```


munEntEnc.h

```

#if DtHLA

#include "milEntEnc.h"
#include "entitySR.h"

// class DtMunitionEntityEncoder:
//
// Instances of DtMunitionEntityEncoder are used to encode a
// munition entity's state repository into a state message in an
// HLA environment.

class DT_DLL_VL DtMunitionEntityEncoder : public DtMilitaryEntityEncoder
{
public:

    // constructor
    DtMunitionEntityEncoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtMunitionEntityEncoder();

    // copy constructor
    DtMunitionEntityEncoder(const DtMunitionEntityEncoder& orig);

    // assignment operator
    DtMunitionEntityEncoder& operator=(const DtMunitionEntityEncoder&
orig);

public:

    // Create a DtMunitionEntityEncoder.
    // Caller is responsible for deletion.
    static DtHlaStateEncoder* create(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

protected:

    // static need<Attribute> member functions.
    DtDECLARE_ENTITY_ATTR_CHECKER(LauncherFlashPresent);

    // static decode<Attribute> member functions.
    DtDECLARE_ENTITY_ATTR_ENCODER(LauncherFlashPresent);
};

```

```
// This macro is used to define simple attribute encoder functions
// (members of a derived DtHlaStateEncoder that encode
// individual attributes). The simple encoding is just to implicitly
// convert the value returned by the inspector to a netType,
// and add it to the ahvps.
#define DtDEFINE_SIMPLE_MUN_ENTITY_ATTR_ENCODER( \
    attrName, netType, inspector) \
    DtDEFINE_SIMPLE_ATTR_ENCODER(DtMunitionEntityEncoder, \
        DtEntityStateRepository, \
        attrName, netType, inspector)

// This macro is used to define simple attribute checker functions
// (members of a derived DtHlaStateEncoder that checker whether
// a particular attribute is out of date. The simple implementation
// is to just compare the values returned by the two StateReps'
// inspector functions.
#define DtDEFINE_SIMPLE_MUN_ENTITY_ATTR_CHECKER( \
    attrName, inspector) \
    DtDEFINE_SIMPLE_ATTR_CHECKER(DtMunitionEntityEncoder, \
        DtEntityStateRepository, \
        attrName, inspector)

#endif
```

netBoolean.h

```

#if DtHLA

#include "NetTypes.h"
#include "MachineTypes.h"

// class DtNetBoolean:
//
// Instances of DtNetBoolean are network representations of bool-
// eans.
// The only thing that's required is that they can be cast to
// DtBooleans.
// This class has NO virtual functions since it is a NetStruct and
// we
// rely on its size and byte layout.

#if !DtRPR016

typedef DtNetU8 DtNetBoolean;

#else

#if !DT_LITTLE_ENDIAN

typedef DtNetU32 DtNetBoolean;

#else

class DT_DLL_VL DtNetBoolean
{
public:

    // default constructor
    DtNetBoolean();

    // destructor
    ~DtNetBoolean();

    // If a copy constructor is implemented, the assignment operator
    should
    // also be implemented. Should generally implement both of them
    anyway to
    // avoid memory problems due to inappropriate compiler provided
    defaults.
    // Access varies so need to put proper access keyword above these
    // declarations.

    // copy constructor

```

```
DtNetBoolean(const DtNetBoolean& orig);

// assignment operator
DtNetBoolean& operator=(const DtNetBoolean& orig);

// from host constructor
DtNetBoolean(DtBoolean val);

// from host assignment operator
DtNetBoolean& operator=(DtBoolean val);

// to host cast operator
DtNetBoolean& operator DtBoolean() const;

protected:

    DtNetU32 myRep;
};
#endif
#endif
#endif
```

NetSocket.h

```

/*
 *
 * NetSocket provides several overloaded constructors that allow the
 * user
 * to specify UDP port, device, and destination address in any
 * combination
 * except port and destination address together.
 *
 * In addition, each constructor has an optional flags argument,
 * which
 * may be set to DtRead (read only), DtWrite (write only) or both
 * (default, DtRead | DtWrite).
 *
 * The default destination address is broadcast.
 * The default network interface device is the first appropriate one
 * located by the constructor.
 */

#ifndef _NetSocket_h
#define _NetSocket_h

#include "MachineTypes.h"    /* needed for <sys/types.h> or rela-
tives */
#include "NetTypes.h"

#if !WIN32
extern "C" {
#include <netinet/in.h>
}
#endif

#if __osf__
#include <sys/param.h> /* needed for stuff that should be
netinet/in.h */
#endif

#ifdef _GT_
#include "IpUdp.h"

#include <sys/socket.h>      /* need for if.h          */
#include <net/if.h>          /* need for if_ether.h */
#include <net/if_arp.h>      /* need for if_ether.h */
#include <netinet/if_ether.h> /* need for struct ether_header */
#endif

#define DtRead((DtU8) 0x01)
#define DtWrite((DtU8) 0x02)

```

```
#define DtReadWrite (DtRead | DtWrite)

/* setting this bit in the flags constructor argument will cause the
   NetSocket to ignore all traffic (incl. broadcast when dest_addr
   is
       multicast and point-to-point) besides that to dest_addr. */
#define DtBindToDestAddr ((DtU8) 0x08)

#define DtQuiet ((DtU8) 0x10)

enum DtInitErrorAction {
    DtInitErrorFatal,
    DtInitErrorWarn,
    DtInitErrorIgnore
};

#include "Socket.h"
#include "vlutil.h"

#ifdef NULL
#undef NULL
#define NULL 0
#endif

#ifdef _GT_

typedef struct gtQueueElement
{
    int datalen;
    caddr_t data;
} GtQueueElement;

#endif

class DT_DLL_VLUTIL DtNetSocket : public DtSocket
{
public:
    DtNetSocket(int port=DtDEFAULT_PORT, DtU8 flags=DtReadWrite);

    /* No default for flags since 3rd arg is needed to disambiguate.
     * Note a real loss, however, since any point-to-point NetSocket
     * should probably be write-only anyway.
     */
    DtNetSocket(int port, DtInetAddr destAddr, DtU8 flags);

    /* This constructor is obsolete, and will eventually disappear
     * Use DtInitNetSocket(port, dest_addr->s_addr) instead */
};
```

```

    DtNetSocket(int port, const struct in_addr *dest_addr,
                DtU8 flags=DtReadWrite)
    {
DtInitNetSocket (port, dest_addr->s_addr, flags, NULL);
    }

    DtNetSocket(char* dummy, int sockfd, DtU8 flags = DtReadWrite);

#ifdef _GT_
    /* It only makes sense to set both destAddr and device on the GT.
     * Under UNIX (with IP sockets), the device is implied by the
destAddr
     * It also only makes sense to speciify from_addr on the _GT_.
On UNIX
     * boxes, the from IP address gets filled in automagically.
     */
    DtNetSocket(int port, DtInetAddr fromAddr, DtInetAddr destAddr,
                const char *device, DtU8 flags = DtReadWrite);
#endif

    virtual ~DtNetSocket();

    int sendTo(caddr_t packet, size_t size, DtInetAddr addr);

    int DtRecv(caddr_t buff, size_t);
    int DtRecv(caddr_t *buffptr, int *status=NULL);
    int DtReportPktsRcvd(void);
    DtInetAddr lastSourceInetAddr() {
return ntohl(sourceSockAddr.sin_addr.s_addr);
    }

    void DtBecomeNonBlocking(void);
    void setUnbundling(int onOff);
    int isUnbundling();
    int dataLeftInCurrentBundle();

    int readFd() { return ((RW_flags & DtRead) ? skt_num : -1); }
    int writeFd() { return ((RW_flags & DtWrite) ? skt_num : -1); }
    int isOk() { return ok; }

    static DtInitErrorAction initErrorAction;
#ifdef PCFTP || WIN32
    static DtU32 DtInetBroadcastMask;
    void DtGetBroadcastMask();
#endif

    int udpPort;          /* must be kept around for GT */

inline int listenToMulticastGroup(DtInetAddr multicastAddr) {
    return PrivateListenToMulticastGroup(multicastAddr, 0); }

```

```
    int dropMulticastGroup(DtInetAddr multicastAddr);

int setMcastTtl(int ttl);

#ifdef _GT_
    int DtGtPacketDequeue(caddr_t);
    void DtGtPacketEnqueue(char *);
#endif

protected:
    void DtInitNetSocket(int port, DtInetAddr dest_addr,
        DtU8 rw_flags, const char *device);

    int recvWithStatus(caddr_t *bufptr, int *status);
    int recvFromNet(caddr_t *bufptr);
    int recvWithUnbundling(caddr_t *bufptr);
    void *nextPdu;
    int sizeLeft;
    int unbundling;
    int m_bExternalSocket;
    DtInetAddr m_devAddr;

private:
    DtNetFloat64 pktBuff8[DtMaxPacketSize/sizeof(DtNetFloat64)];
    DtNetFloat64 unbundleBuff8[DtMaxPacket-
Size/sizeof(DtNetFloat64)];

    int init_iodev(const char *);
    int init_write(DtInetAddr *dest);
    int init_read(DtInetAddr dest /* may become the read addr */);
    int PrivateListenToMulticastGroup(DtInetAddr multicastAddr, int
on_init=0);
    void findNextPdu();
    unsigned int currentHeaderMatchesFirstInBundle();

    DtU8 RW_flags;
    int boundToDestAddr;
    DtInetAddr destInetAddr;
    int firstInBundle;
    int firstExId;
    int firstProtocol;

    int ok;
    int skt_num; /* UNIX socket number. */
    struct sockaddr_in destSockAddr;
    struct sockaddr_in sourceSockAddr;

#ifdef _GT_
    char outBuf[2048]; /* does DIS have a defined maximum? */
```



```
        // No, but Ethernet does, and that's the only
        // thing the GT100 will ever work with

    int DtGtSend(caddr_t, size_t);
    int DtGtRecv(caddr_t bufp, size_t bufsize);
    GtQueueElement *PacketQueueOut = 0;
    GtQueueElement *PacketQueueIn = 0;

    /* These point into the above structure after INIT */
    struct ether_header *ethernetHdr;
    struct ip *ipHdr;
    struct udphdr *udpHdr;
    DtInetAddr          from_addr = 0;
#endif

};
/* The functionality of the older constructor
 *   DtNetSocket(const char *device, int port)
 * has been replaced by the call
 *   DtNetSocket(int port, DtInetBroadcastOfDevice(char *device));
 */
/* example device names are "ec0" and "le0" */
```

NetStructs.h

```
#include "chooseDefines.h"
#include "vlTime.h"
#include "NetTypes.h"
#include "netTimeStamp.h"
#include "netBoolean.h"
#include "globalObjDes.h"

// For backwards compatibility
#include "oldTStamp.h"

#ifdef __cplusplus
#include "mathTypes.h"
#endif

typedef DtNetU8      DtNet8Bits;
typedef DtNetU8      DtNet16Bits[2];
typedef DtNetU8      DtNet32Bits[4];
typedef DtNetU8      DtNet64Bits[8];

#ifdef __cplusplus

#if DT_LITTLE_ENDIAN
class DT_DLL_VL DtEventId;
#endif

class DT_DLL_VL DtNetEntityMarking
{
public:
    DtNetU8      DtCharacterSet;
#define DtMaxEntityMarkingLength 11
    DtNetU8      DtText[DtMaxEntityMarkingLength];
};
class DT_DLL_VL DtNetEntityType
{
public:
    DtNetU8DtEntityKind;
    DtNetU8DtDomain;
    DtNetU16DtCountry;
    DtNetU8DtCategory;
    DtNetU8DtSubCategory;
    DtNetU8DtSpecific;
    DtNetU8DtExtra;

#if DT_LITTLE_ENDIAN
    DtEntityType operator = (DtEntityType);
    DtNetEntityType operator = (DtNetEntityType);
#endif
#endif
#endif
```

```

};

class DT_DLL_VL DtNetRadioEntityType
{
public:
    DtNetU8      kind;
    DtNetU8      domain;
    DtNetU16     country;
    DtNetU8      category;
    DtNetU8      nomenclatureVersion;
    DtNetU16     nomenclature;

#ifdef DT_LITTLE_ENDIAN
    DtRadioEntityType operator = (DtRadioEntityType);
    DtNetRadioEntityType operator = (DtNetRadioEntityType);
#endif
};

class DT_DLL_VL DtNetBurstDescriptor
{
public:
    DtNetEntityType DtMunition;
    DtNetU16DtWarhead;
    DtNetU16DtFuze;
    DtNetU16 DtQuantity;
    DtNetU16 DtRate;

#ifdef DT_LITTLE_ENDIAN
    DtBurstDescriptor operator = (DtBurstDescriptor);
    DtNetBurstDescriptor operator = (DtNetBurstDescriptor);
#endif
};

class DT_DLL_VL DtNetSimulationAddress
{
public:
    DtNetU16DtSite;
    DtNetU16 DtHost;

#ifdef DT_LITTLE_ENDIAN
    DtSimulationAddress operator = (DtSimulationAddress);
    DtNetSimulationAddress operator = (DtNetSimulationAddress);
#endif
};

class DT_DLL_VL DtNetEntityIdentifier
{
public:
    DtNetSimulationAddressDtSimulator;
    DtNetU16DtEntity;
};

```

```
#if DT_LITTLE_ENDIAN
    DtEntityIdentifier operator = (DtEntityIdentifier);
    DtNetEntityIdentifier operator = (DtNetEntityIdentifier);
#endif
};

#if DtHLA
typedef DtNetU32 DtNetObjectId;
#if DtRTI103
typedef DtNetObjectId DtNetGlobalObjectDesignator;
#else
// This class is basically a hack so that the simple encoders and
// decoders will work for DtGlobalObjectDesignators. You can cast
// a char* to a DtNetGlobalObjectDesignator* (and vice versa), and
// you can assign a char* to a DtNetGlobalObjectDesignator. It
// probably
// shouldn't be used for anything else.
#define DtMAX_OBJECT_NAME_LENGTH 64
class DT_DLL_VL DtNetGlobalObjectDesignator
{
public:
    DtNetGlobalObjectDesignator& operator=(
        const DtGlobalObjectDesignator& des);
    operator DtGlobalObjectDesignator() const;

protected:
    char myRep[DtMAX_OBJECT_NAME_LENGTH];
};
#endif // DtRTI103
#else // !DtHLA

typedef DtNetEntityIdentifier DtNetObjectId;
typedef DtNetEntityIdentifier DtNetGlobalObjectDesignator;

#endif // !DtHLA

#if DtHLA

class DT_DLL_VL DtNetFederateId
{
public:
    DtNetU16 site;
    DtNetU16 app;

#if DT_LITTLE_ENDIAN
    DtFederateId operator = (DtFederateId);
    DtNetFederateId operator = (DtNetFederateId);
#endif
};
```

```

class DT_DLL_VL DtNetEventId
{
public:
    DtNetEventId();

    DtNetU16 eventNum;
    DtNetGlobalObjectDesignator objectId;

protected:

    // Copy ctor and assignment operator not implemented,
    // since we don't want to change
    // size when necessary. (NetTypes are sometimes used in
    // structures, where we count on their size.)
    DtNetEventId(const DtNetEventId& orig);
    DtNetEventId operator= (const DtNetEventId& orig);
};

#endif // DtHLA

#if !DtHLA

class DT_DLL_VL DtNetEventId
{
public:
    DtSimulationAddress    DtSimulator;
    DtNetU16               DtEvent;

#if DT_LITTLE_ENDIAN
    DtEventId operator = (DtEventId);
    DtNetEventId operator = (DtNetEventId);
#endif
};

#endif // !DtHLA

#if DT_LITTLE_ENDIAN
class DT_DLL_VL DtNet64Vector
{
public:
    DtNet64Vector()
    {rep[0] = rep[1] = rep[2] = 0.0;}
    DtNet64Vector(const DtVector &vec)
    {rep[0] = vec[0]; rep[1] = vec[1]; rep[2] = vec[2];}
    DtVector operator = (const DtVector &vec)
    {rep[0] = vec[0]; rep[1] = vec[1]; rep[2] = vec[2]; return vec;}
    operator DtVector() const {return DtVector(rep[0], rep[1],
rep[2]);}
    DtNetFloat64 &operator[](int i) { return rep[i];}

```

```
DtNetFloat64 operator[](int i) const { return rep[i];}
private:
    DtNetFloat64 rep[3];
};
#else
typedef DtVector DtNet64Vector;
#endif

class DT_DLL_VL DtNet32Vector
{
public:
    DtNet32Vector()
    {rep[0] = rep[1] = rep[2] = (float)0.0;}
    DtNet32Vector(const DtVector &vec)
    {rep[0] = (float)vec[0]; rep[1] = (float)vec[1]; rep[2] =
(float)vec[2];}
    DtVector operator = (const DtVector &vec)
    {rep[0] = (float)vec[0]; rep[1] = (float)vec[1]; rep[2] =
(float)vec[2]; return vec;}
    operator DtVector() const {return DtVector(rep[0], rep[1],
rep[2]);}
    DtNetFloat32 &operator[](int i) { return rep[i];}
    DtNetFloat32 operator[](int i) const { return rep[i];}
private:
    DtNetFloat32 rep[3];
};

class DT_DLL_VL DtNetEulerAngles
{
public:
    DtNetEulerAngles() { psi = theta = phi = (float)0.0;}
    DtNetEulerAngles(const DtTaitBryan &tait)
    {psi = (DtFloat32)tait.psi(); theta = (DtFloat32)tait.theta();
phi = (DtFloat32)tait.phi();}
    DtTaitBryan operator = (const DtTaitBryan &tait)
    {psi = (DtFloat32)tait.psi(); theta = (DtFloat32)tait.theta();
phi = (DtFloat32)tait.phi(); return tait;}
    operator DtTaitBryan() const {return DtTaitBryan(psi, theta,
phi);}
    DtNetFloat32 psi; /* Counter clockwise about z */
    DtNetFloat32 theta; /* Counter clockwise about y */
    DtNetFloat32 phi; /* Counter clockwise about x */
};

#elif !DT_LITTLE_ENDIAN

/*****
 * On Big Endian Machines, C can include these definitions:
 *****/
```

```
typedef struct DtNetRadioEntityType
{
    DtNetU8kind;
    DtNetU8domain;
    DtNetU16country;
    DtNetU8category;
    DtNetU8    nomenclatureVersion;
    DtNetU16nomenclature;
} DtNetRadioEntityType;

typedef struct DtNetEntityType
{
    DtNetU8DtEntityKind;
    DtNetU8DtDomain;
    DtNetU16DtCountry;
    DtNetU8DtCategory;
    DtNetU8DtSubCategory;
    DtNetU8DtSpecific;
    DtNetU8DtExtra;
} DtNetEntityType;

typedef struct DtNetBurstDescriptor
{
    DtNetEntityType DtMunition;
    DtNetU16DtWarhead;
    DtNetU16DtFuze;
    DtNetU16 DtQuantity;
    DtNetU16 DtRate;
} DtNetBurstDescriptor;

typedef struct DtNetSimulationAddress
{
    DtNetU16DtSite;
    DtNetU16 DtHost;
} DtNetSimulationAddress;

typedef struct DtNetEntityIdentifier
{
    DtNetSimulationAddressDtSimulator;
    DtNetU16DtEntity;
} DtNetEntityIdentifier;

typedef struct DtNetEventId
{
    DtNetSimulationAddressDtSimulator;
    DtNetU16 DtEvent;
} DtNetEventId;

typedef struct DtNetEulerAngles
{

```

```
    DtNetFloat32    psi; /* Counter clockwise about z */
    DtNetFloat32    theta; /* Counter clockwise about y */
    DtNetFloat32    phi; /* Counter clockwise about x */
} DtNetEulerAngles;

typedef DtNetFloat32 DtNet32Vector[3];
typedef DtNetFloat64 DtNet64Vector[3];

#elif
#error Must have C++ to use NetStructs.h on little endian machine.
#endif

/*****
These structures don't need to be C++ classes even on little endian
machines.
*****/

#if DtHLA

#if DtRPR016

typedef struct DtNetArtParamRecord
{
    DtNetEnum        partType;
    DtNetU8           change;
    DtNetU16          partAttachedTo;
    DtNetU32          paramType;
    DtNetFloat32      value;
} DtNetArtParamRecord;

#else    // !DtRPR016

typedef struct DtNetArtParamRecord
{
#if DtDISVERS != 3
    DtNetU8          artOrAttached;
    DtNetU8          DtChange;
#else
    DtNetU16          DtChange;
#endif
    DtNetU16          DtAttachedTo;
    DtNetU32          DtType;
    DtNet64Bits       DtValue;
} DtNetArtParamRecord;

#endif    // DtRPR016

#endif    // DtHLA

#if !DtHLA
```



```
typedef struct DtNetArtParamRecord
{
    #if DtDISVERS != 3
        DtNetU8      artOrAttached;
        DtNetU8      DtChange;
    #else
        DtNetU16     DtChange;
    #endif
        DtNetU16     DtAttachedTo;
        DtNetU32     DtType;
        DtNet64Bits  DtValue;
} DtNetArtParamRecord;

#endif    // !DtHLA

#if DtHLA

typedef DtNetArtParamRecord DtArticulatedParts;
typedef DtNetArtParamRecord DtNetArtPartRecord;

#endif    // DtHLA

#if !DtHLA

typedef DtNetArtParamRecord DtArticulatedParts;
typedef DtNetArtParamRecord DtNetArtPartRecord;

#endif    // !DtHLA

typedef struct DtNetAttachedPart
{
    #if DtDISVERS != 3
        DtNetU8      myArtOrAttached;
        DtNetU8      myChange;
    #else
        DtNetU16     myChange;
    #endif
        DtNetU16     myAttachedTo;
        DtNetU32     myStation;
        DtNetEntityType myType;
} DtNetAttachedPart;

typedef struct DtNetAngularVelocity
{
    DtNetFloat32 wx;
    DtNetFloat32 wy;
    DtNetFloat32 wz;
} DtNetAngularVelocity;
```

```
typedef struct DtNetLinearVelocity
{
    DtNetFloat32 DtX;
    DtNetFloat32 DtY;
    DtNetFloat32 DtZ;
} DtNetLinearVelocity;

typedef struct DtNetWorldCoordinates
{
    DtNetFloat64 DtX;
    DtNetFloat64 DtY;
    DtNetFloat64 DtZ;
} DtNetWorldCoordinates;

typedef struct DtNetEntityCoordinates
{
    DtNetFloat32 DtX;
    DtNetFloat32 DtY;
    DtNetFloat32 DtZ;
} DtNetEntityCoordinates;

typedef struct DtNetTopoUpCoord
{
    DtNetFloat32 East;
    DtNetFloat32 North;
    DtNetFloat32 Up;
} DtNetTopoUpCoord;

typedef struct
{
    DtNetInt32hour;
    DtNetU32hourFrac;
} DtNetClockTime;

#ifdef __cplusplus

DtTime DtClockTime2AbsTime(const DtNetClockTime *clock);
void DtAbsTime2ClockTime(DtTime absTime, DtNetClockTime *clock);

#endif
```

netTimeStamp.h

```

#include "vlTime.h"
#include "tStampType.h"
#include "NetTypes.h"

// Host representation of a timeStamp
typedef DtU32 DtTimeStamp;

// class DtNetTimeStamp:
//
// Instances of DtNetTimeStamp are network representations of a
// DIS time stamp. This class has NO VIRTUAL FUNCTIONS, since it
// is a DtNetType - whose exact byte layout is relied on by
// PDU classes, etc.

class DT_DLL_VL DtNetTimeStamp
{
public:

    // default constructor - creates an instance with a time of zero
    // and timestamp type of relative.
    DtNetTimeStamp();

    // Creates an instance with the time set to time (specified in
    // seconds) modulo an hour, and the timestamp type set to type.
    DtNetTimeStamp(DtTime time,
        DtTimeStampType type = DtTimeStampRelative);

    // destructor
    ~DtNetTimeStamp();

    // copy constructor
    DtNetTimeStamp(const DtNetTimeStamp& orig);

    // assignment operator
    DtNetTimeStamp& operator=(const DtNetTimeStamp& orig);

    // Cast to a DtU32 representing the whole timestamp
    // Useful mainly for backwards compatibility for applications
    // that used to cast DtNetTimeStamp to DtU32 before accessing it.
    operator DtU32 () const;

    // Get the time contained in the timestamp (in seconds).
    // The value returned will be between 0.0 and 3600.0 seconds.
    DtTime time() const;

    // Get the time contained in the timestamp to time % an hour.
    // time is specified in seconds.

```

```
void setTime(DtTime time);

// Set/Get timestamp type
void setType(DtTimeStampType type);
DtTimeStampType type() const;

protected:

    DtNetU32 myRep;

protected:

    static double theHourFracPerHour;
    static double theHourPerHourFrac;
};
```

NetTypes.h

```

#ifndef _NetTypes
#define _NetTypes

#include "vlconfig.h"

#include "MachineTypes.h"

typedef DtU8DtNetU8;
typedef DtInt8DtNetInt8;

#if ! DT_LITTLE_ENDIAN

/*****
 * Big Endian Machines.
 *****/

typedef DtU16DtNetU16;
typedef DtU32DtNetU32;
typedef DtInt16DtNetInt16;
typedef DtInt32DtNetInt32;
typedef DtFloat32DtNetFloat32;
typedef DtFloat64DtNetFloat64;

#else

/*****
 *
 * Little Endian (i.e., Brain-dead) Machines.
 *
 * For example, the Intel 80x86 series.
 *
 *****/

/*
 * Network-order unsigned 16-bit quantity.
 */

class DT_DLL_VLUTIL DtNetU16
{
public:

    DtNetU8msbyte; /* Most Significant Byte */
    DtNetU8lsbyte; /* Least Significant Byte */

    DtNetU16();
    DtNetU16(DtU16);
    DtNetU16(const DtNetU16&);

```

```
        DtU16 operator = (DtU16);
        operator DtU16() const;
};
/*
 * Network-order signed 16-bit quantity.
 */

class DT_DLL_VLUTIL DtNetInt16
{
public:

    DtNetInt8msbyte;
    DtNetInt8lsbyte;

    DtNetInt16();
    DtNetInt16(DtInt16);
    DtNetInt16(const DtNetInt16&);
    DtInt16 operator = (DtInt16);
    operator DtInt16() const;
};

/*
 * Network-order unsigned 32-bit quantity.
 */

class DT_DLL_VLUTIL DtNetU32
{
public:

    DtNetU16msword; /* Most Significant Word */
    DtNetU16lsword; /* Least Significant Word */

    DtNetU32();
    DtNetU32(DtU32);
    DtNetU32(const DtNetU32&);
    DtU32 operator = (DtU32);
    operator DtU32() const;
};

/*
 * Network-order signed 32-bit quantity.
 */

class DT_DLL_VLUTIL DtNetInt32
{
public:

    DtNetInt16msword; /* Most Significant Word */
    DtNetInt16lsword; /* Least Significant Word */
```

```

    DtNetInt32();
    DtNetInt32(DtInt32);
    DtNetInt32(const DtNetInt32&);
    DtInt32 operator = (DtInt32);
    operator DtInt32() const;
};

/*
 * Network-order 32-bit IEEE-754 floating-point quantity.
 */

class DT_DLL_VLUTIL DtNetFloat32
{
public:

    DtNetU32bits;

    DtNetFloat32();
    DtNetFloat32(DtFloat32);
    DtNetFloat32(const DtNetFloat32&);
    DtFloat32 operator = (DtFloat32);
    operator DtFloat32() const;
};

/*
 * Network-order 64-bit IEEE-754 floating-point quantity.
 */

class DT_DLL_VLUTIL DtNetFloat64
{
public:

    DtNetU32mslong; /* Most Significant Long */
    DtNetU32lslong; /* Least Significant Least */

    DtNetFloat64();
    DtNetFloat64(DtFloat64);
    DtNetFloat64(const DtNetFloat64&);
    DtFloat64 operator = (DtFloat64);
    operator DtFloat64() const;
};

#endif

/*****
 * You should probably use either of the two above schemes.
 *****/

```

nullFedAmb.h

```
// Adapted from the DMSO RTI 1.3.4 file NullFederateAmbassador.hh.
// This class provides empty definitions for the pure virtual func-
// tions
// declared as members of RTI::FederateAmbassador.

#ifndef DtNullFederateAmbassador_H_
#define DtNullFederateAmbassador_H_

#include <RTI.hh>
#include "MachineTypes.h"

class DT_DLL_VL DtNullFederateAmbassador : public RTI::FederateAm-
bassador
{
public:

DtNullFederateAmbassador() {}
virtual ~DtNullFederateAmbassador() {}

////////////////////////////////////
// Federation Management Services //
////////////////////////////////////

virtual void synchronizationPointRegistrationSucceeded (
    const char *label) // supplied C4)
throw (
    RTI::FederateInternalError) {}

virtual void synchronizationPointRegistrationFailed (
    const char *label) // supplied C4)
throw (
    RTI::FederateInternalError) {}

virtual void announceSynchronizationPoint (
    const char *label, // supplied C4
    const char *tag)   // supplied C4
throw (
    RTI::FederateInternalError) {}

virtual void federationSynchronized (
    const char *label) // supplied C4)
throw (
    RTI::FederateInternalError) {}

virtual void initiateFederateSave (
    const char *label) // supplied C4
throw (
```



```

    RTI::UnableToPerformSave,
    RTI::FederateInternalError) {}

virtual void federationSaved ()
throw (
    RTI::FederateInternalError) {}

virtual void federationNotSaved ()
throw (
    RTI::FederateInternalError) {}

virtual void requestFederationRestoreSucceeded (
    const char *label) // supplied C4
throw (
    RTI::FederateInternalError) {}

virtual void requestFederationRestoreFailed (
    const char *label,
    const char *reason) // supplied C4
throw (
    RTI::FederateInternalError) {}

virtual void federationRestoreBegun ()
throw (
    RTI::FederateInternalError) {}

virtual void initiateFederateRestore (
    const char          *label,    // supplied C4
    RTI::FederateHandle handle)    // supplied C1
throw (
    RTI::SpecifiedSaveLabelDoesNotExist,
    RTI::CouldNotRestore,
    RTI::FederateInternalError) {}

virtual void federationRestored ()
throw (
    RTI::FederateInternalError) {}

virtual void federationNotRestored ()
throw (
    RTI::FederateInternalError) {}

////////////////////////////////////
// Declaration Management Services //
////////////////////////////////////

virtual void startRegistrationForObjectClass (
    RTI::ObjectClassHandle theClass) // supplied C1
throw (
    RTI::ObjectClassNotPublished,

```

```
    RTI::FederateInternalError) {}

virtual void stopRegistrationForObjectClass (
    RTI::ObjectClassHandle theClass) // supplied C1
throw (
    RTI::ObjectClassNotPublished,
    RTI::FederateInternalError) {}

virtual void turnInteractionsOn (
    RTI::InteractionClassHandle theHandle) // supplied C1
throw (
    RTI::InteractionClassNotPublished,
    RTI::FederateInternalError) {}

virtual void turnInteractionsOff (
    RTI::InteractionClassHandle theHandle) // supplied C1
throw (
    RTI::InteractionClassNotPublished,
    RTI::FederateInternalError) {}

////////////////////////////////////////
// Object Management Services //
////////////////////////////////////////

virtual void discoverObjectInstance (
    RTI::ObjectHandle theObject, // supplied C1
    RTI::ObjectClassHandle theObjectClass, // supplied C1
    const char* theObjectName) // supplied C4
throw (
    RTI::CouldNotDiscover,
    RTI::ObjectClassNotKnown,
    RTI::FederateInternalError) {}

virtual void reflectAttributeValues (
    RTI::ObjectHandle theObject, // supplied
C1
    const RTI::AttributeHandleValuePairSet& theAttributes, // supplied
C4
    const RTI::FedTime& theTime, // supplied
C1
    const char* theTag, // supplied C4
    RTI::EventRetractionHandle theHandle) // supplied
C1
throw (
    RTI::ObjectNotKnown,
    RTI::AttributeNotKnown,
    RTI::FederateOwnsAttributes,
    RTI::InvalidFederationTime,
    RTI::FederateInternalError) {}
```

```

virtual void reflectAttributeValues (
    RTI::ObjectHandle          theObject,      // supplied
C1
    const RTI::AttributeHandleValuePairSet& theAttributes, // supplied
C4
    const char                  *theTag)      // supplied C4
throw (
    RTI::ObjectNotKnown,
    RTI::AttributeNotKnown,
    RTI::FederateOwnsAttributes,
    RTI::FederateInternalError) {}

// 4.6
virtual void receiveInteraction (
    RTI::InteractionClassHandle theInteraction, // supplied
C1
    const RTI::ParameterHandleValuePairSet& theParameters, //
supplied C4
    const RTI::FedTime&               theTime,      // supplied
C4
    const char                        *theTag,      // supplied
C4
    RTI::EventRetractionHandle       theHandle)    // supplied
C1
throw (
    RTI::InteractionClassNotKnown,
    RTI::InteractionParameterNotKnown,
    RTI::InvalidFederationTime,
    RTI::FederateInternalError) {}

virtual void receiveInteraction (
    RTI::InteractionClassHandle theInteraction, // supplied
C1
    const RTI::ParameterHandleValuePairSet& theParameters, //
supplied C4
    const char                        *theTag)      // supplied
C4
throw (
    RTI::InteractionClassNotKnown,
    RTI::InteractionParameterNotKnown,
    RTI::FederateInternalError) {}

virtual void removeObjectInstance (
    RTI::ObjectHandle          theObject, // supplied C1
    const RTI::FedTime&        theTime,  // supplied C4
    const char                 *theTag,   // supplied C4
    RTI::EventRetractionHandle theHandle) // supplied C1
throw (
    RTI::ObjectNotKnown,
    RTI::InvalidFederationTime,

```

```
RTI::FederateInternalError) {}

virtual void removeObjectInstance (
    RTI::ObjectHandle      theObject, // supplied C1
    const char             *theTag)   // supplied C4
throw (
    RTI::ObjectNotKnown,
    RTI::FederateInternalError) {}

virtual void attributesInScope (
    RTI::ObjectHandle      theObject,      // supplied C1
    const RTI::AttributeHandleSet& theAttributes) // supplied C4
throw (
    RTI::ObjectNotKnown,
    RTI::AttributeNotKnown,
    RTI::FederateInternalError) {}

virtual void attributesOutOfScope (
    RTI::ObjectHandle      theObject,      // supplied C1
    const RTI::AttributeHandleSet& theAttributes) // supplied C4
throw (
    RTI::ObjectNotKnown,
    RTI::AttributeNotKnown,
    RTI::FederateInternalError) {}

virtual void provideAttributeValueUpdate (
    RTI::ObjectHandle      theObject,      // supplied C1
    const RTI::AttributeHandleSet& theAttributes) // supplied C4
throw (
    RTI::ObjectNotKnown,
    RTI::AttributeNotKnown,
    RTI::AttributeNotOwned,
    RTI::FederateInternalError) {}

virtual void turnUpdatesOnForObjectInstance (
    RTI::ObjectHandle      theObject,      // supplied C1
    const RTI::AttributeHandleSet& theAttributes) // supplied C4
throw (
    RTI::ObjectNotKnown,
    RTI::AttributeNotOwned,
    RTI::FederateInternalError) {}

virtual void turnUpdatesOffForObjectInstance (
    RTI::ObjectHandle      theObject,      // supplied C1
    const RTI::AttributeHandleSet& theAttributes) // supplied C4
throw (
    RTI::ObjectNotKnown,
    RTI::AttributeNotOwned,
    RTI::FederateInternalError) {}
```

```

////////////////////////////////////
// Ownership Management Services //
////////////////////////////////////

virtual void requestAttributeOwnershipAssumption (
    RTI::ObjectHandle      theObject,          // supplied C1
    const RTI::AttributeHandleSet& offeredAttributes, // supplied C4
    const char             *theTag)           // supplied C4
throw (
    RTI::ObjectNotKnown,
    RTI::AttributeNotKnown,
    RTI::AttributeAlreadyOwned,
    RTI::AttributeNotPublished,
    RTI::FederateInternalError) {}

virtual void attributeOwnershipDivestitureNotification (
    RTI::ObjectHandle      theObject,          // supplied C1
    const RTI::AttributeHandleSet& releasedAttributes) // supplied C4
throw (
    RTI::ObjectNotKnown,
    RTI::AttributeNotKnown,
    RTI::AttributeNotOwned,
    RTI::AttributeDivestitureWasNotRequested,
    RTI::FederateInternalError) {}

virtual void attributeOwnershipAcquisitionNotification (
    RTI::ObjectHandle      theObject,          // supplied C1
    const RTI::AttributeHandleSet& securedAttributes) // supplied C4
throw (
    RTI::ObjectNotKnown,
    RTI::AttributeNotKnown,
    RTI::AttributeAcquisitionWasNotRequested,
    RTI::AttributeAlreadyOwned,
    RTI::AttributeNotPublished,
    RTI::FederateInternalError) {}

virtual void attributeOwnershipUnavailable (
    RTI::ObjectHandle      theObject,          // supplied C1
    const RTI::AttributeHandleSet& theAttributes) // supplied C4
throw (
    RTI::ObjectNotKnown,
    RTI::AttributeNotKnown,
    RTI::AttributeAlreadyOwned,
    RTI::AttributeAcquisitionWasNotRequested,
    RTI::FederateInternalError) {}

virtual void requestAttributeOwnershipRelease (
    RTI::ObjectHandle      theObject,          // supplied C1
    const RTI::AttributeHandleSet& candidateAttributes, // supplied C4
    const char             *theTag)           // supplied C4

```

```
throw (
    RTI::ObjectNotKnown,
    RTI::AttributeNotKnown,
    RTI::AttributeNotOwned,
    RTI::FederateInternalError) {}

virtual void confirmAttributeOwnershipAcquisitionCancellation (
    RTI::ObjectHandle      theObject,          // supplied C1
    const RTI::AttributeHandleSet& theAttributes) // supplied C4
throw (
    RTI::ObjectNotKnown,
    RTI::AttributeNotKnown,
    RTI::AttributeAlreadyOwned,
    RTI::AttributeAcquisitionWasNotCanceled,
    RTI::FederateInternalError) {}

virtual void informAttributeOwnership (
    RTI::ObjectHandle      theObject,          // supplied C1
    RTI::AttributeHandle theAttribute,         // supplied C1
    RTI::FederateHandle theOwner)             // supplied C1
throw (
    RTI::ObjectNotKnown,
    RTI::AttributeNotKnown,
    RTI::FederateInternalError) {}

virtual void attributeIsNotOwned (
    RTI::ObjectHandle      theObject,          // supplied C1
    RTI::AttributeHandle theAttribute)         // supplied C1
throw (
    RTI::ObjectNotKnown,
    RTI::AttributeNotKnown,
    RTI::FederateInternalError) {}

virtual void attributeOwnedByRTI (
    RTI::ObjectHandle      theObject,          // supplied C1
    RTI::AttributeHandle theAttribute)         // supplied C1
throw (
    RTI::ObjectNotKnown,
    RTI::AttributeNotKnown,
    RTI::FederateInternalError) {}

////////////////////////////////////
// Time Management Services //
////////////////////////////////////

virtual void timeRegulationEnabled (
    const RTI::FedTime& theFederateTime) // supplied C4
throw (
    RTI::InvalidFederationTime,
    RTI::EnableTimeRegulationWasNotPending,
```

```
    RTI::FederateInternalError) {}

virtual void timeConstrainedEnabled (
    const RTI::FedTime& theFederateTime) // supplied C4
throw (
    RTI::InvalidFederationTime,
    RTI::EnableTimeConstrainedWasNotPending,
    RTI::FederateInternalError) {}

virtual void timeAdvanceGrant (
    const RTI::FedTime& theTime) // supplied C4
throw (
    RTI::InvalidFederationTime,
    RTI::TimeAdvanceWasNotInProgress,
    RTI::FederateInternalError) {}

virtual void requestRetraction (
    RTI::EventRetractionHandle theHandle) // supplied C1
throw (
    RTI::EventNotKnown,
    RTI::FederateInternalError) {}
};
```

objClassDesc.h

```
#if DtHLA

#include "RTI.hh"
#include "MachineTypes.h"

#include "chooseDefines.h"

// class DtObjClassDesc:
//
// An instance of DtObjClassDesc is a "class descriptor" for an HLA
// Object
// class. A list of all DtObjClassDescs for the various classes in
// a FOM can
// be obtained from a DtFom. A DtObjClassDesc can tell you the set
// of
// attributes associated with the class, the class's handle, name,
// and base
// class. Publication, subscription and update requests are done
// through the
// DtObjClassDesc as well, and the DtObjClassDesc maintains informa-
// tion on
// which attributes of the class are subscribed to by remote feder-
// ates. By
// registering callbacks on start/stopUpdates, a DtObjClassDesc can
// tell you
// which attributes of the class have been subscribed to by other
// federates.

class DT_DLL_VL DtObjClassDesc
{
public:

    // Constructor. Adds the base class' attributes to our attribute
    // set.
    // Used when reading FED file.
    DtObjClassDesc(const char* className,
        DtObjClassDesc* base,
        RTI::ObjectClassHandle handle,
        DtExerciseConn* exConn);

    // destructor
    virtual ~DtObjClassDesc();

    // Add an attribute (Used when reading FED file).
    virtual void addAttr(const char* attrName);

    // Get the object's class handle.
```

```

virtual RTI::ObjectClassHandle handle() const;

// Get the object's class handle.
virtual char* name() const;

// Return a pointer to the base class's descriptor, NULL if none
virtual const DtObjClassDesc* baseClass() const;

// Returns 1 if the class is of class hand or a subclass of hand,
// 0 otherwise
virtual int isOfClass(RTI::ObjectClassHandle hand) const;

// Get the number of attributes associated with this object,
// including privilege to delete (usually handle 0).
virtual int numAttributes() const;

// Get the set of valid attribute handles for this class.
Includes
// attributes inherited from base classes.
virtual const RTI::AttributeHandleSet& attrHandleSet() const;

// Get the set of non-inherited attribute handles for this class.
virtual const RTI::AttributeHandleSet& ownAttrHandles() const;

// Returns DtTrue if the class is currently needed by the federa-
tion,
// i.e. we've received a startRegistrationForObjectClass call for
the
// class. We will only receive those calls if the class has been
published
// using DtObjClassDesc::publish.
virtual DtBoolean classNeededByFederation();

// Publishes the class, along with the attributes specified by
hSet (or all
// attributes if no hSet is given). Also, registers start and
stopUpdates
// callbacks on the published classes. Can be called multiple
times to
// change the set of published classes.
virtual void publish();
virtual void publish(const RTI::AttributeHandleSet& hSet);

// Subscribes to the class, along with the attributes specified
by hSet (or
// all attributes if no hSet is given).
virtual void subscribe(const RTI::AttributeHandleSet& hSet);
virtual void unsubscribe();

// Unsubscribe to the class.

```

```
virtual void unsubscribe();

// Request updates of all objects of this class, include the
attributes
// specified by hSet (or all subscribed attributes if no hSet is
given).
virtual void requestUpdates();
virtual void requestUpdates(const RTI::AttributeHandleSet& hSet);

// Returns the set of published attributes. For this to work
correctly, do
// not publish directly through the RTI. Instead, use
// DtObjClassDesc::publish.
virtual const RTI::AttributeHandleSet& publishedAttrs() const;

// Return true if we have ever published the class using
// DtObjClassDesc::publish, false otherwise.
virtual DtBoolean everPublished() const;

// Returns the set of subscribed attributes. For this to work
correctly,
// do not publish directly through the RTI. Instead, use
// DtObjClassDesc::subscribe.
virtual const RTI::AttributeHandleSet& subscribedAttrs() const;

// Return true if we have ever subscribed to the class using
// DtObjClassDesc::subscribe, false otherwise.
virtual DtBoolean everSubscribed() const;

// Get my RTI ambassador
virtual DtVrlRtiAmbassador* rtiAmb();

// Get my federate ambassador
virtual DtVrlFederateAmbassador* fedAmb();

public:

    static int maxAttributes();
    static void setMaxAttributes(int num);

protected:

    // Callback for handling startRegistration RTI callback
    static void startRegCb(RTI::ObjectClassHandle h,
        void* usr);

    // Callback for handling stopRegistration RTI callback
    static void stopRegCb(RTI::ObjectClassHandle h,
        void* usr);
```

protected:

```
// Callback for handling startRegistration RTI callback
virtual void processStartReg(RTI::ObjectClassHandle h);

// Callback for handling stopRegistration RTI callback
virtual void processStopReg(RTI::ObjectClassHandle h);

// Casts away const-ness. Use with caution.
virtual DtObjClassDesc* self() const;
```

protected:

```
char* myName;
RTI::ObjectClassHandle myHandle;
RTI::AttributeHandleSet* myAttrs;
RTI::AttributeHandleSet* myOwnAttrs;
RTI::AttributeHandleSet* mySubAttrs;
RTI::AttributeHandleSet* myPubAttrs;
DtBoolean myClassNeeded;
DtObjClassDesc* myBase;
DtExerciseConn* myExConn;
DtBoolean myEverPublished;
DtBoolean myEverSubscribed;
```

protected:

```
    static int theMaxAttrs;
};
#ifdef DtRTI103
#else
#endif

#endif
```

objDesList.h

```
// class DtGlobalObjectDesignatorList:
//
// Instances of DtGlobalObjectDesignatorList are list of globally
// significant object identifiers.

#include "hostStructs.h"

class DT_DLL_VL DtGlobalObjectDesignatorList
{
public:

    // default constructor
    DtGlobalObjectDesignatorList();

    // destructor
    virtual ~DtGlobalObjectDesignatorList();

    // copy constructor
    DtGlobalObjectDesignatorList(const DtGlobalObjectDesignatorList&
orig);

    // assignment operator
    DtGlobalObjectDesignatorList& operator=(
        const DtGlobalObjectDesignatorList& orig);

    // equivlance operators
    int operator==(const DtGlobalObjectDesignatorList& other) const;
    int operator!=(const DtGlobalObjectDesignatorList& other) const;

    // Set/Get number of objects. Setting to a number lower than
    // the current number will remove trailing objects from the list.
    virtual void setNumObjects(int num);
    virtual int numObjects() const;

    // Set index'th object designator.
    virtual DtBoolean setObject(int index,
        const DtGlobalObjectDesignator& des);

    // Get index'th object designator. Set valid to DtFalse if bad
    index,
    // DtTrue otherwise.
    virtual const DtGlobalObjectDesignator& object(
        int index, DtBoolean* valid = NULL) const;

    // Print data.
    virtual void print() const;
```

```
// Set from network representation.
virtual void setFromNet(const void* netRep, int length);

// Returns network representation, fills size with size of
netRep.
// Return value is pointer to internal object with same lifetime
as this.
// Do not delete.
virtual const void* netRep(int* size = NULL) const;

// Returns size of network representation.
virtual int netSize() const;

protected:

    int myNum;
    DtGlobalObjectDesignator* myObjs;
};
```

objectType.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"

#include "packets/objectTypeP.h"

// class DtObjectType:
//
// DtObjectType is a class which contains information identifying
// the type of a synthetic environment object.

class DT_DLL_VL DtObjectType
{
public:

    // Constructors
    DtObjectType();
    DtObjectType(const DtObjectType& orig);

    // Assignment operator
    DtObjectType& operator=(const DtObjectType& orig);
    DtObjectType& operator=(const DtNetObjectType& orig);

    // Destructor
    virtual ~DtObjectType();

    // Set/Get the domain
    virtual void setDomain(DtU8 value);
    virtual DtU8 domain() const;

    // Set/Get the kind
    virtual void setKind(DtU8 value);
    virtual DtU8 kind() const;

    // Set/Get the category
    virtual void setCategory(DtU8 value);
    virtual DtU8 category() const;

    // Set/Get the subcategory
    virtual void setSubcategory(DtU8 value);
    virtual DtU8 subcategory() const;

    // Print the information contained in the record
    virtual void print() const;
```

```
// Load/Unload data from a packet record
virtual void unloadNetRecord(const DtNetObjectType& record);
virtual void loadNetRecord(DtNetObjectType& record) const;

protected:

    // Data members:
    DtU8          myDomain;
    DtU8          myKind;
    DtU8          myCategory;
    DtU8          mySubcategory;

};
#endif
#endif
```

objIdHashKey.h

```
#include "baseHashKey.h"
#include "hostStructs.h"

// class DtObjectIdHashKey:
//
// Instances of DtObjectIdHashKey are hash keys made from DtObject-
// tIds.
// They can be used with DtBaseHashList to store arbitrary pieces of
// data hashed by DtObjectId.

class DT_DLL_VL DtObjectIdHashKey : public DtBaseHashKey
{
public:

    // default constructor
    DtObjectIdHashKey(const DtObjectId& id);

    // destructor
    virtual ~DtObjectIdHashKey();

    // copy constructor
    DtObjectIdHashKey(const DtObjectIdHashKey& orig);

    // assignment operator
    DtObjectIdHashKey& operator=(const DtObjectIdHashKey& orig);

    // make a copy of this hash key
    virtual DtBaseHashKey* clone() const;

protected:

    // get the string representation of this key
    virtual const DtString& stringRep() const;

    // compute the hashed value for this key
    virtual DtU32 computeHashValue();

protected:

    DtObjectId myId;
    DtString myStr;
};
```


objIdHashLst.h

```

#include "hashlist.h"
#include "hostStructs.h"

// class DtObjectIdHashlist:
//
// Instances of DtObjectIdHashlist are hashlists, where you can
// store associations between DtObjectIds and arbitrary pieces of
// data.

class DT_DLL_VL DtObjectIdHashlist
{
public:
    // default constructor. Pass DtTrue only if the data that you are
    // listing
    // are instances of a class derived from DtListedObject.
    DtObjectIdHashlist(DtBoolean intrusive = DtFalse);

    // destructor
    virtual ~DtObjectIdHashlist();

    // In the event that an item with key already exists, then add
    // will replace the data and move it to the location in list.
    // If your data was allocated, do not add unless you know key is
    // unique, or a memory leak will result.

    // Add an association between key and data to the list
    virtual DtListItem* add(const DtObjectId& key, void* data);

    // Remove the current association involving key from the list.
    // Returns the data that was associated with key.
    virtual void* remove(const DtObjectId& key);

    // Lookup the data associated with key.
    virtual void* lookup(const DtObjectId& key) const;
    virtual void* operator[](const DtObjectId& key) const;

    // Get a DtList containing all of the void* data that you
    // have listed, regardless of keys.
    virtual const DtList* list() const;

protected:
    // copy constructor
    DtObjectIdHashlist(const DtObjectIdHashlist& orig);

    // assignment operator

```

```
DtObjectIdHashlist& operator=(const DtObjectIdHashlist& orig);  
protected:  
    DtList* myList;  
    DtHashlist* myHashlist;  
};  
typedef DtObjectIdHashlist DtObjectIdHashList;
```

objRefCbInfo.h

```

#if DtHLA

#include "RTI.hh"
#include "stateMsg.h"

#include "chooseDefines.h"

// Object reflect attribute values callback signature.
typedef void (*DtObjectReflectAttributeValuesCb)(
    const DtStateMsg& stateMsg,
    DtHlaObject* obj,
    void* usr);

// class DtObjectReflectAttributeValuesCbInfo:
//
// Instances of DtReflectAttributeValuesCbInfo are used for
// providing
// a callback when attributes change for an object in the federa-
// tion.

class DT_DLL_VL DtObjectReflectAttributeValuesCbInfo
{
public:

    // constructor
    DtObjectReflectAttributeValuesCbInfo(
        DtObjectReflectAttributeValuesCb cb,
        void* usr);

    // destructor
    virtual ~DtObjectReflectAttributeValuesCbInfo();

    // copy constructor
    DtObjectReflectAttributeValuesCbInfo(
        const DtObjectReflectAttributeValuesCbInfo& orig);

    // assignment operator
    DtObjectReflectAttributeValuesCbInfo& operator=(
        const DtObjectReflectAttributeValuesCbInfo& orig);

    // Check whether one callback info is equal to another.
    virtual int match(
        const DtObjectReflectAttributeValuesCbInfo* const other)
const;

    // Check whether one callback info is equal to another.
    virtual int operator==(

```

```
        const DtObjectReflectAttributeValuesCbInfo& other) const;

    // Get the callback function.
    virtual const DtObjectReflectAttributeValuesCb& cbFunction()
const;

    // Get the user data.
    virtual void* userData() const;

protected:

    DtObjectReflectAttributeValuesCb myFunc;
    void* myUsr;
};
#endif
```

pairSock.h

```
#include "Socket.h"

class DT_DLL_VLUTIL DtSocketPairSocket : public DtSocket
{
public:
    DtSocketPairSocket();
    DtSocketPairSocket(int sockNum);

    int sendTo(caddr_t packet, size_t size, DtInetAddr addr);
    int recvWithStatus(caddr_t buffptr, size_t size,
                      int *status=NULL);
    int DtRecv(caddr_t *buffptr, int *status);
    int DtRecv(caddr_t buff, size_t size);
    void DtBecomeNonBlocking(void);

    int otherFd() { return ofd; }
    int readFd() { return myFd; }
    int writeFd() { return myFd; }

private:
    int myFd;
    int ofd;
    char buffer[1500];
};
```

pdu.h

```
#include <stdlib.h>
#include "packets/pduHeader.h"
#include "disEnums.h"
#include "tStampType.h"
#include "baseMsg.h"

#define DtSTATUS_OK      0
#define DtSTATUS_NOT_OK 1

extern DT_DLL_VL int DtProtocolVersionToSend;
extern DT_DLL_VL int DtProtocolVersionToRecvMin;
extern DT_DLL_VL int DtProtocolVersionToRecvMax;

class DT_DLL_VL DtPdu : public DtBaseMsg
{
public:

    // Destructor
    virtual ~DtPdu();

    // Copy constructor
    DtPdu(const DtPdu &pdu,
          DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // Assignment operator
    DtPdu &operator=(const DtPdu &pdu);

    // Older function that returns a pointer to the network representation.
    // The newer DtBaseMsg::netRep should generally be used instead.
    virtual void* packet() const;

    // Get PDU status. DtSTATUS_OK means it's good. Other values
    // indicate a bad PDU. Return codes are derived PDU-dependent.
    virtual int status() const;

    // Set/Get protocol version from PDU header.
    virtual void setVersion(int vers);
    virtual int protocolVersion() const;

    // Set/Get exerciseID in PDU header.
    virtual void setExerciseId(int id);
    virtual int exerciseId() const;

    // Returns PDU kind from PDU header
    virtual DtPduKind kind() const;
```

```

#if DtDISVERS != 3
    // Returns protocol family from PDU header.
    virtual DtProtocolFamily protocolFamily() const;
#endif

    // Set the PDU header's time stamp to reflect time and type.
    // Time is in seconds, but only the time modulo an hour
    // actually gets into the time stamp.
    virtual void setTimeStamp(DtTime time,
        DtTimeStampType type = DtTimeStampRelative);

    // Return the time (in seconds past an hour) encoded in the
    // PDU header's time stamp.
    virtual DtTime timeStamp() const;

    // Return the type of the time stamp, as indicated in PDU header.
    virtual DtTimeStampType timeStampType() const;

    // Since timestamp only represents number of seconds past the
hour,
    // guessTimeValid chooses the hour that would make the timestamp
    // closest to reference time, and returns a time that is times-
tamp
    // seconds past that hour.
    virtual DtTime guessTimeValid(DtTime referenceTime) const;

    // Print the contents of the PDU by calling printHeader
    // and printData.
    virtual void print() const;

    // Print the PDU's header.
    virtual void printHeader() const;

    // Print the PDU's data (not including header).
    // Base DtPdu version just prints hex.
    virtual void printData() const;

protected:

    // Constructor.
    // PDU is not fully constructed until initPdu is called.
    DtPdu();

    // Sets the PDU's status to okStatus.
    virtual void setStatus(int okStatus);

    // Returns a pointer to the PDU's header.
    virtual DtNetPduHeader* netHeader() const;

    // Must be called after default ctor before PDU is completely

```

```
// constructed. Fills in PDU header, and initializes DtBaseMsg.
virtual void initPdu(int size, DtBufferPtr buffer);

// Called by from-net-rep ctor. Initializes DtBaseMsg.
virtual void initPdu(const void* initial, DtBufferPtr buffer);

// Returns the PDU kind associated with the derived PDU class.
virtual DtPduKind internalGetPduKind() const = 0;

#if DtDISVERS != 3
    // Maps PDU kind to ProtocolFamily.
    virtual DtProtocolFamily kindToFamily(DtPduKind kind) const;
#endif

// Called by changeSize. Sets the size in the header to size.
virtual void sizeChanged(int size);

// zeros out all bytes after the PDU header
virtual void clearData();

// Initializes PDU header. Called by initPdu.
virtual void initHeader(DtPduKind kind, int size);

protected:

    int myOKStatus;
};
#define netPdu packet()

#if DtDISVERS != 3
#endif
```


pduCbInfo.h

```

#if !DtHLA

#include "pduCbMgr.h"

// class DtPduCallbackInfo:
//
// Instances of DtPduCallbackInfo are used by DtPduCallbackManager
// to keep track of PDU callback functions.

class DT_DLL_VL DtPduCallbackInfo
{
public:

    // constructor
    DtPduCallbackInfo(DtPduCallbackFcn func, void* usr);

    // destructor
    virtual ~DtPduCallbackInfo();

    // copy constructor
    DtPduCallbackInfo(const DtPduCallbackInfo& orig);

    // assignment operator
    DtPduCallbackInfo& operator=(const DtPduCallbackInfo& orig);

    // Get whether one callback info equals another.
    virtual int match(const DtPduCallbackInfo& other) const;

    // Get the callback function.
    virtual DtPduCallbackFcn func() const;

    // Get the user data.
    virtual void* usr() const;

protected:

    DtPduCallbackFcn myFunc;
    void* myUsr;

};
#endif

```

pduCbMgr.h

```
#if !DtHLA

#include "vllist.h"
#include "pdu.h"

// PDU callback function signature.
typedef void (*DtPduCallbackFcn)(DtPdu* pdu, void* usr);

// class DtPduCallbackManager:
//
// Instances of DtPduCallbackManager are used to manage callbacks
// on the PDU kinds for the exercise connection.

class DT_DLL_VL DtPduCallbackManager
{
public:

    // default constructor
    DtPduCallbackManager();

    // destructor
    virtual ~DtPduCallbackManager();

    // copy constructor
    DtPduCallbackManager(const DtPduCallbackManager& orig);

    // assignment operator
    DtPduCallbackManager& operator=(const DtPduCallbackManager&
orig);

    // Add/Remove a function to be called when a particular PDU kind
    // is received.
    virtual void addPduCallback(DtPduKind k,
        DtPduCallbackFcn fcn, void* usr);
    virtual void removePduCallback(DtPduKind k,
        DtPduCallbackFcn fcn, void* usr);

    // Call the callbacks registered for the PDU's kind.
    virtual void processPdu(const DtPdu& pdu) const;

    // Get the number of callbacks registered for the PDU kind.
    virtual int numCallbacks(DtPduKind k) const;

protected:

    // Add/Remove callback info for a particular PDU kind.
```

```
    virtual void addPduCallback(DtPduKind k, DtPduCallbackInfo*
info);
    virtual void removePduCallback(DtPduKind k, const DtPduCallback-
Info& info);

    // Empty the list.
    // Non-virtual since called in dtor.
    void emptyList();

    // Copy the list.
    // Non-virtual since called in ctor.
    void copyList(const DtPduCallbackManager& orig);

protected:

#define DtMAX_PDU_KINDS 255
    DtList myCbList[DtMAX_PDU_KINDS];
};
#endif
```

pduDatRecSet.h

```
#if !DtHLA

#include "datumRecSet.h"

// An instance of this class is a DtDatumRecordSet that uses a part
// of
// a DtPduWithData as its network representation. Normally, a
// DtDatumRecord manages the memory associated with its network
// representation
// through its DtBaseMsg base class. However, since a DtPduWith-
// Data's
// netRep memory is managed by that DtPduWithData, DtPduDatumRecord-
// Sets must
// defer ITS memory management responsibilities to the DtPduWith-
// Data.
// This is accomplished by overriding changeSize, addBytes and
// deleteBytes
// here, with versions that call the DtPduWithData's versions of the
// functions.
// No one should need to use this class, except for DtPduWithData.

class DT_DLL_VL DtPduDatumRecordSet : public DtDatumRecordSet
{
public:

    // Constructor
    DtPduDatumRecordSet(DtPduWithData* pdu);

    // Destructor
    virtual ~DtPduDatumRecordSet();

    // Copy ctor
    DtPduDatumRecordSet(const DtPduDatumRecordSet& orig);

    // Assignment operator
    DtPduDatumRecordSet& operator=(const DtPduDatumRecordSet& orig);

protected:

    // Changes the size of the datumRecSet, by asking my PDU to change
    // size, then reinitializing to use the right piece of the PDU's
    // buffer.
    virtual void changeSize(int size);

    // Inserts numBytes zeroed-out bytes at the indicated offset
    // into the net rep, by asking my PDU to insert bytes, then
    // reinitializing to use the right piece of the PDU's buffer.
```

```
virtual void insertBytes(int offset, int numBytes);

// Inserts numBytes zeroed-out bytes at the indicated addr
// within the net rep, by asking my PDU to insert bytes, then
// reinitializing to use the right piece of the PDU's buffer.
virtual void insertBytes(void* addr, int numBytes);

// Deletes numBytes bytes at the indicated offset in the net rep,
// by asking my PDU to insert bytes, then reinitializing to use
// the right piece of the PDU's buffer.
virtual void deleteBytes(int offset, int numBytes);

// Deletes numBytes bytes at the indicated addr within the net
rep,
// by asking my PDU to insert bytes, then reinitializing to use
// the right piece of the PDU's buffer.
virtual void deleteBytes(void* addr, int numBytes);

protected:

    DtPduWithData* myPdu;
};
#endif
```

pduFactory.h

```
#if !DtHLA

#include "pdu.h"

// Pdu creation function signature.
typedef DtPdu* (*DtPduCreator)(const DtNetPduHeader* packet,
    DtBufferPtr buffer);

// class DtPduFactory:
//
// Instances of DtPduFactory are used to create PDUs (protocol data
// units) or packets, typically for DIS (as opposed to HLA).

// DtADD_PDU_CREATOR adds a PDU classes static create function to
// a PDU factory. For example, DtADD_PDU_CREATOR(EntityState)
// produces:
// addCreator(DtEntityStatePduKind, DtEntityStatePdu::create);
#define DtADD_PDU_CREATOR(x) \
    addCreator(Dt##x##PduKind, Dt##x##Pdu::create);

class DT_DLL_VL DtPduFactory
{
public:

    // default constructor
    DtPduFactory();

    // destructor
    virtual ~DtPduFactory();

    // copy constructor
    DtPduFactory(const DtPduFactory& orig);

    // assignment operator
    DtPduFactory& operator=(const DtPduFactory& orig);

    // Create the specified pdu.
    // Note: Caller is responsible for deleting.
    // Returns NULL if pdu creation function is not in the factory.
    virtual DtPdu* createPdu(const DtNetPacket* packet,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // Add a pdu creation function to the factory.
    // The function must return a new'ed pdu object which will
    // be deleted by the user.
    // Returns any previous pdu creation function which was
    // replaced by the add.
```

```
    virtual DtPduCreator addCreator(DtPduKind kind, DtPduCreator
creator);
    // Returns whether packet is appropriate DIS version.

    virtual int packetGood(const DtNetPacket* packet) const;

public:
    static int dfltPacketGood(const DtNetPacket* packet);

protected:

    // myCreators is an array of theMaxPduKinds items.
    DtPduCreator* myCreators;

protected:

    static const int theMaxPduKinds;
};
#endif
```

pduWithData.h

```
#if !DtHLA

#include "hostStructs.h"
#include "simanPdu.h"
#include "datumIdType.h"
#include "datumRecSet.h"

#define DtDataStaticSize 40

class DT_DLL_VL DtPduWithData : public DtSimanPdu
{
public:

    // Constructor
    DtPduWithData();

    // returns number of fixed fields
    virtual int numFixedFields() const;

    // returns number of variable length fields
    virtual int numVarFields() const;

    // Returns the datum param found in the index'th variable
    // datum record. Index is between 1 and numRecords. Returns
    // DtDatumParam(0) if index is out of bounds.
    virtual DtDatumParam varDatumId(int index) const;

    // Returns the datum param found in the index'th fixed
    // datum record. Index is between 1 and numRecords. Returns
    // DtDatumParam(0) if index is out of bounds.
    virtual DtDatumParam fixedDatumId(int index) const;

    // Returns the datum param found in the index'th fixed or variable
    // datum record. Index is between 1 and numRecords. Returns
    // DtDatumParam(0) if index is out of bounds.
    virtual DtDatumParam datumId(int index) const;
};
```



```

    // Returns the number of bytes of data contained in the value
    // fields of the index'th fixed or variable datum record,
    // depending on whether fixedOrVar is DtFIXED or DtVAR. If
DtFIXED,
    // the answer is always 4.
    virtual int datumSizeBytes(int fixedOrVar, int index) const;

    // Looks through all fixed and variable records for one
containing
    // param as the parameter ID. If it finds one, fills supplied
fixedOrVar
    // pointer with DtFIXED or DtVAR, and index with the index to
indicate
    // which record was found, and fills retcode with DtDATUM_VALID.
If param
    // was not found, retcode is filled with DtDATUM_BAD_ID, and
index and
    // fixedOrVar are invalid.
    virtual void getIndex(DtDatumParam param, int *fixedOrVar, int
*index,
        int *retcode = NULL) const;

// ***** INSPECTING DATUM VALUES BY INDEX *****
// If you know the data type of the value contained in a particular
// record, use one of the following to return it. fixedOrVar
(either DtFIXED
// or DtVAR) and index (between 1 and numRecords) indicate which
record
// you are inspecting. datumValByteArray is used when the data
type is
// a character string, or something other than a long, unsigned
long, float
// or double. It returns a pointer to a character string containing
the
// exact bytes in the PDU's network representation that represent
the value
// being inspected. The supplied retcode pointer will be filled
with
// DtDATUM_VALID if the value was successfully inspected, and
DtDATUM_BAD_LENGTH
// if either the index was invalid, or the size of the data indi-
cated that
// it could not be interpreted as the chosen data type.

    virtual DtFloat32 datumValFloat32(int fixedOrVar, int index,
        int *retcode = NULL) const;
    virtual DtFloat64 datumValFloat64(int fixedOrVar, int index,
        int *retcode = NULL) const;
    virtual unsigned long datumValUnsigned32(int fixedOrVar, int
index,

```

```
        int *retcode = NULL) const;
    virtual long    datumValInt32(int fixedOrVar, int index,
        int *retcode = NULL) const;
    virtual const char*    datumValByteArray(int fixedOrVar, int
index) const;

// ***** INSPECTING DATUM VALUES BY PARAM ID *****
// If you know the data type of the value contained in a particular
// record, use one of the following to return it. The particular
// record
// inspected will be the first one containing param as the datum ID.
// datumValByteArray is used when the data type is
// a character string, or something other than a long, unsigned
// long, float
// or double. It returns a pointer to a character string containing
// the
// exact bytes in the PDU's network representation that represent
// the value
// being inspected. The supplied retcode pointer will be filled
// with
// DtDATUM_VALID if the value was successfully inspected, and
// DtDATUM_BAD_LENGTH
// if either param was not found, or the size of the data indicated
// that
// it could not be interpreted as the chosen data type.

    virtual DtFloat32    datumValFloat32(DtDatumParam param,
        int *retcode = NULL) const;
    virtual DtFloat64    datumValFloat64(DtDatumParam param,
        int *retcode = NULL) const;
    virtual unsigned long datumValUnsigned32(DtDatumParam param,
        int *retcode = NULL) const;
    virtual long    datumValInt32(DtDatumParam param,
        int *retcode = NULL) const;
    virtual const char*    datumValByteArray(DtDatumParam param,
        int *retcode = NULL) const;

// ***** MUTATOR FUNCTIONS *****

    // Initialize the data section of the PDU to reflect the fixed
// parameters
    // and variable parameters and sizes indicated. fixedParams is a
// NULL-
    // terminated array of DtDatumParams. specList is an array of
// DtVarDatumSpecs
    // terminated with the DtNullVarDatumSpec constant. This call
// can be made
    // instead of calls to setNumVarDatumRecords, setNumFixedDatum-
// Record,
    // and repeated calls to setDatumParam and setVarDataBytes.
```

```

virtual void initDataSection(const DtDatumParam *fixedParams,
                             const DtVarDatumSpec *specList);

    // Sets the number of fixed or variable fields to newNum. Previously
    // existing fixed fields with indices greater than newNum are
    // deleted.
    // If newNum exceeds the number of existing fields, trailing new
    // fields
    // are created with zeros as datum params, sizes (for var only)
    // and values.
    virtual void setNumFixedFields(int newNum);
    virtual void setNumVarFields(int newNum);

    // Set the datum ID in the index'th fixed or variable datum record
    // to
    // param, depending on whether fixedOrVar is DtFIXED or DtVAR.
    // Supplied
    // retcode pointer is filled with DtDATUM_VALID if successful, or
    // DtDATUM_
    // BAD_INDEX if the index is not valid.
    virtual void setDatumParam(int fixedOrVar, int index, DtDatumP-
    am param,
        int *retcode = NULL);

    // Allocates length bytes for the index'th variable datum record.
    // Should be called before using one of the setDatumVal functions
    // to actually set the data. Supplied retcode pointer is set to
    // DtDATUM_VALID if successful, or DtDATUM_BAD_INDEX if the index
    // is not valid.
    virtual void setVarDataBytes(int index, int length, int *retcode
    = NULL);

// ***** SETTING DATA VALUES BY INDEX *****
// Use one of the following to set the data value in a fixed or vari-
// able datum
// record, depending on the data type of your data. fixedOrVar
// (either DtFIXED
// or DtVAR) and index (between 1 and numRecords) indicate which
// record
// you are modifying. setDatumValByteArray is used when the data
// type is
// a character string, or something other than a long, unsigned
// long, float
// or double. It takes a pointer to a character string containing
// the
// exact bytes that should be used in the PDU's network representa-
// tion.
// The supplied retcode pointer will be filled with

```

```
// DtDATUM_VALID if the value was successfully set,
DtDATUM_BAD_INDEX if the
// index is invalid, and DtDATUM_BAD_LENGTH if the size of the data
indicated that
// it could not be set from as the chosen data type.

    virtual void setDatumValFloat64(int fixedOrVar, int index,
DtFloat64 val,
        int *retcode = NULL);
    virtual void setDatumValFloat32(int fixedOrVar, int index,
DtFloat32 val,
        int *retcode = NULL);
    virtual void setDatumValInt32(int fixedOrVar, int index, long
val,
        int *retcode = NULL);
    virtual void setDatumValUnsigned32(int fixedOrVar, int index,
unsigned long val,
        int *retcode = NULL);
    virtual void setDatumValByteArray(int fixedOrVar, int index,
const char *val,
        int *retcode = NULL);

// ***** SETTING DATA VALUES BY PARAM ID *****
// Use one of the following to set the data value in a fixed or vari-
able datum
// record, depending on the data type of your data. The particular
record modified
// will be the first one containing param as its parameter ID.
// setDatumValBytesArray is used when the data type is
// a character string, or something other than a long, unsigned
long, float
// or double. It takes a pointer to a character string containing
the
// exact bytes that should be used in the PDU's network representa-
tion.
// The supplied retcode pointer will be filled with
// DtDATUM_VALID if the value was successfully set, DtDATUM_BAD_ID
if param
// is invalid, and DtDATUM_BAD_LENGTH if the size of the data indi-
cated that
// it could not be set from the chosen data type.

    virtual void setDatumValFloat64(DtDatumParam param, DtFloat64
val,
        int *retcode = NULL);
    virtual void setDatumValFloat32(DtDatumParam param, DtFloat32
val,
        int *retcode = NULL);
    virtual void setDatumValInt32(DtDatumParam param, long val,
int *retcode = NULL);
```

```

    virtual void setDatumValUnsigned32(DtDatumParam param, unsigned
long val,
                                     int *retcode = NULL);
    virtual void setDatumValByteArray(DtDatumParam param, const char
*val,
                                     int *retcode = NULL);

    // Implements DtPdu's virtual function to print the PDU's data
    virtual void printData() const;

    // Helper function that prints only the contents of the data
records
    virtual void printDataSection() const;

protected:

    friend class DtPduDatumRecordSet;

    // Returns our datum record set helper object.
    virtual DtDatumRecordSet* datumRecSet();
    // Const version of above
    virtual const DtDatumRecordSet* datumRecSet() const;

    // Returns a pointer to the data section of the PDU - the part
that
    // starts with the numberOfFixedFields field.
    virtual DtNetDataPduTail *dataSection() const = 0;

    // Return a pointer to the index'th fixed data record in the
// network representation. Index is between 1 and numRecords.
    virtual DtNetFixDatum *nthFixedDataRecord(int index) const;

    // Return a pointer to the index'th variable data record in the
// network representation. Index is between 1 and numRecords.
    virtual DtNetVarDatum *nthVarDataRecord(int index) const;

    // Returns a pointer to the index'th fixed or variable record in
the
    // network representation, depending on whether fixedOrVar is
DtFIXED
    // or DtVAR.
    virtual void *datumValPtr(int fixedOrVar, int index) const;

    // Destructor
    virtual ~DtPduWithData();

protected:
    DtDatumRecordSet* myRecSet;
};
#endif

```

physEntDec.h

```
#if DtHLA

#include "baseEntDec.h"

// class DtPhysicalEntityDecoder:
//
// Instances of DtPhysicalEntityDecoder are used to decode a
// physical entity state message into a state repository in an HLA
// (as opposed to DIS) environment.

class DT_DLL_VL DtPhysicalEntityDecoder : public DtBaseEntityDecoder
{
public:

    // constructor
    DtPhysicalEntityDecoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtPhysicalEntityDecoder();

    // copy constructor
    DtPhysicalEntityDecoder(const DtPhysicalEntityDecoder& orig);

    // assignment operator
    DtPhysicalEntityDecoder& operator=(const DtPhysicalEntityDecoder&
orig);

public:

    // Create a DtPhysicalEntityDecoder.
    // User is responsible for deletion.
    static DtHlaStateDecoder* create(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

protected:

    // Add decoding functions for the attributes.
    // Note: Non-virtual since meant to be called from constructors
    to
        //          add decoders.
        void addMyDecoders();

protected:
```

```

    // static decode<AttributeName> member functions.
    DtDECLARE_ENTITY_ATTR_DECODER(ArticulatedParametersArray);
#if DtrPR016
    DtDECLARE_ENTITY_ATTR_DECODER(ArticulatedParametersCount);
#endif
    DtDECLARE_ENTITY_ATTR_DECODER(DamageState);
    DtDECLARE_ENTITY_ATTR_DECODER(EngineSmokeOn);
    DtDECLARE_ENTITY_ATTR_DECODER(FlamesPresent);
    DtDECLARE_ENTITY_ATTR_DECODER(HasFuelSupplyCap);
    DtDECLARE_ENTITY_ATTR_DECODER(HasRecoveryCap);
    DtDECLARE_ENTITY_ATTR_DECODER(HasRepairCap);
    DtDECLARE_ENTITY_ATTR_DECODER(HatchState);
    DtDECLARE_ENTITY_ATTR_DECODER(Immobilized);
    DtDECLARE_ENTITY_ATTR_DECODER(LifeformState);
    DtDECLARE_ENTITY_ATTR_DECODER(LightsState);
    DtDECLARE_ENTITY_ATTR_DECODER(Marking);
    DtDECLARE_ENTITY_ATTR_DECODER(PowerPlantOn);
    DtDECLARE_ENTITY_ATTR_DECODER(RampDeployed);
    DtDECLARE_ENTITY_ATTR_DECODER(SmokePlumePresent);
    DtDECLARE_ENTITY_ATTR_DECODER(TentDeployed);
    DtDECLARE_ENTITY_ATTR_DECODER(TrailState);

    // called by the decoder for the ArticulatedParametersArray
    attribute.
    static void decodeArtParts(int numParams,
        DtNetArtParamRecord* netParams,
        DtArtPartList* apList);

    // called by the decoder for the ArticulatedParametersArray
    attribute.
    static void decodeAttParts(int numParams,
        DtNetArtParamRecord* netParams,
        DtAttachedPartList* attList);
};
// These macros are used to define simple attribute decoder func-
// tions
// (members of a derived DtHlaStateDecoder that decode
// individual attributes). The simple decoding is just to
// convert the value pulled out of the ahvps to a host type using
// the cast expression castExpr, and pass it to the mutator func-
// tion.
#define DtDEFINE_SIMPLE_PHYS_ENTITY_ATTR_DECODER_WITH_CAST( \
    attrName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_ATTR_DECODER_WITH_CAST( \
        DtPhysicalEntityDecoder, DtEntityStateRepository, \
        attrName, netType, mutator, castExpr)

#define DtDEFINE_SIMPLE_PHYS_ENTITY_ATTR_DECODER( \
    attrName, netType, inspector) \
    DtDEFINE_SIMPLE_ATTR_DECODER_WITH_CAST( \

```

```
    attrName, netType, inspector, (netType))  
#endif
```


physEntEnc.h

```

#if DtHLA

#include "baseEntEnc.h"
#include "entitySR.h"

// class DtPhysicalEntityEncoder:
//
// Instances of DtPhysicalEntityEncoder are used to encode a
// physical entity's state repository into a state message in an
// HLA environment.

class DT_DLL_VL DtPhysicalEntityEncoder : public DtBaseEntityEncoder
{
public:

    // constructor
    DtPhysicalEntityEncoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtPhysicalEntityEncoder();

    // copy constructor
    DtPhysicalEntityEncoder(const DtPhysicalEntityEncoder& orig);

    // assignment operator
    DtPhysicalEntityEncoder& operator=(const DtPhysicalEntityEncoder&
orig);

public:

    // Create a DtPhysicalEntityEncoder.
    // Caller is responsible for deletion.
    static DtHlaStateEncoder* create(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

protected:

    // static need<Attribute> member functions.
    DtDECLARE_ENTITY_ATTR_CHECKER(ArticulatedParametersArray);
#if DtRPR016
    DtDECLARE_ENTITY_ATTR_CHECKER(ArticulatedParametersCount);
#endif
    DtDECLARE_ENTITY_ATTR_CHECKER(DamageState);
    DtDECLARE_ENTITY_ATTR_CHECKER(EngineSmokeOn);

```

```
DtDECLARE_ENTITY_ATTR_CHECKER(FlamesPresent);
DtDECLARE_ENTITY_ATTR_CHECKER(HasFuelSupplyCap);
DtDECLARE_ENTITY_ATTR_CHECKER(HasRecoveryCap);
DtDECLARE_ENTITY_ATTR_CHECKER(HasRepairCap);
DtDECLARE_ENTITY_ATTR_CHECKER(HatchState);
DtDECLARE_ENTITY_ATTR_CHECKER(Immobilized);
DtDECLARE_ENTITY_ATTR_CHECKER(LifeformState);
DtDECLARE_ENTITY_ATTR_CHECKER(LightsState);
DtDECLARE_ENTITY_ATTR_CHECKER(Marking);
DtDECLARE_ENTITY_ATTR_CHECKER(PowerPlantOn);
DtDECLARE_ENTITY_ATTR_CHECKER(RampDeployed);
DtDECLARE_ENTITY_ATTR_CHECKER(SmokePlumePresent);
DtDECLARE_ENTITY_ATTR_CHECKER(TentDeployed);
DtDECLARE_ENTITY_ATTR_CHECKER(TrailState);

// static decode<Attribute> member functions.
DtDECLARE_ENTITY_ATTR_ENCODER(ArticulatedParametersArray);
#if DtrPR016
DtDECLARE_ENTITY_ATTR_ENCODER(ArticulatedParametersCount);
#endif
DtDECLARE_ENTITY_ATTR_ENCODER(DamageState);
DtDECLARE_ENTITY_ATTR_ENCODER(EngineSmokeOn);
DtDECLARE_ENTITY_ATTR_ENCODER(FlamesPresent);
DtDECLARE_ENTITY_ATTR_ENCODER(HasFuelSupplyCap);
DtDECLARE_ENTITY_ATTR_ENCODER(HasRecoveryCap);
DtDECLARE_ENTITY_ATTR_ENCODER(HasRepairCap);
DtDECLARE_ENTITY_ATTR_ENCODER(HatchState);
DtDECLARE_ENTITY_ATTR_ENCODER(Immobilized);
DtDECLARE_ENTITY_ATTR_ENCODER(LifeformState);
DtDECLARE_ENTITY_ATTR_ENCODER(LightsState);
DtDECLARE_ENTITY_ATTR_ENCODER(Marking);
DtDECLARE_ENTITY_ATTR_ENCODER(PowerPlantOn);
DtDECLARE_ENTITY_ATTR_ENCODER(RampDeployed);
DtDECLARE_ENTITY_ATTR_ENCODER(SmokePlumePresent);
DtDECLARE_ENTITY_ATTR_ENCODER(TentDeployed);
DtDECLARE_ENTITY_ATTR_ENCODER(TrailState);
};
// This macro is used to define simple attribute encoder functions
// (members of a derived DtHlaStateEncoder that encode
// individual attributes). The simple encoding is just to implicitly
// convert the value returned by the inspector to a netType,
// and add it to the ahvps.
#define DtDEFINE_SIMPLE_PHYS_ENTITY_ATTR_ENCODER( \
    attrName, netType, inspector) \
    DtDEFINE_SIMPLE_ATTR_ENCODER(DtPhysicalEntityEncoder, \
        DtEntityStateRepository, \
        attrName, netType, inspector)

// This macro is used to define simple attribute checker functions
```

```
// (members of a derived DtHlaStateEncoder that checker whether
// a particular attribute is out of date. The simple implementation
// is to just compare the values returned by the two StateReps'
// inspector functions.
#define DtDEFINE_SIMPLE_PHYS_ENTITY_ATTR_CHECKER( \
    attrName, inspector) \
    DtDEFINE_SIMPLE_ATTR_CHECKER(DtPhysicalEntityEncoder, \
        DtEntityStateRepository, \
        attrName, inspector)

#endif
```

pointObjStPdu.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "hostStructs.h"
#include "pdu.h"
#include "mathTypes.h"
#include "exerciseConn.h"

#include "packets/pointObjStP.h"

#include "objectType.h"

// class DtPointObjectStatePdu:
//
// Instances of DtPointObjectStatePdu are used to state information
for
// synthetic environment objects.

// DtPointObjectStatePdu callback function signature.
typedef void (*DtPointObjectStatePduCb)(DtPointObjectStatePdu* pdu,
void* usr);

class DT_DLL_VL DtPointObjectStatePdu : public DtPdu
{
public:

    // Constructors
    DtPointObjectStatePdu(DtBufferPtr buffer =
DtUSE_INTERNAL_BUFFER);
    DtPointObjectStatePdu(const DtNetPointObjectStatePdu* initial,
DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtPointObjectStatePdu(const DtPointObjectStatePdu& orig);

    // destructor
    virtual ~DtPointObjectStatePdu();

    // Assignment operator
    DtPointObjectStatePdu& operator=(const DtPointObjectStatePdu&
orig);

    // Set/Get object ID
    virtual void setObjectID(const DtEntityIdentifier& id);
    virtual const DtEntityIdentifier& objectID() const;

    // Set/Get referenced object ID
```

```
virtual void setReferencedObjectID(const DtEntityIdentifier& id);
virtual const DtEntityIdentifier& referencedObjectID() const;

// Set/Get update number
virtual void setUpdateNumber(DtU16 field);
virtual DtU16 updateNumber() const;

// Set/Get the force ID
virtual void setForceID(DtForceType force);
virtual DtForceType forceID() const;

// Set/Get the modifications (by 8-bit integer)
virtual void setModifications(DtU8 value);
virtual DtU8 modifications() const;

// Set/Get the modifications (by field)
virtual void setModificationsField(
    DtModificationField field, DtBoolean value);
virtual DtBoolean modificationsField(
    DtModificationField field) const;

// Set/Get the object type
virtual void setObjectType(const DtObjectType& type);
virtual const DtObjectType& objectType() const;

// Set/Get location
virtual void setLocation(DtConstVector coords);
virtual DtConstVector location() const;

// Set/Get orientation
virtual void setOrientation(const DtTaitBryan& orientation);
virtual const DtTaitBryan& orientation() const;

// Set/Get appearance, the lower 16 bits (as an integer)
virtual void setAppearance(DtU16 app);
virtual DtU16 appearance() const;

// Set/Get appearance by attribute for the lower 16 bits
virtual void setAppearanceAttribute(
    DtAppearAttribute attr, DtAppearAttrVal val);
virtual DtAppearAttrVal appearanceAttribute(
    DtAppearAttribute attr) const;

// Set/Get specific appearance, the upper 32 bits (as a 32 bit
integer)
virtual void setSpecificAppearance(DtU32 app);
virtual DtU32 specificAppearance() const;

// Set/Get specific appearance by attribute, the upper 32 bits
virtual void setSpecificAppearanceAttribute(
```

```
    const DtAppearAttribute attr, const DtAppearAttrVal val);
virtual DtAppearAttrVal specificAppearanceAttribute(
    const DtAppearAttribute attr) const;

// Set/Get the requester ID
virtual void setRequesterID(const DtSimulationAddress& type);
virtual const DtSimulationAddress& requesterID() const;

// Set/Get the receiving ID
virtual void setReceivingID(const DtSimulationAddress& type);
virtual const DtSimulationAddress& receivingID() const;

// Print the data contained by this PDU
virtual void printData() const;

public:

    // Create a DtPointObjectStatePdu.
    // Caller is responsible for deletion.
    static DtPdu* create(const DtNetPacket* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // Add (register)/Remove (deregister) function to be called when
    // pdu is received.
    static void addCallback(DtExerciseConn* conn,
        DtPointObjectStatePduCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtPointObjectStatePduCb cb, void* usr);

protected:

    // Returns the internal PDU kind
    virtual DtPduKind internalGetPduKind() const;

    // Casts the netPdu element to our network struct
    virtual DtNetPointObjectStatePdu* netPduStruct() const;

    // Casts the 'this' pointer for changing elements in a const
    function
    // Non-virtual since you can't override based on return type
    DtPointObjectStatePdu* self() const;

protected:

    // Size of PDU with zero parameter records.
    static int theStaticSize;

protected:

    // Temporary objects for reference returns
```

```
DtEntityIdentifier myEntityID;  
DtObjectType myObjectType;  
DtVector myVector;  
DtTaitBryan myEulerAngles;  
DtSimulationAddress mySimulationAddress;  
  
};  
#endif  
#endif
```

printUtil.h

```
#include <stdio.h>
extern FILE* DtDumpFile;

#include "MachineTypes.h"

DT_DLL_VLUTIL void DtDumpInteger(const char *label, long i);
DT_DLL_VLUTIL void DtDumpFloat(const char *label, double f);
DT_DLL_VLUTIL void DtDumpString(const char *label, const char
*string);
DT_DLL_VLUTIL void DtDumpEnum(const char *label, const char
*enumStr, int enumNum);
DT_DLL_VLUTIL void DtDumpHexBytes(const char *label, const void
*bytes, int size);
DT_DLL_VLUTIL void DtDumpHexWord(const char *label, DtU32 word);
DT_DLL_VLUTIL void DtDumpBoolean(const char *label, DtBoolean val,
int onlyIfTrue = 0);
```


ProcControl.h

```
#include "vLTime.h"

DT_DLL_VLUTIL void DtSleep(DtTime sleep_period);

/* polls line-buffered standard input
 * returns NULL if no input
 * returns internal buffer of line if have input
 * exits on ^D
 */
DT_DLL_VLUTIL char *DtPollInputLine(void);

#ifdef PCFTP
    /* the following is used as the first argument to the PC version
       of select. It cannot take a 0 like unix versions. I should be
       able
       to get this number programatically. */
    extern DtMaximumSocketNumber;
#endif
```

queryPdu.h

```
#if !DtHLA

#include "hostStructs.h"
#include "simanPdu.h"

#if DtDISVERS != 3
#define DtQueryStaticSize 40
#else
#define DtQueryStaticSize 44
#endif

class DT_DLL_VL DtDataQueryPdu;
typedef DtDataQueryPdu DtDataQueryInteraction;
typedef void (*DtDataQueryPduCb)(DtDataQueryPdu *pdu, void *usr);

class DT_DLL_VL DtDataQueryPdu : public DtSimanPdu
{
public:

/***** STANDARD CONSTRUCTORS *****/

    DtDataQueryPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtDataQueryPdu(const DtNetDataQueryPdu *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    static DtPdu *create(const DtNetPacket *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    static void addCallback(DtExerciseConn *conn,
        DtDataQueryPduCb cb, void *usr);
    static void removeCallback(DtExerciseConn *conn,
        DtDataQueryPduCb cb, void *usr);

/***** INSPECTOR FUNCTIONS *****/

    unsigned long requestId() const;
    DtTime timeInterval() const;
    unsigned long numFixedFields() const;
    unsigned long numVarFields() const;
    DtDatumParam fixedDatumId(int index) const;
    DtDatumParam varDatumId(int index) const;

/***** MUTATOR FUNCTIONS *****/

    void setRequestId(unsigned long id);
    void setTimeInterval(DtTime seconds);
    int setFixedDatumId(int index, DtDatumParam id);
    int setVarDatumId(int index, DtDatumParam id);
```

```
void initDatumIds(unsigned long numFixedFields,
                  unsigned long numVarFields);

void printData() const;

protected:
    DtPduKind internalGetPduKind() const {return DtDataQueryPdu-
Kind;}
};
#endif
```

radarBeamDec.h

```
#if DtHLA

#ifndef DtRadarBeamDecoder_H_
#define DtRadarBeamDecoder_H_

#include "chooseDefines.h"
#include "baseEmitttrDec.h"

// This is used to declare attribute decoding functions
// which are members of the class.
#define DtDECLARE_RADARBEAM_ATTR_DECODER(attrName) \
    DtDECLARE_ATTR_DECODER(DtEmitterBeamRepository, attrName)

// class DtRadarBeamDecoder:
//
// Instances of DtRadarBeamDecoder are

class DT_DLL_VL DtRadarBeamDecoder : public DtBaseEmitterBeamDecoder
{
public:

    // default constructor
    DtRadarBeamDecoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtRadarBeamDecoder();

    // copy constructor
    DtRadarBeamDecoder(const DtRadarBeamDecoder& orig);

    // assignment operator
    DtRadarBeamDecoder& operator=(const DtRadarBeamDecoder& orig);

public:

    // Create a DtBaseEntityDecoder. Caller is responsible for deletion.
    static DtHlaStateDecoder* create(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

protected:

    // Add decoding functions for the attributes.
```

```

    // Note: Non-virtual since meant to be called from constructors
to
    //          add decoders.
    void addMyDecoders();

    // static decode<AttributeName> member functions.
    DtDECLARE_RADARBEAM_ATTR_DECODER(TrackObjectIDs);
    DtDECLARE_RADARBEAM_ATTR_DECODER(HighDensityTrack);
};
// These macros are used to define simple attribute decoder func-
tions
// (members of a derived DtHlaStateDecoder that decode
// individual attributes). The simple decoding is just to
// convert the value pulled out of the ahvps to a host type using
// the cast expression castExpr, and pass it to the mutator func-
tion.
#define DtDEFINE_SIMPLE_RADARBEAM_ATTR_DECODER_WITH_CAST( \
    attrName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_ATTR_DECODER_WITH_CAST( \
        DtRadarBeamDecoder, DtEmitterBeamRepository, \
        attrName, netType, mutator, castExpr)

#define DtDEFINE_SIMPLE_RADARBEAM_ATTR_DECODER( \
    attrName, netType, inspector) \
    DtDEFINE_SIMPLE_RADARBEAM_ATTR_DECODER_WITH_CAST( \
        attrName, netType, inspector, (netType))

#endif    //DtRadarBeamDecoder_H_

```

radarBeamEnc.h

```
#if DtHLA

#ifndef DtRadarBeamEncoder_H_
#define DtRadarBeamEncoder_H_

#include "chooseDefines.h"
#include "baseEmitttrEnc.h"

// This is used to declare static member functions for determining
// whether or not an attribute needs to be encoded.
#define DtDECLARE_RADARBEAM_ATTR_CHECKER(attrName) \
    DtDECLARE_ATTR_CHECKER(DtEmitterBeamRepository, attrName)

// This is used to declare attribute encoding functions
// which are members of the class.
#define DtDECLARE_RADARBEAM_ATTR_ENCODER(attrName) \
    DtDECLARE_ATTR_ENCODER(DtEmitterBeamRepository, attrName)

// class DtRadarBeamEncoder:
//
// Instances of DtRadarBeamEncoder are

class DT_DLL_VL DtRadarBeamEncoder : public DtBaseEmitterBeamEncoder
{
public:

    // default constructor
    DtRadarBeamEncoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtRadarBeamEncoder();

    // copy constructor
    DtRadarBeamEncoder(const DtRadarBeamEncoder& orig);

    // assignment operator
    DtRadarBeamEncoder& operator=(const DtRadarBeamEncoder& orig);

public:

    // Create a DtBaseEntityEncoder. Caller is responsible for dele-
    // tion.
    static DtHlaStateEncoder* create(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);
```

```

protected:
    DtDECLARE_RADARBEAM_ATTR_ENCODER(TrackObjectIDs);
    DtDECLARE_RADARBEAM_ATTR_ENCODER(HighDensityTrack);

    DtDECLARE_RADARBEAM_ATTR_CHECKER(TrackObjectIDs);
    DtDECLARE_RADARBEAM_ATTR_CHECKER(HighDensityTrack);
};
// This macro is used to define simple attribute encoder functions
// (members of a derived DtHlaStateEncoder that encode
// individual attributes). The simple encoding is just to implicitly
// convert the value returned by the inspector to a netType,
// and add it to the ahvps.
#define DtDEFINE_SIMPLE_RADARBEAM_ATTR_ENCODER( \
    attrName, netType, inspector) \
    DtDEFINE_SIMPLE_ATTR_ENCODER(DtRadarBeamEncoder, \
        DtEmitterBeamRepository, \
        attrName, netType, inspector)

// This macro is used to define simple attribute checker functions
// (members of a derived DtHlaStateEncoder that check whether
// a particular attribute is out of date. The simple implementation
// is to just compare the values returned by the two StateReps'
// inspector functions.
#define DtDEFINE_SIMPLE_RADARBEAM_ATTR_CHECKER( \
    attrName, inspector) \
    DtDEFINE_SIMPLE_ATTR_CHECKER(DtRadarBeamEncoder, \
        DtEmitterBeamRepository, \
        attrName, inspector)

#endif //DtRadarBeamEncoder_H_

```

radioRecvrDec.h

```
#if DtHLA

#ifndef DtRadioReceiverDecoder_H_
#define DtRadioReceiverDecoder_H_

#include "chooseDefines.h"
#include "baseEmbedDec.h"

// This is used to declare attribute decoding functions
// which are members of the class.
#define DtDECLARE_RADIORECVR_ATTR_DECODER(attrName) \
    DtDECLARE_ATTR_DECODER(DtRadioReceiverRepository, attrName)

// class DtRadioReceiverDecoder:
//
// Instances of DtRadioReceiverDecoder are

class DT_DLL_VL DtRadioReceiverDecoder : public DtBaseEmbeddedSys-
temDecoder
{
public:

    // default constructor
    DtRadioReceiverDecoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtRadioReceiverDecoder();

    // copy constructor
    DtRadioReceiverDecoder(const DtRadioReceiverDecoder& orig);

    // assignment operator
    DtRadioReceiverDecoder&operator=(const DtRadioReceiverDecoder&
orig);

public:

    // Create a DtBaseEntityDecoder. Caller is responsible for dele-
    tion.
    static DtHlaStateDecoder* create(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

protected:
```

```

    // Add decoding functions for the attributes.
    // Note: Non-virtual since meant to be called from constructors
to
    //          add decoders.
    void addMyDecoders();

    // static decode<AttributeName> member functions.

    DtDECLARE_RADIORECVR_ATTR_DECODER(RadioNumber);
    DtDECLARE_RADIORECVR_ATTR_DECODER(ReceivedPower);
    DtDECLARE_RADIORECVR_ATTR_DECODER(ReceivedTransmitter);
    DtDECLARE_RADIORECVR_ATTR_DECODER(ReceiverState);
};
// These macros are used to define simple attribute decoder func-
tions
// (members of a derived DtHlaStateDecoder that decode
// individual attributes). The simple decoding is just to
// convert the value pulled out of the ahvps to a host type using
// the cast expression castExpr, and pass it to the mutator func-
tion.
#define DtDEFINE_SIMPLE_RADIORECVR_ATTR_DECODER_WITH_CAST( \
    attrName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_ATTR_DECODER_WITH_CAST( \
        DtRadioReceiverDecoder, DtRadioReceiverRepository, \
        attrName, netType, mutator, castExpr)

#define DtDEFINE_SIMPLE_RADIORECVR_ATTR_DECODER( \
    attrName, netType, inspector) \
    DtDEFINE_SIMPLE_RADIORECVR_ATTR_DECODER_WITH_CAST( \
        attrName, netType, inspector, (netType))

#endif    //DtRadioReceiverDecoder_H_

```

radioRecvrEnc.h

```
#if DtHLA

#ifndef DtRadioReceiverEncoder_H_
#define DtRadioReceiverEncoder_H_

#include "chooseDefines.h"
#include "baseEmbedEnc.h"

// This is used to declare static member functions for determining
// whether or not an attribute needs to be encoded.
#define DtDECLARE_RADIORECVR_ATTR_CHECKER(attrName) \
    DtDECLARE_ATTR_CHECKER(DtRadioReceiverRepository, attrName)

// This is used to declare attribute encoding functions
// which are members of the class.
#define DtDECLARE_RADIORECVR_ATTR_ENCODER(attrName) \
    DtDECLARE_ATTR_ENCODER(DtRadioReceiverRepository, attrName)

// class DtRadioReceiverEncoder:
//
// Instances of DtRadioReceiverEncoder are

class DT_DLL_VL DtRadioReceiverEncoder : public DtBaseEmbeddedSystemEncoder
{
public:

    // default constructor
    DtRadioReceiverEncoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtRadioReceiverEncoder();

    // copy constructor
    DtRadioReceiverEncoder(const DtRadioReceiverEncoder& orig);

    // assignment operator
    DtRadioReceiverEncoder& operator=(const DtRadioReceiverEncoder&
orig);

public:

    // Create a DtBaseEntityEncoder. Caller is responsible for deletion.
    static DtHlaStateEncoder* create(
```

```

        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

protected:

    DtDECLARE_RADIORECVR_ATTR_CHECKER(RadioNumber);
    DtDECLARE_RADIORECVR_ATTR_CHECKER(ReceivedPower);
    DtDECLARE_RADIORECVR_ATTR_CHECKER(ReceivedTransmitter);
    DtDECLARE_RADIORECVR_ATTR_CHECKER(ReceiverState);

    DtDECLARE_RADIORECVR_ATTR_ENCODER(RadioNumber);
    DtDECLARE_RADIORECVR_ATTR_ENCODER(ReceivedPower);
    DtDECLARE_RADIORECVR_ATTR_ENCODER(ReceivedTransmitter);
    DtDECLARE_RADIORECVR_ATTR_ENCODER(ReceiverState);
};
// This macro is used to define simple attribute encoder functions
// (members of a derived DtHlaStateEncoder that encode
// individual attributes). The simple encoding is just to implicitly
// convert the value returned by the inspector to a netType,
// and add it to the ahvps.
#define DtDEFINE_SIMPLE_RADIORECVR_ATTR_ENCODER( \
    attrName, netType, inspector) \
    DtDEFINE_SIMPLE_ATTR_ENCODER(DtRadioReceiverEncoder, \
        DtRadioReceiverRepository, \
        attrName, netType, inspector)

// This macro is used to define simple attribute checker functions
// (members of a derived DtHlaStateEncoder that check whether
// a particular attribute is out of date. The simple implementation
// is to just compare the values returned by the two StateReps'
// inspector functions.
#define DtDEFINE_SIMPLE_RADIORECVR_ATTR_CHECKER( \
    attrName, inspector) \
    DtDEFINE_SIMPLE_ATTR_CHECKER(DtRadioReceiverEncoder, \
        DtRadioReceiverRepository, \
        attrName, inspector)

#endif //DtRadioReceiverEncoder_H_

```

radioRecvrSR.h

```
#include "embedSystemSR.h"
#include "chooseDefines.h"

// class DtRadioReceiverRepository:
//
// Instances of DtRadioReceiverRepository are

class DT_DLL_VL DtRadioReceiverRepository : public DtEmbeddedSystem-
Repository
{
public:

    // default constructor
    DtRadioReceiverRepository();

    // destructor
    virtual ~DtRadioReceiverRepository();

    // copy constructor
    DtRadioReceiverRepository(const DtRadioReceiverRepository& orig);

    // assignment operator
    DtRadioReceiverRepository& operator=(const DtRadioReceiverReposi-
tory& orig);

    // return an empty object of the same type as this
    virtual DtStateRepository* clone() const;

    static DtRadioReceiverRepository* create();

    // Set/Get the Receiver radio number.
    virtual void setRadioId(unsigned int number);
    virtual unsigned int radioId() const;

    // Set/Get the received power.
    virtual void setReceivedPower(DtFloat32 power);
    virtual DtFloat32 receivedPower() const;

    // Set/Get the received transmitter.
    virtual void setTransmitRadioId(const DtGlobalObjectDesignator&
id);
    virtual const DtGlobalObjectDesignator& transmitRadioId() const;

    // Set/Get the receiver state.
    virtual void setReceiverState(DtRadioReceiverState state);
    virtual DtRadioReceiverState receiverState() const;
```

```
virtual void printData() const;

protected:
    int myReceiverRadioNumber;
    DtFloat32 myReceivedPower;
    DtGlobalObjectDesignator myTransmitId;
    DtRadioReceiverState myReceiverState;
};
```

radioXmitDec.h

```
#if DtHLA

#ifndef DtRadioTransmitterDecoder_H_
#define DtRadioTransmitterDecoder_H_

#include "chooseDefines.h"
#include "baseEmbedDec.h"

// This is used to declare attribute decoding functions
// which are members of the class.
#define DtDECLARE_RADIOXMIT_ATTR_DECODER(attrName) \
    DtDECLARE_ATTR_DECODER(DtRadioTransmitterRepository, attrName)

// class DtRadioTransmitterDecoder:
//
// Instances of DtRadioTransmitterDecoder are

class DT_DLL_VL DtRadioTransmitterDecoder : public DtBaseEmbedded-
SystemDecoder
{
public:

    // default constructor
    DtRadioTransmitterDecoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtRadioTransmitterDecoder();

    // copy constructor
    DtRadioTransmitterDecoder(const DtRadioTransmitterDecoder& orig);

    // assignment operator
    DtRadioTransmitterDecoder& operator=
        (const DtRadioTransmitterDecoder& orig);

public:

    // Create a DtBaseEntityDecoder. Caller is responsible for dele-
    // tion.
    static DtHlaStateDecoder* create(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

protected:
```

```

    // Add decoding functions for the attributes.
    // Note: Non-virtual since meant to be called from constructors
to
    //          add decoders.
    void addMyDecoders();

    // static decode<AttributeName> member functions.
    DtDECLARE_RADIOXMIT_ATTR_DECODER(AntennaPattern);
    DtDECLARE_RADIOXMIT_ATTR_DECODER(AntennaPatternType);
    DtDECLARE_RADIOXMIT_ATTR_DECODER(CryptoKeyID);

    // Not used before RPR 0.8
    DtDECLARE_RADIOXMIT_ATTR_DECODER(CryptoMode);

    DtDECLARE_RADIOXMIT_ATTR_DECODER(CryptoSystem);
    DtDECLARE_RADIOXMIT_ATTR_DECODER(DetailedModulationType);
    DtDECLARE_RADIOXMIT_ATTR_DECODER(Frequency);
    DtDECLARE_RADIOXMIT_ATTR_DECODER(FrequencyBandwidth);
    DtDECLARE_RADIOXMIT_ATTR_DECODER(FrequencyHopInUse);
    DtDECLARE_RADIOXMIT_ATTR_DECODER(InputSource);
    DtDECLARE_RADIOXMIT_ATTR_DECODER(ModulationParameters);
    DtDECLARE_RADIOXMIT_ATTR_DECODER(MajorModulationType);
    DtDECLARE_RADIOXMIT_ATTR_DECODER(ModulationSystemType);
    DtDECLARE_RADIOXMIT_ATTR_DECODER(Power);
    DtDECLARE_RADIOXMIT_ATTR_DECODER(PsuedoNoiseSpectrumInUse);
    DtDECLARE_RADIOXMIT_ATTR_DECODER(RadioNumber);
    DtDECLARE_RADIOXMIT_ATTR_DECODER(RadioType);
    DtDECLARE_RADIOXMIT_ATTR_DECODER(TimeHopInUse);
    DtDECLARE_RADIOXMIT_ATTR_DECODER(TransmitState);
    DtDECLARE_RADIOXMIT_ATTR_DECODER(WorldLocation);
};
// These macros are used to define simple attribute decoder func-
tions
// (members of a derived DtHlaStateDecoder that decode
// individual attributes). The simple decoding is just to
// convert the value pulled out of the ahvps to a host type using
// the cast expression castExpr, and pass it to the mutator func-
tion.
#define DtDEFINE_SIMPLE_RADIOXMIT_ATTR_DECODER_WITH_CAST( \
    attrName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_ATTR_DECODER_WITH_CAST( \
        DtRadioTransmitterDecoder, DtRadioTransmitterRepository, \
        attrName, netType, mutator, castExpr)

#define DtDEFINE_SIMPLE_RADIOXMIT_ATTR_DECODER( \
    attrName, netType, inspector) \
    DtDEFINE_SIMPLE_RADIOXMIT_ATTR_DECODER_WITH_CAST( \
        attrName, netType, inspector, (netType))

#endif    //DtRadioTransmitterDecoder_H_

```

radioXmitEnc.h

```
#if DtHLA

#ifndef DtRadioTransmitterEncoder_H_
#define DtRadioTransmitterEncoder_H_

#include "chooseDefines.h"
#include "baseEmbedEnc.h"

// This is used to declare static member functions for determining
// whether or not an attribute needs to be encoded.
#define DtDECLARE_RADIOXMIT_ATTR_CHECKER(attrName) \
    DtDECLARE_ATTR_CHECKER(DtRadioTransmitterRepository, attrName)

// This is used to declare attribute encoding functions
// which are members of the class.
#define DtDECLARE_RADIOXMIT_ATTR_ENCODER(attrName) \
    DtDECLARE_ATTR_ENCODER(DtRadioTransmitterRepository, attrName)

// class DtRadioTransmitterEncoder:
//
// Instances of DtRadioTransmitterEncoder are

class DT_DLL_VL DtRadioTransmitterEncoder : public DtBaseEmbedded-
SystemEncoder
{
public:

    // default constructor
    DtRadioTransmitterEncoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtRadioTransmitterEncoder();

    // copy constructor
    DtRadioTransmitterEncoder(const DtRadioTransmitterEncoder& orig);

    // assignment operator
    DtRadioTransmitterEncoder& operator=(const DtRadioTransmitterEn-
coder& orig);

public:

    // Create a DtBaseEntityEncoder. Caller is responsible for dele-
tion.
    static DtHlaStateEncoder* create(
```



```
DtExerciseConn* exConn,
DtObjClassDesc* classDesc);
```

protected:

```
DtDECLARE_RADIOXMIT_ATTR_CHECKER(AntennaPattern);
DtDECLARE_RADIOXMIT_ATTR_CHECKER(AntennaPatternType);
DtDECLARE_RADIOXMIT_ATTR_CHECKER(CryptoKeyID);

// Not used before RPR 0.8
DtDECLARE_RADIOXMIT_ATTR_CHECKER(CryptoMode);

DtDECLARE_RADIOXMIT_ATTR_CHECKER(CryptoSystem);
DtDECLARE_RADIOXMIT_ATTR_CHECKER(DetailedModulationType);
DtDECLARE_RADIOXMIT_ATTR_CHECKER(Frequency);
DtDECLARE_RADIOXMIT_ATTR_CHECKER(FrequencyBandwidth);
DtDECLARE_RADIOXMIT_ATTR_CHECKER(FrequencyHopInUse);
DtDECLARE_RADIOXMIT_ATTR_CHECKER(InputSource);
DtDECLARE_RADIOXMIT_ATTR_CHECKER(ModulationParameters);
DtDECLARE_RADIOXMIT_ATTR_CHECKER(MajorModulationType);
DtDECLARE_RADIOXMIT_ATTR_CHECKER(ModulationSystemType);
DtDECLARE_RADIOXMIT_ATTR_CHECKER(Power);
DtDECLARE_RADIOXMIT_ATTR_CHECKER(PsuedoNoiseSpectrumInUse);
DtDECLARE_RADIOXMIT_ATTR_CHECKER(RadioNumber);
DtDECLARE_RADIOXMIT_ATTR_CHECKER(RadioType);
DtDECLARE_RADIOXMIT_ATTR_CHECKER(TimeHopInUse);
DtDECLARE_RADIOXMIT_ATTR_CHECKER(TransmitState);
DtDECLARE_RADIOXMIT_ATTR_CHECKER(WorldLocation);

DtDECLARE_RADIOXMIT_ATTR_ENCODER(AntennaPattern);
DtDECLARE_RADIOXMIT_ATTR_ENCODER(AntennaPatternType);
DtDECLARE_RADIOXMIT_ATTR_ENCODER(CryptoKeyID);

// Not used before RPR 0.8
DtDECLARE_RADIOXMIT_ATTR_ENCODER(CryptoMode);

DtDECLARE_RADIOXMIT_ATTR_ENCODER(CryptoSystem);
DtDECLARE_RADIOXMIT_ATTR_ENCODER(DetailedModulationType);
DtDECLARE_RADIOXMIT_ATTR_ENCODER(Frequency);
DtDECLARE_RADIOXMIT_ATTR_ENCODER(FrequencyBandwidth);
DtDECLARE_RADIOXMIT_ATTR_ENCODER(FrequencyHopInUse);
DtDECLARE_RADIOXMIT_ATTR_ENCODER(InputSource);
DtDECLARE_RADIOXMIT_ATTR_ENCODER(ModulationParameters);
DtDECLARE_RADIOXMIT_ATTR_ENCODER(MajorModulationType);
DtDECLARE_RADIOXMIT_ATTR_ENCODER(ModulationSystemType);
DtDECLARE_RADIOXMIT_ATTR_ENCODER(Power);
DtDECLARE_RADIOXMIT_ATTR_ENCODER(PsuedoNoiseSpectrumInUse);
DtDECLARE_RADIOXMIT_ATTR_ENCODER(RadioNumber);
DtDECLARE_RADIOXMIT_ATTR_ENCODER(RadioType);
DtDECLARE_RADIOXMIT_ATTR_ENCODER(TimeHopInUse);
```

```
DtDECLARE_RADIOXMIT_ATTR_ENCODER(TransmitState);
DtDECLARE_RADIOXMIT_ATTR_ENCODER(WorldLocation);
};
// This macro is used to define simple attribute encoder functions
// (members of a derived DtHlaStateEncoder that encode
// individual attributes). The simple encoding is just to implicitly
// convert the value returned by the inspector to a netType,
// and add it to the ahvps.
#define DtDEFINE_SIMPLE_RADIOXMIT_ATTR_ENCODER( \
    attrName, netType, inspector) \
    DtDEFINE_SIMPLE_ATTR_ENCODER(DtRadioTransmitterEncoder, \
        DtRadioTransmitterRepository, \
        attrName, netType, inspector)

// This macro is used to define simple attribute checker functions
// (members of a derived DtHlaStateEncoder that check whether
// a particular attribute is out of date. The simple implementation
// is to just compare the values returned by the two StateReps'
// inspector functions.
#define DtDEFINE_SIMPLE_RADIOXMIT_ATTR_CHECKER( \
    attrName, inspector) \
    DtDEFINE_SIMPLE_ATTR_CHECKER(DtRadioTransmitterEncoder, \
        DtRadioTransmitterRepository, \
        attrName, inspector)

#endif //DtRadioTransmitterEncoder_H_
```

radioXmitSR.h

```
#include "chooseDefines.h"
#include "embedSystemSR.h"
#include "antPatRec.h"

enum DtLongWordPortion
{
    DtUpperWord,
    DtLowerWord
};

// class DtRadioTransmitterRepository:
//
// Instances of DtRadioTransmitterRepository are
Error - trouble finding class open brace.
```

receivePdu.h

```
#if !DtHLA

#include "hostStructs.h"
#include "pdu.h"
#include "packets/transmitterP.h"
#define DtReceiverStaticSize 36

class DT_DLL_VL DtReceiverPdu;
typedef void (*DtReceiverPduCb)(DtReceiverPdu *pdu, void *usr);

class DT_DLL_VL DtReceiverPdu : public DtPdu
{
public:

/***** STANDARD CONSTRUCTORS *****/

    DtReceiverPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtReceiverPdu(const DtNetReceiverPdu *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    static DtPdu *create(const DtNetPacket *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    static void addCallback(DtExerciseConn *conn,
        DtReceiverPduCb cb, void *usr);
    static void removeCallback(DtExerciseConn *conn,
        DtReceiverPduCb cb, void *usr);

/***** INSPECTOR FUNCTIONS *****/

    const DtEntityIdentifier &entityId() const;
    unsigned int radioId() const;
    DtRadioReceiverState receiverState() const;
    float receivedPower() const;
    const DtEntityIdentifier &transmitEntityId() const;
    unsigned int transmitRadioId() const;

/***** MUTATOR FUNCTIONS *****/

    void setEntityId(const DtEntityIdentifier &id);
    void setRadioId(unsigned int id);
    void setReceiverState(DtRadioReceiverState state);
    void setReceivedPower(float power);
    void setTransmitEntityId(const DtEntityIdentifier &id);
    void setTransmitRadioId(unsigned int id);

    void printData() const;
protected:
```

```
DtPduKind internalGetPduKind() const {return DtReceiverPduKind;}
DtEntityIdentifier theId;
DtEntityIdentifier transId;
};
#endif // !DtHLA
```

recIntrCbInfo.h

```
#if DtHLA

#include "RTI.hh"
#include "MachineTypes.h"
#include "chooseDefines.h"

// class DtReceiveInteractionCbInfo:
//
// Instances of DtReceiveInteractionCbInfo are used for providing
// a callback when an interaction is received from another federate.

typedef void (*DtReceiveInteractionCb)(DtInteraction* inter, void*
usr);

class DT_DLL_VL DtReceiveInteractionCbInfo
{
public:

    // constructor
    DtReceiveInteractionCbInfo(RTI::InteractionClassHandle h,
        DtReceiveInteractionCb cb,
        void* usr);

    // destructor
    virtual ~DtReceiveInteractionCbInfo();

    // copy constructor
    DtReceiveInteractionCbInfo(
        const DtReceiveInteractionCbInfo& orig);

    // assignment operator
    DtReceiveInteractionCbInfo& operator=(
        const DtReceiveInteractionCbInfo& orig);

    // Check whether one callback info is equal to another.
    virtual int match(const DtReceiveInteractionCbInfo* const other)
const;

    // Check whether one callback info is equal to another.
    virtual int operator==(const DtReceiveInteractionCbInfo& other)
const;

    // Get the interaction class handle.
    virtual const RTI::InteractionClassHandle& interactionClass()
const;

    // Get the callback function.
```

```
virtual const DtReceiveInteractionCb& cbFunction() const;

// Get the user data.
virtual void* userData() const;

protected:

    RTI::InteractionClassHandle myClass;
    DtReceiveInteractionCb myFunc;
    void* myUsr;
};
#endif
```

recordPdu.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "hostStructs.h"
#include "recordSetPdu.h"
#include "exerciseConn.h"

#include "packets/recordP.h"

// class DtRecordPdu:
//
// Instances of DtRecordPdu are used to send responses to the Record
// Query
// PDU.

// DtRecordPdu callback function signature.
typedef void (*DtRecordPduCb)(DtRecordPdu* pdu, void* usr);

class DT_DLL_VL DtRecordPdu : public DtRecordSetPdu
{
public:

    // Constructors
    DtRecordPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtRecordPdu(const DtNetRecordPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtRecordPdu(const DtRecordPdu& orig);

    // destructor
    virtual ~DtRecordPdu();

    // Assignment operator
    DtRecordPdu& operator=(const DtRecordPdu& orig);

    // Set/Get originating entity identifier
    virtual void setOriginatingEntityID(const DtEntityIdentifier&
entity);
    virtual const DtEntityIdentifier& originatingEntityID() const;

    // Set/Get receiving entity identifier
    virtual void setReceivingEntityID(const DtEntityIdentifier&
entity);
    virtual const DtEntityIdentifier& receivingEntityID() const;

    // Set/Get request ID
```



```

virtual void setRequestID(DtU32 id);
virtual DtU32 requestID() const;

// Set/Get required reliability service
virtual void setReqRelService(enum DtRequiredReliabilityService
service);
virtual DtRequiredReliabilityService reqRelService() const;

// Set/Get the event type
virtual void setEventType(DtEventType type);
virtual DtEventType eventType() const;

// Set/Get the response serial number
virtual void setResponseSerialNumber(DtU32 num);
virtual DtU32 responseSerialNumber() const;

// Print the data contained by this PDU
virtual void printData() const;

public:

    // Create a DtRecordPdu.
    // Caller is responsible for deletion.
    static DtPdu* create(const DtNetPacket* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // Add (register)/Remove (deregister) function to be called when
    // pdu is received.
    static void addCallback(DtExerciseConn* conn,
        DtRecordPduCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtRecordPduCb cb, void* usr);

protected:

    // Returns a pointer to the beginning of the record set data
    struct.
    virtual DtNetRecordSetPdu* netRecordSetPduStruct() const;

    // Returns the internal PDU kind
    virtual DtPduKind internalGetPduKind() const;

    // Casts the netPdu element to our network struct
    // Non-virtual since you can't override based on return type
    virtual DtNetRecordPdu* netPduStruct() const;

    // Casts the 'this' pointer for changing elements in a const
    function
    // Non virtual since you can't override based on return type
    DtRecordPdu* self() const;

```

protected:

```
// Size of PDU with zero data bytes.  
static int theStaticSize;
```

protected:

```
// Temporary objects for reference returns  
DtEntityIdentifier myEntityID;
```

```
};  
#endif  
#endif
```

recordSet.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"
#include "hostStructs.h"
#include "datumIdType.h"

#include "packets/recordSetP.h"

// class DtRecordSet:
//
// Instances of DtRecordSet are used to contain the record sets used
// by the PDU classes which inherit from the record set PDU base
// class.

class DT_DLL_VL DtRecordSet
{
public:

    // Constructors
    DtRecordSet();
    DtRecordSet(const DtRecordSet& orig);

    // Destructor
    virtual ~DtRecordSet();

    // Assignment operators
    DtRecordSet& operator=(const DtRecordSet& orig);
    DtRecordSet& operator=(const DtNetRecordSet& orig);

    // Set/Get the datum ID.
    virtual DtBoolean setDatumID(DtDatumParam value);
    virtual DtDatumParam datumID() const;

    // Set/Get the serial number.
    virtual void setSerialNumber(DtU32 value);
    virtual DtU32 serialNumber() const;

    // Set/Get the record length in bits.
    // setRecordBitLength() also resets the number of records to
    zero, and
    // may return DtFalse (with no action) if the length is invalid.
    virtual DtBoolean setRecordBitLength(DtU16 bitLength);
    virtual DtU16 recordBitLength() const;

```

```
// Set/Get the record length in bytes.
// setRecordLength() also resets the number of records to zero,
and
// may return DtFalse (with no action) if the length is invalid.
virtual DtBoolean setRecordByteLength(int byteLength);
virtual int recordByteLength() const;

// Set/Get the number of of records in this set
// When the number of records is changed, old records are not
preserved
virtual DtBoolean setRecordCount(DtU16 bytes);
virtual DtU16 recordCount() const;

// Set/Get multiple data bytes. Provide a pointer and data is
copied to
// and from that memory. Will copy recordByteLength() bytes on
set/get.
// These functions return DtFalse if the record index is out of
range.
virtual DtBoolean setDataBytes(int recordIndex, const void*
data);
virtual DtBoolean getDataBytes(int recordIndex, void* buff)
const;

// Set/Get a data byte. These functions return DtFalse if the
record
// index or byte index is out of range.
virtual DtBoolean setDataByte(
    int recordIndex, DtU8 value, int byteIndex = 0);
virtual DtBoolean getDataByte(
    int recordIndex, DtU8& value, int byteIndex = 0) const;

// Set/Get data in signed 16-bit integer format.
// These functions return DtFalse if the record length is not 16-
bits,
// or if the record index is out of range.
virtual DtBoolean setDataInt16(int recordIndex, DtInt16& value);
virtual DtBoolean getDataInt16(int recordIndex, DtInt16& value)
const;

// Set/Get data in unsigned 16-bit integer format.
// These functions return DtFalse if the record length is not 16-
bits,
// or if the record index is out of range.
virtual DtBoolean setDataU16(int recordIndex, DtU16& value);
virtual DtBoolean getDataU16(int recordIndex, DtU16& value)
const;

// Set/Get data in signed 32-bit integer format.
```

```

    // These functions return DtFalse if the record length is not 32-
    bits,
    // or if the record index is out of range.
    virtual DtBoolean setDataInt32(int recordIndex, DtInt32& value);
    virtual DtBoolean getDataInt32(int recordIndex, DtInt32& value)
    const;

    // Set/Get data in unsigned 32-bit integer format.
    // These functions return DtFalse if the record length is not 32-
    bits,
    // or if the record index is out of range.
    virtual DtBoolean setDataU32(int recordIndex, DtU32& value);
    virtual DtBoolean getDataU32(int recordIndex, DtU32& value)
    const;

    // Set/Get data in 32-bit floating point format.
    // These functions return DtFalse if the record length is not 32-
    bits,
    // or if the record index is out of range.
    virtual DtBoolean setDataFloat32(int recordIndex, DtFloat32&
    value);
    virtual DtBoolean getDataFloat32(int recordIndex, DtFloat32&
    value) const;

    // Set/Get data in 64-bit floating point format.
    // These functions return DtFalse if the record length is not 64-
    bits,
    // or if the record index is out of range.
    virtual DtBoolean setDataFloat64(int recordIndex, DtFloat64&
    value);
    virtual DtBoolean getDataFloat64(int recordIndex, DtFloat64&
    value) const;

    // Get the total number of bytes required for the network repre-
    sentation
    // of the record set.
    virtual int recordSetLength() const;

    // Print this record
    virtual void print() const;

    // Load/unload data from a packet record
    virtual void unloadNetRecordSet(const DtNetRecordSet& record);
    virtual void loadNetRecordSet(DtNetRecordSet& record) const;

```

protected:

DtDatumParam	myDatumID;
DtU32	mySerialNumber;
DtU16	myRecordLength;

DtU16	myRecordCount;
char*	myData;

```
};  
#endif  
#endif
```

recordSetPdu.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "hostStructs.h"
#include "pdu.h"
#include "packets/recordSetP.h"

#include "recordSet.h"

// class DtRecordSetPdu:
//
// DtRecordSetPdu is an abstract base class used by PDUs which
// require the
// record set functionality.

class DT_DLL_VL DtRecordSetPdu : public DtPdu
{
public:

    // destructor
    virtual ~DtRecordSetPdu();

    // Assignment operator
    DtRecordSetPdu& operator=(const DtRecordSetPdu& orig);

    // Set/Get the number of record sets
    virtual DtBoolean setNumRecordSets(DtU32 numSets);
    virtual DtU32 numRecordSets() const;

    // Set/Get a record set. If the index is out of bounds, then
    DtFalse
    // is returned. If the space would exceed the maximum space
    permitted
    // for a PDU (for a set) then DtFalse may be returned.
    virtual DtBoolean setRecordSet(int index, const DtRecordSet&
    recSet);
    virtual DtBoolean getRecordSet(int index, DtRecordSet& recSet)
    const;

    // Print the data contained by this PDU
    virtual void printData() const;

protected:

```

```
// Constructors (protected because this is an abstract base
class)
DtRecordSetPdu();
DtRecordSetPdu(const DtRecordSetPdu& orig);

// This function checks the data records for consistency between
the
// actual size of the PDU and the size suggested by the PDU. It
also
// tries to fix any problems that are detected. This function is
// intended to be called in the PDU constructor that accepts
network
// data immediately following initpdu.
// Non-virtual since meant to be called from ctor.
DtBoolean validateRecordSets(DtNetRecordSetPdu& netStruct);

// Returns a pointer to the beginning of the record set data
struct.
// This must be implemented by the derived class because the
location
// of the record sets within the PDU can vary.
virtual DtNetRecordSetPdu* netRecordSetPduStruct() const = 0;

// Returns a pointer to the first byte of variable length data
virtual char* var() const;

// Returns the length of data used by the record sets beginning at
// the location specified by netRecordSetPduStruct(). This can
be used
// by the derived class if any data follows the record set data.
virtual int netRecordSetLength() const;

// Returns a buffer offset to the given record set.
virtual int recordSetOffset(int recordSetIndex) const;

// Casts the 'this' pointer for changing elements in a const
function
// Non virtual since you can't override based on return type
DtRecordSetPdu* self() const;

protected:

// These are used by recordSetOffset() to save the value and
index of
// the last offset which was computed in case it can be used
again.
int mySaveRecCount;
int mySaveBuffCount;

};
```



```
#endif  
#endif
```

recQueryPdu.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "hostStructs.h"
#include "pdu.h"
#include "exerciseConn.h"

#include "packets/recQueryP.h"

// class DtRecordQueryPdu:
//
// Instances of DtRecordQueryPdu are used to request data records
// from an
// entity.

// DtRecordQueryPdu callback function signature.
typedef void (*DtRecordQueryPduCb)(DtRecordQueryPdu* pdu, void*
usr);

class DT_DLL_VL DtRecordQueryPdu : public DtPdu
{
public:

    // Constructors
    DtRecordQueryPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtRecordQueryPdu(const DtNetRecordQueryPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtRecordQueryPdu(const DtRecordQueryPdu& orig);

    // destructor
    virtual ~DtRecordQueryPdu();

    // Assignment operator
    DtRecordQueryPdu& operator=(const DtRecordQueryPdu& orig);

    // Set/Get originating entity identifier
    virtual void setOriginatingEntityID(const DtEntityIdentifier&
entity);
    virtual const DtEntityIdentifier& originatingEntityID() const;

    // Set/Get receiving entity identifier
    virtual void setReceivingEntityID(const DtEntityIdentifier&
entity);
    virtual const DtEntityIdentifier& receivingEntityID() const;
```

```

// Set/Get request ID
virtual void setRequestID(DtU32 id);
virtual DtU32 requestID() const;

// Set/Get required reliability service
virtual void setReqRelService(enum DtRequiredReliabilityService
service);
virtual DtRequiredReliabilityService reqRelService() const;

// Set/Get the event type
virtual void setEventType(DtEventType type);
virtual DtEventType eventType() const;

// Set/Get the time
virtual void setTime(DtU32 value);
virtual DtU32 time() const;

// Set/Get the number of records to be queried for. Set may
return DtFalse
// if the number of records is too large.
virtual DtBoolean setNumRecords(DtU32 num);
virtual DtU32 numRecords() const;

// Set/Get a record ID. These methods return DtFalse if the index
is out
// of range.
virtual DtBoolean setRecordID(int index, DtDatumParam value);
virtual DtBoolean getRecordID(int index, DtDatumParam& value)
const;

// Print the data contained by this PDU
virtual void printData() const;

public:

// Create a DtRecordQueryPdu.
// Caller is responsible for deletion.
static DtPdu* create(const DtNetPacket* initial,
DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

// Add (register)/Remove (deregister) function to be called when
// pdu is received.
static void addCallback(DtExerciseConn* conn,
DtRecordQueryPduCb cb, void* usr);
static void removeCallback(DtExerciseConn* conn,
DtRecordQueryPduCb cb, void* usr);

protected:

// Returns the internal PDU kind

```

```
virtual DtPduKind internalGetPduKind() const;

// Casts the netPdu element to our network struct
// Non-virtual since you can't override based on return type
virtual DtNetRecordQueryPdu* netPduStruct() const;

// Casts the 'this' pointer for changing elements in a const
function
// Non virtual since you can't override based on return type
DtRecordQueryPdu* self() const;

// Returns a pointer to the first byte of variable length data
virtual char* var() const;

protected:

// Size of PDU with zero data bytes.
static int theStaticSize;

protected:

// Temporary objects for reference returns
DtEntityIdentifier myEntityID;

};
#endif
#endif
```

refAttrCbInfo.h

```

#if DtHLA

#include "refAttrCbList.h"
#include "RTI.hh"

#if DtRTI103
#define ObjectHandle ObjectID
#endif

// class DtReflectAttributeValuesCbInfo:
//
// Instances of DtReflectAttributeValuesCbInfo are used for
// providing
// a callback when attributes change for an object in federation.
// This class is a helper class to DtReflectAttributeValuesCbList.

class DT_DLL_VL DtReflectAttributeValuesCbInfo
{
public:

    // constructor
    DtReflectAttributeValuesCbInfo(RTI::ObjectHandle id,
        DtReflectAttributeValuesCb cb,
        void* usr);

    // destructor
    virtual ~DtReflectAttributeValuesCbInfo();

    // copy constructor
    DtReflectAttributeValuesCbInfo(
        const DtReflectAttributeValuesCbInfo& orig);

    // assignment operator
    DtReflectAttributeValuesCbInfo& operator=(
        const DtReflectAttributeValuesCbInfo& orig);

    // Check whether one callback info is equal to another.
    virtual int match(const DtReflectAttributeValuesCbInfo* const
other) const;

    // Check whether one callback info is equal to another.
    virtual int operator==(const DtReflectAttributeValuesCbInfo&
other) const;

    // Get the object id.
    virtual const RTI::ObjectHandle& objectId() const;

```

```
// Get the callback function.
virtual const DtReflectAttributeValuesCb& cbFunction() const;

// Get the user data.
virtual void* userData() const;

protected:

    RTI::ObjectHandle myId;
    DtReflectAttributeValuesCb myFunc;
    void* myUsr;
};
#endif
```

refAttrCbList.h

```

#if DtHLA

#include "vllist.h"
#include "RTI.hh"

#include "chooseDefines.h"

#if DtRTI103
#define ObjectHandle ObjectID
#endif

// class DtReflectAttributeValuesCbList:
//
// Instances of DtReflectAttributeValuesCbList are used to provide
// a list of callbacks (DtReflectAttributeValuesCbInfo) which are
// called when attributes change for an object in a federation.

typedef void (*DtReflectAttributeValuesCb)(const DtStateMsg&
stateMsg,
void* usr);

class DT_DLL_VL DtReflectAttributeValuesCbList
{
public:

    // default constructor
    DtReflectAttributeValuesCbList();

    // destructor
    virtual ~DtReflectAttributeValuesCbList();

    // copy constructor
    DtReflectAttributeValuesCbList(
        const DtReflectAttributeValuesCbList& orig);

    // assignment operator
    DtReflectAttributeValuesCbList& operator=(
        const DtReflectAttributeValuesCbList& orig);

    // Add/Remove a reflect attribute callback to/from the list.
    // Returns whether add/remove took place.
    virtual int add(RTI::ObjectHandle id, DtReflectAttributeValuesCb
cb,
        void* usr);
    virtual int remove(RTI::ObjectHandle id, DtReflectAttributeVal-
uesCb cb,
        void* usr);

```

```
// Remove all callbacks from the list.
virtual void clear();

// State messages are for a specific object id.
// Callbacks are for a particular object id.
// Call any callbacks in the list for the object id specified in
// the state message.
virtual void apply(const DtStateMsg &msg);
```

protected:

```
// Remove all callbacks from the list.
// Non-virtual since meant to be called from dtor, otherwise
// should call clear().
void dtorClear();

// Copy another callback list into this object's callback list.
// Non-virtual since meant to be called from ctor.
void ctorCopyList(const DtReflectAttributeValuesCbList& orig);
```

protected:

```
    DtList myList;
};
#endif
```


refGenericObj.h

```

#ifdef DtHLA

#include "hReflectedObj.h"
#include "genericSR.h"

// Define the signature for callback functions that can be regis-
// tered with
// DtReflectedGenericObjects or DtReflectedGenericObjectLists. We
// first have to
// forward declare the DtReflectedGenericObject class.
typedef void (*DtGenericObjectCallbackFunction)
    (DtReflectedGenericObject* obj, void* userData);

// class DtReflectedGenericObject:
//
// Instances of DtReflectedGenericObject are used to represent
// "GenericObject"
// objects simulated remotely, that we hear about through the RTI.
// Instances
// are created and deleted automatically by DtReflectedGenericOb-
// jectList in
// response to discover and remove RTI service invocations. This
// class is
// derived from DtDtReflectedObject, which does most of the work.
// The derived
// class mostly exists to provide typesafety, and to configure the
// base class
// object.

class DtReflectedGenericObject : public DtReflectedObject
{
public:

    // constructor
    DtReflectedGenericObject(DtHlaObject* obj, DtExerciseConn* conn);

    // destructor
    virtual ~DtReflectedGenericObject();

    // Returns a pointer to the DtGenericStateRepository, through
    // which you can
    // inspect current state information.
    virtual DtGenericStateRepository* genericStateRep() const;

    // Synonym for genericStateRep()
    virtual DtGenericStateRepository* gsr() const;

```

```
    // Functions to get the next and previous objects in the list.
The "wrap"
    // functions cycle to the beginning/end of the list.
    virtual DtReflectedGenericObject* next() const;
    virtual DtReflectedGenericObject* prev() const;
    virtual DtReflectedGenericObject* wrapNext() const;
    virtual DtReflectedGenericObject* wrapPrev() const;

    // Registration/deregistration of postUpdate callbacks.
    virtual void addPostUpdateCallback(
        DtGenericObjectCallbackFunction cb, void* userData);
    virtual void removePostUpdateCallback(
        DtGenericObjectCallbackFunction cb, void* userData);
};
#endif
```

refGenList.h

```

#if DtHLA

#include "hRefObjList.h"
#include "refGenericObj.h"

// class DtReflectedGenericObjectList:
//
// An instance of this class manages the list of Generic objects
// that are
// simulated remotely, and reflected to us by the RTI. This class
// is derived
// from DtReflectedObjectList, which does most of the work. The
// derived class
// mostly exists to provide typesafety, and to let the base class
// know which
// FOM classes should be managed by this list.

class DtReflectedGenericObjectList : public DtReflectedObjectList
{
public:

    // Constructor
    DtReflectedGenericObjectList(DtExerciseConn* exConn,
        const char* className);

    // destructor
    virtual ~DtReflectedGenericObjectList();

    // The following public functions are not required by VR-Link,
    // but they are
    // strongly recommended. The functions call down to base class
    // versions,
    // and cast to and from more generic types.

    // Look up object by identifier.
    virtual DtReflectedGenericObject* lookup(RTI::ObjectHandle id);

    // Look up object by name (global object designator).
    virtual DtReflectedGenericObject* lookup(const DtGlobalObjectDes-
        ignator& objName);

    // Get the first/last object in the list.
    virtual DtReflectedGenericObject* first() const;
    virtual DtReflectedGenericObject* last() const;

    // Functions that allow a user to register callbacks that will be
    // called

```

```
// when "Generic" objects are added and removed from the list.

// Registration/deregistration of object addition callbacks
virtual void addGenericAdditionCallback(
    DtGenericObjectCallbackFunction cb, void* userData);
virtual void removeGenericAdditionCallback(
    DtGenericObjectCallbackFunction cb, void* userData);

// Registration/deregistration of object removal callbacks
virtual void addGenericRemovalCallback(
    DtGenericObjectCallbackFunction cb, void* userData);
virtual void removeGenericRemovalCallback(
    DtGenericObjectCallbackFunction cb, void* userData);

protected:

    // Override of virtual function that the base class uses to
    create the
    // appropriate kind of DtReflectedObject. Here, we create a
    // DtReflectedGenericObject.
    virtual DtReflectedObject* newReflectedObject(DtHlaObject* obj)
const;
};
#endif
```

refObjCbInfo.h

```

#include "chooseDefines.h"
#include "vllist.h"

// Callback function that takes a DtReflectedObject and user data
typedef void (*DtReflectedObjectCallbackFunction)(
    DtReflectedObject* obj, void* userData);

// class DtReflectedObjectCallbackInfo:
//
// Instances of DtReflectedObjectCallbackInfo are used to contain
// the function
// and user data associated with a DtReflectedObject callback,

class DT_DLL_VL DtReflectedObjectCallbackInfo : public DtListedObj
{
public:
    // default constructor
    DtReflectedObjectCallbackInfo();

    // constructor
    DtReflectedObjectCallbackInfo(
        DtReflectedObjectCallbackFunction cb, void* userData);

    // destructor
    virtual ~DtReflectedObjectCallbackInfo();

    // copy constructor
    DtReflectedObjectCallbackInfo(const DtReflectedObjectCallback-
Info& orig);

    // assignment operator
    DtReflectedObjectCallbackInfo& operator=(
        const DtReflectedObjectCallbackInfo& orig);

    // equality operators
    int operator==(const DtReflectedObjectCallbackInfo& orig) const;
    int operator!=(const DtReflectedObjectCallbackInfo& orig) const;

    // set/get the callback function
    virtual void setCallbackFunction(DtReflectedObjectCallbackFunc-
tion funct);
    virtual DtReflectedObjectCallbackFunction callbackFunction()
const;

    // set/get the user data
    virtual void setUserData(void* data);

```

```
    virtual void* userData() const;

protected:

    DtReflectedObjectCallbackFunction myCallbackFunction;
    void* myUserData;
};
```

refObjMgr.h

```

#if DtHLA

#include "callbacks.h"
#include "objIdHashLst.h"
#include "hlaObjFact.h"

// class DtReflectedHlaObjectManager:
//
// Instances of DtReflectedHlaObjectManager are used to manage the
// creation and
// destruction of HLA objects from the network.

class DT_DLL_VL DtReflectedHlaObjectManager
{
public:

    // constructor
    DtReflectedHlaObjectManager(DtHlaObjectFactory* factory);

    // destructor
    virtual ~DtReflectedHlaObjectManager();

    // Add/remove a callback to be called when an object which meets
the
    // federate's subscription criteria is discovered.
    virtual void addDiscoverObjectCallback(RTI::ObjectClassHandle
handle,
        DtDiscoverObjectCb cb,
        void* usr);
    virtual void removeDiscoverObjectCallback(RTI::ObjectClassHandle
handle,
        DtDiscoverObjectCb cb,
        void* usr);

    // Add/remove a callback to be called when an object has been
// removed from the federation or no longer meets the interest
// criteria of the federate.
    virtual void addRemoveObjectCallback(RTI::ObjectHandle id,
        DtRemoveObjectCb cb,
        void* usr);
    virtual void removeRemoveObjectCallback(RTI::ObjectHandle id,
        DtRemoveObjectCb cb,
        void* usr);

#endif DtRTI103
    // Process the discovery of an object.
    // Creates the object, sets it up for reflecting, and

```

```
// calls discover callbacks.
virtual void processDiscoverObject(RTI::ObjectHandle id,
    RTI::ObjectClassHandle handle);
#else
    // Process the discovery of an object.
    // Creates the object, sets it up for reflecting, and
    // calls discover callbacks.
    virtual void processDiscoverObject(RTI::ObjectHandle id,
        RTI::ObjectClassHandle handle, const char* name);
#endif

    // Process the removal of an object.
    // Cleans up the object from reflecting.
    virtual void processRemoveObject(RTI::ObjectHandle id);

    // Process an update message.
    virtual void processReflectAttributeValues(const DtStateMsg
&msg);

    // Get the list of all HLA objects.
    virtual const DtList& allHlaObjects() const;

    // Get the HLA object based on its identifier.
    virtual DtHlaObject* hlaObject(RTI::ObjectHandle id) const;

protected:

    // Call DtHlaObject::processRemoveObject().
    // Provides friendship towards DtHlaObject to derived classes.
    virtual void objectProcessRemoveObject(DtHlaObject* hlaObj);

    // Call DtHlaObject::callReflect().
    // Provides friendship towards DtHlaObject to derived classes.
    virtual void objectProcessReflect(DtHlaObject* hlaObj,
        const DtStateMsg &msg);

protected:

    // Note: The copy constructor and assignment operator are
protected
    // because there should only be one object manager per exercise
    // connection.
    // These should not be called.

    // copy constructor
    DtReflectedHlaObjectManager(const DtReflectedHlaObjectManager&
orig);

    // assignment operator
    DtReflectedHlaObjectManager& operator=(
```



```
        const DtReflectedHlaObjectManager& orig);  
  
protected:  
  
    // myDiscoverCbList is an array of DtHlaObject::theMaxObject-  
Classes  
    // items.  
    DtList* myDiscoverCbList;  
    DtHlaObjectFactory* myFactory;  
    DtObjectIdHashList myObjectList;  
};  
#endif
```

regCtlCbInfo.h

```
#if DtHLA

#if !DtRTI103

#include "regCtlCbList.h"
#include "RTI.hh"
#include "stateMsg.h"

// class DtRegControlCbInfo:
//
// Instances of DtRegControlCbInfo are used for providing
// a callback when the RTI update control services are invoked.
// This class is a helper class to DtRegControlCbList.

class DT_DLL_VL DtRegControlCbInfo
{
public:

    // constructor
    DtRegControlCbInfo(RTI::ObjectClassHandle h,
        DtRegControlCb cb,
        void* usr);

    // destructor
    virtual ~DtRegControlCbInfo();

    // copy constructor
    DtRegControlCbInfo(
        const DtRegControlCbInfo& orig);

    // assignment operator
    DtRegControlCbInfo& operator=(
        const DtRegControlCbInfo& orig);

    // Check whether one callback info is equal to another.
    virtual int match(const DtRegControlCbInfo* const other) const;

    // Check whether one callback info is equal to another.
    virtual int operator==(const DtRegControlCbInfo& other) const;

    // Get the object class handle.
    virtual const RTI::ObjectClassHandle& objectClass() const;

    // Get the callback function.
    virtual const DtRegControlCb& cbFunction() const;

    // Get the user data.
```

```
    virtual void* userData() const;

protected:

    RTI::ObjectClassHandle myClass;
    DtRegControlCb myFunc;
    void* myUsr;
};
typedef DtRegControlCbInfo DtStartRegsCbInfo;
typedef DtRegControlCbInfo DtStopRegsCbInfo;

#endif
#endif
```

regCtlCbList.h

```
#if DtHLA

#if !DtRTI103

#include "vllist.h"
#include "RTI.hh"
#include "chooseDefines.h"

// class DtRegControlCbList:
//
// Instances of DtRegControlCbList are used to provide
// a list of callbacks (DtRegControlCbInfo) which are
// invoked when a registration control advisory is called by RTI.

typedef void (*DtRegControlCb)(
    RTI::ObjectClassHandle objectClass,
    void* usr);

typedef DtRegControlCb DtStartRegCb;
typedef DtRegControlCb DtStopRegCb;

class DT_DLL_VL DtRegControlCbList
{
public:

    // default constructor
    DtRegControlCbList();

    // destructor
    virtual ~DtRegControlCbList();

    // copy constructor
    DtRegControlCbList(
        const DtRegControlCbList& orig);

    // assignment operator
    DtRegControlCbList& operator=(
        const DtRegControlCbList& orig);

    // Add/Remove a start updates callback to/from the list.
    // Returns whether add/remove took place.
    virtual int add(RTI::ObjectClassHandle h, DtRegControlCb cb,
        void* usr);
    virtual int remove(RTI::ObjectClassHandle h, DtRegControlCb cb,
        void* usr);

    // Remove all callbacks from the list.
```

```
virtual void clear();

// Callbacks are for a particular object class handle.
// Call any callbacks in the list for the object class handle.
virtual void apply(RTI::ObjectClassHandle objectClass);

protected:

    // Remove all callbacks from the list.
    // Non-virtual since meant to be called from dtor, otherwise
    // should call clear().
    void dtorClear();

    // Copy another callback list into this object's callback list.
    // Non-virtual since meant to be called from ctor.
    void ctorCopyList(const DtRegControlCbList& orig);

protected:

    DtList myList;
};
typedef DtRegControlCbList DtStartRegCbList;
typedef DtRegControlCbList DtStopRegCbList;

#endif
#endif
```

remObjCbInfo.h

```
#if DtHLA

#include "RTI.hh"
#include "stateMsg.h"
#include "vLList.h"

// class DtRemoveObjectCbInfo:
//
// Instances of DtRemoveObjectCbInfo are used for providing
// a callback when an object has been removed from the federation or
// no longer meets the interest criteria of the federate.
// This class is a helper class to DtRemoveObjectCbList.

typedef void (*DtRemoveObjectCb)(DtHlaObject* obj, void* usr);

class DT_DLL_VL DtRemoveObjectCbInfo
{
public:

    // constructor
    DtRemoveObjectCbInfo(DtRemoveObjectCb cb, void* usr);

    // destructor
    virtual ~DtRemoveObjectCbInfo();

    // copy constructor
    DtRemoveObjectCbInfo(
        const DtRemoveObjectCbInfo& orig);

    // assignment operator
    DtRemoveObjectCbInfo& operator=(
        const DtRemoveObjectCbInfo& orig);

    // Check whether one callback info is equal to another.
    virtual int match(const DtRemoveObjectCbInfo* const other) const;

    // Check whether one callback info is equal to another.
    virtual int operator==(const DtRemoveObjectCbInfo& other) const;

    // Get the callback function.
    virtual const DtRemoveObjectCb& cbFunction() const;

    // Get the user data.
    virtual void* userData() const;

protected:
```

```
    DtRemoveObjectCb myFunc;  
    void* myUsr;  
};  
#endif
```

removeEntRPdu.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "hostStructs.h"
#include "simanPdu.h"
#include "packets/removeEntRP.h"
#include "exerciseConn.h"

// class DtRemoveEntityRPdu:
//
// Instances of DtRemoveEntityRPdu are used to send/receive
// reliable remove entity PDUs.

// DtRemoveEntityRPdu callback function signature.
typedef void (*DtRemoveEntityRPduCb)(DtRemoveEntityRPdu* pdu, void*
usr);

class DT_DLL_VL DtRemoveEntityRPdu : public DtSimanPdu
{
public:

    // Constructors
    DtRemoveEntityRPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtRemoveEntityRPdu(const DtNetRemoveEntityRPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtRemoveEntityRPdu(const DtEntityIdentifier& to,
        const DtEntityIdentifier& from,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtRemoveEntityRPdu(const DtRemoveEntityRPdu& orig);

    // destructor
    virtual ~DtRemoveEntityRPdu();

    // Assignment operator
    DtRemoveEntityRPdu& operator=(const DtRemoveEntityRPdu& orig);

    // Set/Get the required reliability service.
    virtual void setReqRelService(DtRequiredReliabilityService
value);
    virtual DtRequiredReliabilityService reqRelService() const;

    // Set/Get the request id.
    virtual void setRequestId(int id);
    virtual int requestId() const;

    // Print the data contained by this PDU
    virtual void printData() const;
```


public:

```
// Create a DtRemoveEntityRPdu.  
// Caller is responsible for deletion.  
static DtPdu* create(const DtNetPacket* initial,  
    DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);  
  
// Add (register)/Remove (deregister) function to be called when  
// pdu is received.  
static void addCallback(DtExerciseConn* conn,  
    DtRemoveEntityRPduCb cb, void* usr);  
static void removeCallback(DtExerciseConn* conn,  
    DtRemoveEntityRPduCb cb, void* usr);
```

protected:

```
// Returns the internal PDU kind  
virtual DtPduKind internalGetPduKind() const;
```

protected:

```
static int theStaticSize;
```

```
};  
#endif  
#endif
```

removePdu.h

```
#if !DtHLA

#include "hostStructs.h"
#include "simanPdu.h"

#define DtRemoveStaticSize 28

class DT_DLL_VL DtRemoveEntityPdu;
typedef DtRemoveEntityPdu DtRemoveEntityInteraction;
typedef void (*DtRemoveEntityPduCb)(DtRemoveEntityPdu *pdu, void
*usr);

class DT_DLL_VL DtRemoveEntityPdu : public DtSimanPdu
{
    public:

/***** STANDARD CONSTRUCTORS *****/

    DtRemoveEntityPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtRemoveEntityPdu(const DtNetRemoveEntityPdu *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

/***** ADDITIONAL CONSTRUCTORS *****/

    DtRemoveEntityPdu(const DtEntityIdentifier &to,
        const DtEntityIdentifier &from,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    static DtPdu *create(const DtNetPacket *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    static void addCallback(DtExerciseConn *conn,
        DtRemoveEntityPduCb cb, void *usr);
    static void removeCallback(DtExerciseConn *conn,
        DtRemoveEntityPduCb cb, void *usr);

/***** INSPECTOR FUNCTIONS *****/

#if DtDISVERS != 3
    int requestId() const;
#endif

/***** MUTATOR FUNCTIONS *****/

#if DtDISVERS != 3
    void setRequestId(int id);
    void printData() const;
#endif
}
```

```
protected:  
    DtPduKind internalGetPduKind() const {return DtRemoveEntityPdu-  
Kind;}  
};  
#endif
```

repairPdu.h

```
#if !DtHLA

#include "hostStructs.h"
#include "pdu.h"
#include "mathTypes.h"
#include "packets/logisticsP.h"

#define DtRepairResponseStaticSize 28

class DT_DLL_VL DtRepairResponsePdu;
typedef DtRepairResponsePdu DtRepairResponseInteraction;
typedef void (*DtRepairResponsePduCb)(DtRepairResponsePdu *pdu,
void *usr);

class DT_DLL_VL DtRepairResponsePdu : public DtPdu
{
public:

/***** STANDARD CONSTRUCTORS *****/

    DtRepairResponsePdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtRepairResponsePdu(const DtNetRepairResponsePdu *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    static DtPdu *create(const DtNetPacket *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    static void addCallback(DtExerciseConn *conn,
        DtRepairResponsePduCb cb, void *usr);
    static void removeCallback(DtExerciseConn *conn,
        DtRepairResponsePduCb cb, void *usr);

/***** INSPECTOR FUNCTIONS *****/

    const DtEntityIdentifier &receivingId() const;
    const DtEntityIdentifier &repairingId() const;
    DtRepairResult repairResult() const;

/***** MUTATOR FUNCTIONS *****/

    void setReceivingId(const DtEntityIdentifier &id);
    void setRepairingId(const DtEntityIdentifier &id);
    void setRepairResult(DtRepairResult result);

    void printData() const;
protected:
    DtPduKind internalGetPduKind() const {return DtRepairResponseP-
duKind;}
```

```

    DtEntityIdentifier theId;
};
/*****
/***** REPAIR COMPLETE PDU *****/
/*****/

#define DtRepairCompleteStaticSize 28

typedef void (*DtRepairCompletePduCb)(DtRepairCompletePdu *pdu,
void *usr);

class DT_DLL_VL DtRepairCompletePdu : public DtPdu
{
public:

/***** STANDARD CONSTRUCTORS *****/

    DtRepairCompletePdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtRepairCompletePdu(const DtNetRepairCompletePdu *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    static DtPdu *create(const DtNetPacket *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    static void addCallback(DtExerciseConn *conn,
        DtRepairCompletePduCb cb, void *usr);
    static void removeCallback(DtExerciseConn *conn,
        DtRepairCompletePduCb cb, void *usr);

/***** INSPECTOR FUNCTIONS *****/

    const DtEntityIdentifier &receivingId() const;
    const DtEntityIdentifier &repairingId() const;
    DtRepairType repairType() const;

/***** MUTATOR FUNCTIONS *****/

    void setReceivingId(const DtEntityIdentifier &id);
    void setRepairingId(const DtEntityIdentifier &id);
    void setRepairType(DtRepairType type);

    void printData() const;
protected:
    DtPduKind internalGetPduKind() const {return DtRepairCompleteP-
duKind;}
    DtEntityIdentifier theId;
};

#endif

```

resupplyPdu.h

```
#if !DtHLA

#include "hostStructs.h"
#include "pdu.h"
#include "mathTypes.h"
#include "packets/logisticsP.h"

#define DtResupplyCancelStaticSize 24

class DT_DLL_VL DtResupplyCancelPdu;
typedef DtResupplyCancelPdu DtResupplyCancelInteraction;
typedef void (*DtResupplyCancelPduCb)(DtResupplyCancelPdu *pdu,
void *usr);

class DT_DLL_VL DtResupplyCancelPdu : public DtPdu
{
    public:

/***** STANDARD CONSTRUCTORS *****/

    DtResupplyCancelPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtResupplyCancelPdu(const DtNetResupplyCancelPdu *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtResupplyCancelPdu(const DtEntityIdentifier &receiveId,
        const DtEntityIdentifier &supplyId,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    static DtPdu *create(const DtNetPacket *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    static void addCallback(DtExerciseConn *conn,
        DtResupplyCancelPduCb cb, void *usr);
    static void removeCallback(DtExerciseConn *conn,
        DtResupplyCancelPduCb cb, void *usr);

/***** INSPECTOR FUNCTIONS *****/

    const DtEntityIdentifier &receivingId() const;
    const DtEntityIdentifier &supplyingId() const;

/***** MUTATOR FUNCTIONS *****/

    void setReceivingId(const DtEntityIdentifier &id);
    void setSupplyingId(const DtEntityIdentifier &id);

    void printData() const;
protected:
```

```

    DtPduKind internalGetPduKind() const {return DtResupplyCancelP-
duKind;}
    DtEntityIdentifier theId;
};
/*****/
/***** RESUPPLY OFFER PDU *****/
/*****/

#define DtResupplyOfferStaticSize 28
#define DtSUPPLY_VALID 1
#define DtSUPPLY_BAD_INDEX 0

typedef DtResupplyOfferPdu DtResupplyOfferInteraction;
typedef void (*DtResupplyOfferPduCb)(DtResupplyOfferPdu *pdu, void
*usr);

class DT_DLL_VL DtResupplyOfferPdu : public DtPdu
{
public:

/***** STANDARD CONSTRUCTORS *****/

    DtResupplyOfferPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtResupplyOfferPdu(const DtNetResupplyOfferPdu *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

/***** ADDITIONAL CONSTRUCTORS *****/

    DtResupplyOfferPdu(const DtEntityIdentifier &receiver,
        const DtEntityIdentifier &supplier,
        const DtEntityType &supply,
        float quantity,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    static DtPdu *create(const DtNetPacket *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    static void addCallback(DtExerciseConn *conn,
        DtResupplyOfferPduCb cb, void *usr);
    static void removeCallback(DtExerciseConn *conn,
        DtResupplyOfferPduCb cb, void *usr);

/***** INSPECTOR FUNCTIONS *****/

    const DtEntityIdentifier &receivingId() const;
    const DtEntityIdentifier &supplyingId() const;
    int numSupplyTypes() const;
    const DtEntityType &nthSupply(int supplyIndex,
        float *quantity = NULL,
        int *retCode = NULL) const;
    float supplyQuantity(const DtEntityType &supply) const;

```

```
/****** MUTATOR FUNCTIONS *****/

void      setReceivingId(const DtEntityIdentifier &requester);
void      setSupplyingId(const DtEntityIdentifier &servicer);
void      setNumSupplyTypes(int numTypes);
void      setNthSupply(int supplyIndex,
                        const DtEntityType &supply,
                        float quantity,
                        int *retCode = NULL);

void printData() const;
protected:
    DtPduKind internalGetPduKind() const {return DtResupplyOfferPdu-
Kind;}
    int indexInBounds(int ind) const;
    DtEntityIdentifier theId;
    DtEntityType supType;
};

/******
/****** RESUPPLY RECEIVED PDU *****/
/******

#define DtResupplyReceivedStaticSize 28

typedef DtResupplyReceivedPdu DtResupplyRecInteraction;
typedef void (*DtResupplyReceivedPduCb)(DtResupplyReceivedPdu *pdu,
void *usr);

class DT_DLL_VL DtResupplyReceivedPdu : public DtPdu
{
public:

/****** STANDARD CONSTRUCTORS *****/

    DtResupplyReceivedPdu(DtBufferPtr buffer =
DtUSE_INTERNAL_BUFFER);
    DtResupplyReceivedPdu(const DtNetResupplyReceivedPdu *initial,
DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

/****** ADDITIONAL CONSTRUCTORS *****/

    DtResupplyReceivedPdu(const DtEntityIdentifier &receiver,
                        const DtEntityIdentifier &supplier,
                        const DtEntityType &supply,
                        float quantity,
                        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    static DtPdu *create(const DtNetPacket *initial,
```



```

        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
static void addCallback(DtExerciseConn *conn,
        DtResupplyReceivedPduCb cb, void *usr);
static void removeCallback(DtExerciseConn *conn,
        DtResupplyReceivedPduCb cb, void *usr);

/***** INSPECTOR FUNCTIONS *****/

const DtEntityIdentifier &receivingId() const;
const DtEntityIdentifier &supplyingId() const;
int numSupplyTypes() const;
const DtEntityType &nthSupply(int supplyIndex,
        float *quantity = NULL,
        int *retCode = NULL) const;
float supplyQuantity(const DtEntityType &supply) const;

/***** MUTATOR FUNCTIONS *****/

void      setReceivingId(const DtEntityIdentifier &requester);
void      setSupplyingId(const DtEntityIdentifier &servicer);
void      setNumSupplyTypes(int numTypes);
void      setNthSupply(int supplyIndex,
        const DtEntityType &supply,
        float quantity,
        int *retCode = NULL);

void printData() const;

protected:
    DtPduKind internalGetPduKind() const {return DtResupplyReceived-
PduKind;}
    int indexInBounds(int ind) const;
    DtEntityIdentifier theId;
    DtEntityType supType;
};

#endif

```

rprDecFact.h

```
#if DtHLA

#include "decFactory.h"

// class DtRprStateDecoderFactory:
//
// Instances of DtRprStateDecoderFactory are used to create a Real-
// time
// Platform Reference (RPR) Federation Object Model (FOM) decoder.

// If you define DtSIMPLE_RPR to 1, and include this file, then this
// file
// defines a class called DtSimpleRprStateDecoderFactory, which
// contains
// mappings for only the subset of RPR FOM that is used by most MAK
// tools.
#if DtSIMPLE_RPR
#define DtRprStateDecoderFactory DtSimpleRprStateDecoderFactory
#endif

class DT_DLL_VL DtRprStateDecoderFactory : public DtStateDecoderFac-
tory
{
public:

    // constructor
    DtRprStateDecoderFactory(DtExerciseConn* exConn);

    // destructor
    virtual ~DtRprStateDecoderFactory();

    // copy constructor
    DtRprStateDecoderFactory(const DtRprStateDecoderFactory& orig);

    // assignment operator
    DtRprStateDecoderFactory& operator=(const DtRprStateDecoderFac-
tory& orig);
};
#endif
```

rprEncFact.h

```

#if DtHLA

#include "encFactory.h"

// class DtRprStateEncoderFactory:
//
// Instances of DtRprStateEncoderFactory are used to create encoders
// which
// know how to encode state repositories for RPR (Real-time Platform
// Reference) FOM specific objects into an HLA state message.

// If you define DtSIMPLE_RPR to 1, and include this file, then this
// file
// defines a class called DtSimpleRprStateEncoderFactory, which
// contains
// mappings for only the subset of RPR FOM that is used by most MAK
// tools.
#if DtSIMPLE_RPR
#define DtRprStateEncoderFactory DtSimpleRprStateEncoderFactory
#endif

class DT_DLL_VL DtRprStateEncoderFactory : public DtStateEncoderFac-
tory
{
public:

    // constructor
    // Adds RPR FOM specific encoders.
    DtRprStateEncoderFactory(DtExerciseConn* exConn);

    // destructor
    virtual ~DtRprStateEncoderFactory();

    // copy constructor
    DtRprStateEncoderFactory(const DtRprStateEncoderFactory& orig);

    // assignment operator
    DtRprStateEncoderFactory& operator=(
        const DtRprStateEncoderFactory& orig);
};
#endif

```

rprFomMap.h

```
#if DtHLA

#include "emptyFomMap.h"
#include "chooseDefines.h"

// class DtRprFomMapper:
//
// Instances of DtRprFomMapper are DtFomMappers configured to map
// between
// VR-Link classes and functionality to RPR FOM classes, attributes
// and
// parameters.

// If you define DtSIMPLE_RPR to 1, and include this file, then this
// file
// defines a class called DtSimpleFomMapper - a FOM Mapper that
// contains
// mappings for only the subset of RPR FOM objects/interactions that
// are used
// by most MAK tools.
#if DtSIMPLE_RPR
#define DtRprFomMapper DtSimpleRprFomMapper
#endif

class DT_DLL_VL DtRprFomMapper : public DtEmptyFomMapper
{
public:

    // default constructor
    DtRprFomMapper(double vers = 0.8);

    // destructor
    virtual ~DtRprFomMapper();

    // Virtual function override.  init actually initializes most
    data.  It is
    // called from DtExerciseConn constructor.
    virtual void init(DtExerciseConn* conn);

public:

    // Creator function - returns new instance of DtRprFomMapper.
    usr is cast
    // to a pointer to double.  If the pointer is NULL, or the value
    pointed to
    // is 0.0, the default version is used in the DtRprFomMapper
    constructor.
```

```
    // Otherwise the value is passed to the DtRprFomMapper
    constructor as the
    // version number.
    static DtFomMapper* create(void* usr);
```

protected:

```
    // copy constructor
    DtRprFomMapper(const DtRprFomMapper& orig);

    // assignment operator
    DtRprFomMapper& operator=(const DtRprFomMapper& orig);

    // Set myInterClasses to contain default mappings.
    virtual void initInterClasses();

    // Set myObjClassesClasses to contain default mappings.
    virtual void initObjClasses();

    // Setup the object class choosers.
    virtual void initObjChoosers();

    // Setup the interaction class choosers.
    virtual void initInterChoosers();
```

protected:

```
    // The default entity class chooser function.
    static char* dfltEntityClassChooser(
        const char* vrlinkClassName,
        DtExerciseConn* exConn,
        void* usr);

    // The entity class chooser function for older RPR FOM versions.
    static char* entityClassChooserPre08(
        const char* vrlinkClassName,
        DtExerciseConn* exConn,
        void* usr);

    // The default emitter beam class chooser function.
    static char* dfltEmitterBeamClassChooser(
        const char* vrlinkClassName,
        DtExerciseConn* exConn,
        void* usr);

    // The default emitter beam class chooser function.
    static char* emitterBeamClassChooserPre08(
        const char* vrlinkClassName,
        DtExerciseConn* exConn,
        void* usr);
```

```
// The default signal class chooser function.  
static char* dfltSignalClassChooser(  
    DtExerciseConn* exConn, const DtInteraction& inter);  
};  
#endif
```

rprlDecFact.h

```

#if DtHLA

#include "intDecFact.h"

// class DtRprInteractionDecoderFactory:
//
// Instances of DtRprInteractionDecoderFactory are
// DtInteractionDecoderFactories configured with the VR-Link's
// default set of Encoders.

// If you define DtSIMPLE_RPR to 1, and include this file, then this
// file
// defines a class called DtSimpleRprInteractionDecoderFactory,
// which contains
// mappings for only the subset of RPR FOM that is used by most MAK
// tools.
#if DtSIMPLE_RPR
#define DtRprInteractionDecoderFactory DtSimpleRprInteractionDe-
coderFactory
#endif

class DT_DLL_VL DtRprInteractionDecoderFactory :
    public DtInteractionDecoderFactory
{
public:

    // constructor
    DtRprInteractionDecoderFactory(DtExerciseConn* exConn);

    // destructor
    virtual ~DtRprInteractionDecoderFactory();

    // copy constructor
    DtRprInteractionDecoderFactory(
        const DtRprInteractionDecoderFactory& orig);

    // assignment operator
    DtRprInteractionDecoderFactory& operator=(
        const DtRprInteractionDecoderFactory& orig);
};

#endif
#endif

```

rprlEncFact.h

```
#if DtHLA

#include "intEncFact.h"

// class DtRprInteractionEncoderFactory:
//
// Instances of DtRprInteractionEncoderFactory are
// DtInteractionEncoderFactories configured with the VR-Link's
// default
// set of Encoders.

// If you define DtSIMPLE_RPR to 1, and include this file, then this
// file
// defines a class called DtSimpleRprInteractionEncoderFactory,
// which contains
// mappings for only the subset of RPR FOM that is used by most MAK
// tools.
#if DtSIMPLE_RPR
#define DtRprInteractionEncoderFactory DtSimpleRprInteractionEn-
coderFactory
#endif

class DT_DLL_VL DtRprInteractionEncoderFactory :
    public DtInteractionEncoderFactory
{
public:

    // constructor
    // Adds RPR FOM specific encoders.
    DtRprInteractionEncoderFactory(DtExerciseConn* exConn);

    // destructor
    virtual ~DtRprInteractionEncoderFactory();

    // copy constructor
    DtRprInteractionEncoderFactory(
        const DtRprInteractionEncoderFactory& orig);

    // assignment operator
    DtRprInteractionEncoderFactory& operator=(
        const DtRprInteractionEncoderFactory& orig);
};

#endif
#endif
```


rprInterFact.h

```

#if DtHLA

#include "hInterFactory.h"

// class DtRprInteractionFactory:
//
// Instances of DtRprInteractionFactory are used to create RPR
// (Real-time Platform Reference) FOM interactions.

// If you define DtSIMPLE_RPR to 1, and include this file, then this
// file
// defines a class called DtSimpleRprInteractionFactory, which
// contains
// mappings for only the subset of RPR FOM that is used by most MAK
// tools.
#if DtSIMPLE_RPR
#define DtRprInteractionFactory DtSimpleRprInteractionFactory
#endif

class DT_DLL_VL DtRprInteractionFactory : public DtInteractionFac-
tory
{
public:

    // constructor
    // Adds creators for the RPR FOM interactions.
    DtRprInteractionFactory(DtExerciseConn* exConn);

    // destructor
    virtual ~DtRprInteractionFactory();

    // copy constructor
    DtRprInteractionFactory(const DtRprInteractionFactory& orig);

    // assignment operator
    DtRprInteractionFactory& operator=(const DtRprInteractionFactory&
orig);
};
#endif

```

rprObjFact.h

```
#if DtHLA

#include "hlaObjFact.h"

#include "chooseDefines.h"

// class DtRprHlaObjectFactory:
//
// Instances of DtRprHlaObjectFactory are used to create RPR (Real-
// time
// Platform Reference) FOM HLA objects.

class DT_DLL_VL DtRprHlaObjectFactory : public DtHlaObjectFactory
{
public:

    // constructor
    DtRprHlaObjectFactory(DtExerciseConn* conn);

    // destructor
    virtual ~DtRprHlaObjectFactory();

    // copy constructor
    DtRprHlaObjectFactory(const DtRprHlaObjectFactory& orig);

    // assignment operator
    DtRprHlaObjectFactory& operator=(const DtRprHlaObjectFactory&
orig);
};
#endif
```

script.h

```
#include "vllist.h"
#include "vcPdu.h"

typedef struct
{
    DtTime t;
    DtViewControlPdu *pdu;
} StealthInstruction;

/* A DtScript is a DtList of StealthInstructions */
class DT_DLL_VL DtScript : public DtList
{
public:
    DtScript(char *fname);
    DtScript() {}
    ~DtScript();
    void addInstruction(DtTime time, DtViewerCommand command,
        char *args);
    void addInstruction(DtTime time, DtViewControlPdu *pdu);
    void clear();
};
```

servReqPdu.h

```
#if !DtHLA

#ifndef DtServiceRequestPdu_H_
#define DtServiceRequestPdu_H_

#include "pdu.h"
#include "packets/logisticsP.h"
#include "hostStructs.h"

#define DtServiceRequestStaticSize 28
#define DtSUPPLY_VALID 1
#define DtSUPPLY_BAD_INDEX 0

class DT_DLL_VL DtServiceRequestPdu;
typedef DtServiceRequestPdu DtServiceRequestInteraction;
typedef void (*DtServiceRequestPduCb)(DtServiceRequestPdu *pdu,
void *usr);

class DT_DLL_VL DtServiceRequestPdu : public DtPdu
{
public:

/***** STANDARD CONSTRUCTORS *****/

DtServiceRequestPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
DtServiceRequestPdu(const DtNetServiceRequestPdu *initial,
DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

static DtPdu *create(const DtNetPacket *initial,
DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
static void addCallback(DtExerciseConn *conn,
DtServiceRequestPduCb cb, void *usr);
static void removeCallback(DtExerciseConn *conn,
DtServiceRequestPduCb cb, void *usr);

/***** ADDITIONAL CONSTRUCTORS *****/

DtServiceRequestPdu(const DtEntityIdentifier &requester,
const DtEntityIdentifier &servicer,

#ifdef _DOS
/* This is some screwed up stuff. This is a hack that will go down in
infamy. This is how the world ends, not with a hack, but with a
compiler bug. Someday, children, all the weirdness will be gone,
and
this ifdef can be taken out. */
DtServiceType serviceType = (DtServiceType) 2,
```

```

#else
    DtServiceType serviceType = DtServiceTypeRepair,
#endif
    DtBufferPtr another = DtUSE_INTERNAL_BUFFER);

    DtServiceRequestPdu(const DtEntityIdentifier &requester,
        const DtEntityIdentifier &servicer,
        const DtEntityType &supply,
        float quantity,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

/***** INSPECTOR FUNCTIONS *****/

const DtEntityIdentifier &requestingId() const;
const DtEntityIdentifier &servicingId() const;
DtServiceType serviceType() const;
int numSupplyTypes() const;
const DtEntityType &nthSupply(int supplyIndex,
    float *quantity = NULL,
    int *retCode = NULL) const;
float supplyQuantity(const DtEntityType &supply) const;

/***** MUTATOR FUNCTIONS *****/

void    setRequestingId(const DtEntityIdentifier &requester);
void    setServicingId(const DtEntityIdentifier &servicer);
void    setServiceType(DtServiceType service);
void    setNumSupplyTypes(int numTypes);
void    setNthSupply(int supplyIndex,
    const DtEntityType &supply,
    float quantity,
    int *retCode = NULL);

void printData() const;
protected:
    DtPduKind internalGetPduKind() const {return DtServiceRequestP-
duKind;}
    int indexInBounds(int ind) const;
    DtEntityIdentifier reqId;
    DtEntityIdentifier servId;
    DtEntityType supType;
};
#endif

```

setDataPdu.h

```
#if !DtHLA

#include "hostStructs.h"
#include "pduWithData.h"
#include "datumIdType.h"

#define DtSetDataStaticSize 40

class DT_DLL_VL DtSetDataPdu;
typedef DtSetDataPdu DtSetDataInteraction;
typedef void (*DtSetDataPduCb)(DtSetDataPdu *pdu, void *usr);

class DT_DLL_VL DtSetDataPdu : public DtPduWithData
{
public:

/***** STANDARD CONSTRUCTORS *****/

    DtSetDataPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtSetDataPdu(const DtNetSetDataPdu *initial,
                  DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    static DtPdu *create(const DtNetPacket *initial,
                        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    static void addCallback(DtExerciseConn *conn,
                           DtSetDataPduCb cb, void *usr);
    static void removeCallback(DtExerciseConn *conn,
                               DtSetDataPduCb cb, void *usr);

/***** INSPECTOR FUNCTIONS *****/

    unsigned long requestId() const;

/***** MUTATOR FUNCTIONS *****/

    void setRequestId(unsigned long id);

    void printData() const;

protected:

    DtPduKind internalGetPduKind() const {return DtSetDataPduKind;}
    DtNetDataPduTail *dataSection() const;
};
#endif
```

setDataRPdu.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "hostStructs.h"
#include "pduWithData.h"
#include "datumIdType.h"
#include "exerciseConn.h"
#include "packets/setDataRP.h"

// class DtSetDataRPdu:
//
// Instances of DtSetDataRPdu are used to send/receive
// reliable set data PDUs.

// DtSetDataRPdu callback function signature.
typedef void (*DtSetDataRPduCb)(DtSetDataRPdu* pdu, void* usr);

class DT_DLL_VL DtSetDataRPdu : public DtPduWithData
{
public:

    // Constructors
    DtSetDataRPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtSetDataRPdu(const DtNetSetDataRPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtSetDataRPdu(const DtSetDataRPdu& orig);

    // destructor
    virtual ~DtSetDataRPdu();

    // Assignment operator
    DtSetDataRPdu& operator=(const DtSetDataRPdu& orig);

    // Set/Get the request id.
    virtual void setRequestId(unsigned long id);
    virtual unsigned long requestId() const;

    // Set/Get the required reliability service.
    virtual void setReqRelService(DtRequiredReliabilityService
value);
    virtual DtRequiredReliabilityService reqRelService() const;

    // Print the data contained by this PDU
    virtual void printData() const;

public:

```

```
// Create a DtSetDataRPdu.
// Caller is responsible for deletion.
static DtPdu* create(const DtNetPacket* initial,
    DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

// Add (register)/Remove (deregister) function to be called when
// pdu is received.
static void addCallback(DtExerciseConn* conn,
    DtSetDataRPduCb cb, void* usr);
static void removeCallback(DtExerciseConn* conn,
    DtSetDataRPduCb cb, void* usr);

protected:

    // Returns the internal PDU kind
    virtual DtPduKind internalGetPduKind() const;

    // Returns the data portion of the net structure.
    virtual DtNetDataPduTail* dataSection() const;

protected:

    static int theStaticSize;

};
#endif
#endif
```


setRecordPdu.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "hostStructs.h"
#include "recordSetPdu.h"
#include "exerciseConn.h"

#include "packets/setRecordP.h"

// class DtSetRecordPdu:
//
// Instances of DtSetRecordPdu are used to initialize or change
// internal
// parameter values using record sets.

// DtSetRecordPdu callback function signature.
typedef void (*DtSetRecordPduCb)(DtSetRecordPdu* pdu, void* usr);

class DT_DLL_VL DtSetRecordPdu : public DtRecordSetPdu
{
public:

    // Constructors
    DtSetRecordPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtSetRecordPdu(const DtNetSetRecordPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtSetRecordPdu(const DtSetRecordPdu& orig);

    // destructor
    virtual ~DtSetRecordPdu();

    // Assignment operator
    DtSetRecordPdu& operator=(const DtSetRecordPdu& orig);

    // Set/Get originating entity identifier
    virtual void setOriginatingEntityID(const DtEntityIdentifier&
entity);
    virtual const DtEntityIdentifier& originatingEntityID() const;

    // Set/Get receiving entity identifier
    virtual void setReceivingEntityID(const DtEntityIdentifier&
entity);
    virtual const DtEntityIdentifier& receivingEntityID() const;

    // Set/Get request ID

```

```
virtual void setRequestID(DtU32 id);
virtual DtU32 requestID() const;

// Set/Get required reliability service
virtual void setReqRelService(enum DtRequiredReliabilityService
service);
virtual DtRequiredReliabilityService reqRelService() const;

// Print the data contained by this PDU
virtual void printData() const;

public:

// Create a DtSetRecordPdu.
// Caller is responsible for deletion.
static DtPdu* create(const DtNetPacket* initial,
DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

// Add (register)/Remove (deregister) function to be called when
// pdu is received.
static void addCallback(DtExerciseConn* conn,
DtSetRecordPduCb cb, void* usr);
static void removeCallback(DtExerciseConn* conn,
DtSetRecordPduCb cb, void* usr);

protected:
// Returns a pointer to the beginning of the record set data
struct.
virtual DtNetRecordSetPdu* netRecordSetPduStruct() const;

// Returns the internal PDU kind
virtual DtPduKind internalGetPduKind() const;

// Casts the netPdu element to our network struct
// Non-virtual since you can't override based on return type
virtual DtNetSetRecordPdu* netPduStruct() const;

// Casts the 'this' pointer for changing elements in a const
function
// Non virtual since you can't override based on return type
DtSetRecordPdu* self() const;

protected:

// Size of PDU with zero data bytes.
static int theStaticSize;

protected:

// Temporary objects for reference returns
```

```
DtEntityIdentifier myEntityID;  
  
};  
#endif  
#endif
```

signalPdu.h

```
#if !DtHLA

#include "hostStructs.h"
#include "pdu.h"
#include "packets/signalP.h"
#include "interaction.h"
#define DtSignalStaticSize 32

class DT_DLL_VL DtSignalPdu;
typedef DtSignalPdu DtSignalInteraction;
typedef void (*DtSignalInteractionCb)(DtSignalInteraction *pdu,
void *usr);

class DT_DLL_VL DtSignalPdu : public DtInteraction
{
public:

/***** STANDARD CONSTRUCTORS *****/

    DtSignalPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtSignalPdu(const DtNetSignalPdu *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    static DtPdu *create(const DtNetPacket *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    static void addCallback(DtExerciseConn *conn,
        DtSignalInteractionCb cb, void *usr);
    static void removeCallback(DtExerciseConn *conn,
        DtSignalInteractionCb cb, void *usr);

/***** INSPECTOR FUNCTIONS *****/

    const DtEntityIdentifier &entityId() const;
    unsigned int radioId() const;
    DtEncodingClass encodingClass() const;
    unsigned int encodingType() const;
    long sampleRate() const;
    int numDataBits() const;
    int numDataBytes() const;
    int numSamples() const;
    unsigned char dataByte(int byteNum) const;
    void getData(unsigned char *buff) const;
#if DtDISVERS != 3
    int tdlType() const;
#endif

/***** MUTATOR FUNCTIONS *****/
```

```
void setEntityId(const DtEntityIdentifier &id);
void setRadioId(unsigned int id);
void setEncodingClass(DtEncodingClass encodingClass);
void setEncodingType(unsigned int codeType);
void setSampleRate(long rate);
void setNumDataBits(int lengthInBits);
void setNumDataBytes(int lengthInBytes);
void setNumSamples(int numSamples);
void setDataByte(int byteNum, unsigned char data);
void setData(unsigned char *data, int numBytes);
#if DtDISVERS != 3
    void setTdlType(int type);
#endif

    void printData() const;
protected:
    DtPduKind internalGetPduKind() const {return DtSignalPduKind;}
    DtEntityIdentifier theId;
};
#endif // !DtHLA
```

simanPdu.h

```
#if !DtHLA

#include "pdu.h"
#include "packets/simanP.h"
#include "hostStructs.h"
#include "interaction.h"

class DT_DLL_VL DtSimanPdu : public DtInteraction
{
    public:

    /***** INSPECTOR FUNCTIONS *****/

        virtual const DtEntityIdentifier &senderId() const;
        virtual const DtEntityIdentifier &receiverId() const;

    /***** MUTATOR FUNCTIONS *****/

        virtual void setSenderId(const DtEntityIdentifier &id);
        virtual void setReceiverId(const DtEntityIdentifier &id);

        virtual void printData() const;

    protected:
        DtEntityIdentifier cache;
};
#endif
```

simRprFomMap.h

```
#if DtHLA

// This header file provides a definition for the class DtSimpleRpr-
// FomMapper,
// a FOM Mapper that includes mappings for only the subset of the
// RPR FOM
// (with MAK extensions) that is used by most MAK applications. It
// includes
// all entity classes, aggregates, emissions, as well as fire, deto-
// nate,
// collision, stealth control and logger control interactions.
// Using this FOM
// Mapper rather than the default RPR FOM Mapper significantly cuts
// down
// executable size.

#ifndef DtSIMPLE_RPR
#define DtSIMPLE_RPR 1
#else
#define DtSIMPLE_RPR_WAS_DEFINED 1
#endif

#ifdef DtRprFomMapper_H_
#undef DtRprFomMapper_H_
#define DtRprFomMapper_H_WAS_DEFINED
#endif

#include "rprFomMap.h"

#undef DtRprFomMapper

#ifndef DtRprFomMapper_H_WAS_DEFINED
#undef DtRprFomMapper_H_
#endif

#ifndef DtSIMPLE_RPR_WAS_DEFINED
#undef DtSIMPLE_RPR
#endif

#endif
```

smoother.h

```
#include "deadReckoner.h"
#include "smoothPosWDR.h"
#include "smoothOriWDR.h"

// class DtSmoother:
//
// Instances of DtSmoother are used to provide approximated
// position and orientation information for entities.
// Position and orientation information is approximated over time
// from the last set absolute (non-approximated) information.
// When a new position is set, the current position moves smoothly
// to reflect the new information, rather than jumping to the new
// absolute position.

class DT_DLL_VL DtSmoother : public DtDeadReckoner
{
public:

    // constructor
    DtSmoother();

    // destructor
    virtual ~DtSmoother();

    // copy constructor
    DtSmoother(const DtSmoother& orig);

    // assignment operator
    DtSmoother& operator=(const DtSmoother& orig);

    // Create a copy of this object. Memory is allocated and it is
the
    // responsibility of the caller to free the memory via delete.
    virtual DtApproximator* clone() const;

    // Set absolute location.
    virtual void setLocation(DtConstVector loc,
        DtTime t = DtGetSimTime());
    // Get the approximated location at the specified time.
    virtual DtConstVector approxLocation(
        DtTime t = DtGetSimTime()) const;

    // Set absolute velocity.
    virtual void setVelocity(DtConstVector vel,
        DtTime t = DtGetSimTime());
    // Get the approximated velocity at the specified time.
    virtual DtConstVector approxVelocity(
```



```

        DtTime t = DtGetSimTime()) const;

// Set absolute orientation.
virtual void setOrientation(const DtTaitBryan& ori,
        DtTime t = DtGetSimTime());
// Get approximated orientation (euler angles) at the specified
time.
virtual const DtTaitBryan& approxEuler(
        DtTime t = DtGetSimTime()) const;

// Set/Get absolute acceleration.
virtual void setAcceleration(DtConstVector acc,
        DtTime t = DtGetSimTime());
virtual DtConstVector acceleration(
        DtTime t = DtGetSimTime()) const;

// Set/Get absolute rotational velocity.
virtual void setRotationalVelocity(DtConstVector vel,
        DtTime t = DtGetSimTime());
virtual DtConstVector rotationalVelocity(
        DtTime t = DtGetSimTime()) const;

// Get the approximated data body to geocentric coordinate
matrix.
virtual DtConstDcm approxBodyToGeoc(
        DtTime t = DtGetSimTime()) const;

// Set/Get the individual smoothing switch. In order for
smoothing
// to be enabled, the "master" smoothing-enabled switch must be
on, AND
// the individual smoothing switch (accessed through these func-
tions),
// must be on. Individual smoothing switch defaults to on.
virtual void setSmoothEnable(DtBoolean on);
virtual DtBoolean smoothEnable() const;

// Set/Get smoothing period - the amount of time over
// which smoothing takes place. Pass a value of -1 to indicate
// use of the default smoothing period.
virtual void setSmoothPeriod(DtTime smoothPeriod);
virtual DtTime smoothPeriod() const;

public:

// Set/Get the default smoothing period
static void setDfltSmoothPeriod(DtTime smoothPeriod);
static DtTime dfltSmoothPeriod();

// Set/Get the value of the "master" smoothing-enabled switch.

```

```
// Master smoothing switch defaults to on.
static void setMasterSmoothEnable(DtBoolean onOff);
static DtBoolean masterSmoothEnable();

static DtTime theDfltSmoothPeriod;

protected:

// provide inputs to the actual smoothing algorithm
virtual void feedSmoothing();

// Get rid of constness on the "this" pointer in order to access
// non-const functions from const member functions (such as
simple
// accessors). This should be used sparingly since it bypasses
// the const protection.
// Note: This is essentially like an operator and is
//       non-virtual since can't override based on return type.
//       Allows derived classes to also define this member.
DtSmoother* self() const;

protected:

DtSmoothPosWithDr mySpos;
DtSmoothOriWithDr mySorient;
};
```

smoothOriWDR.h

```
#include "smoothUtil.h"
// class DtSmoothOriWithDr:
//
// Instances of DtSmoothOriWithDr are

class DT_DLL_VL DtSmoothOriWithDr : public DtSmoothOrient
{
public:

    // default constructor
    DtSmoothOriWithDr(DtDeadReckoner* dr, DtTime smoothT);

    // destructor
    virtual ~DtSmoothOriWithDr();

protected:

    // copy constructor
    DtSmoothOriWithDr(const DtSmoothOriWithDr& orig);

    // assignment operator
    DtSmoothOriWithDr& operator=(const DtSmoothOriWithDr& orig);

    // The default action is to dead-reckon
    virtual void dfltAction(DtTime simTime,
        void *usr, DtDcmRef os);
};
```

smoothPosWDR.h

```
#include "smoothUtil.h"

// class DtSmoothPosWithDr:
//
// Instances of DtSmoothPosWithDr are

class DT_DLL_VL DtSmoothPosWithDr : public DtSmoothPos
{
public:

    // default constructor
    DtSmoothPosWithDr(DtDeadReckoner* dr, DtTime smoothT);

    // destructor
    virtual ~DtSmoothPosWithDr();

protected:

    // copy constructor
    DtSmoothPosWithDr(const DtSmoothPosWithDr& orig);

    // assignment operator
    DtSmoothPosWithDr& operator=(const DtSmoothPosWithDr& orig);

    // The default action is to dead-reckon
    virtual void dfltAction(DtTime simTime, void* usr,
        DtVectorRef vs, DtVectorRef ps);
};
```

smoothUtil.h

```

#include "mathTypes.h"
#include "vlTime.h"
#include "vlprint.h"
#include "chooseDefines.h"

// ----- POSITION -----

class DT_DLL_VL DtSmoothPos
{
public:

    // Constructors. smoothT is the initial smoothPeriod. Use a
    value of
    // -1 to indicate use of the default period.
    DtSmoothPos(void *usr, DtTime smoothT,
        DtConstVector pos, DtConstVector vel, DtConstVector acc);
    /* WARNING: not fully constructed until input() & tick() are
    called */
    DtSmoothPos(void *usr, DtTime smoothT);

    virtual ~DtSmoothPos() { };

    void input(DtConstVector pos, DtConstVector vel, DtConstVector
    acc,
        DtTime simTime, int freeze=0);
    void tick(DtTime simTime);

    DtConstVector position() const { return ps; }
    DtConstVector velocity() const { return vs; }

    // Set/Get the individual smoothing switch. In order for
    smoothing
    // to be enabled, the "master" smoothing-enabled switch must be
    on, AND
    // the individual smoothing switch (accessed through these func-
    tions),
    // must be on.
    virtual void setSmoothEnable(int on);
    virtual int smoothEnable() const;

    // Set/Get smoothPeriod. If set to -1, smooth period reverts to
    the
    // current default smooth period.
    virtual void setSmoothPeriod(DtTime t);
    virtual DtTime smoothPeriod() const;

public:

```

```
// Set/Get default smooth period
static void setDfltSmoothPeriod(DtTime t);
static DtTime dfltSmoothPeriod();

// Set/Get the value of the "master" smoothing-enabled switch.
static void setMasterSmoothEnable(DtBoolean onOff);
static DtBoolean masterSmoothEnable();
```

protected:

```
DtSmoothPos(const DtSmoothPos&);

virtual void dfltAction(DtTime simTime, void *usr,
    DtVectorRef vs, DtVectorRef ps) = 0;
void init(void *u, DtTime t);
void setT1data();
```

protected:

```
int frozen;

DtTime t0, t1, t2, lastT;

DtTime myLocalTau;
DtTime myLocalHtau;
DtTime myLocalTauInv;
DtTime* myTau;
DtTime* myHtau;
DtTime* myTauInv;

DtVector ps, vs, p0, v0, dp, dv, p1, v1, a1, a2, ap;
int myEnable;
int needT1data, needT2data;
void *usr;
```

protected:

```
static DtTime theDfltTau;
static DtTime theDfltHtau;
static DtTime theDfltTauInv;
static DtBoolean theMasterSmoothEnable;
};
// ----- ORIENTATION -----
```

```
class DT_DLL_VL DtSmoothOrient
{
public:
```

```

    // Constructors. smoothT is the initial smoothPeriod. Use a
    value of
    // -1 to indicate use of the default period.
    DtSmoothOrient(void *usr, DtTime smoothT,
        DtConstDcm orient, const DtDirectedRotRate &drr);
    /* WARNING: not fully constructed until input() & tick() are
    called */
    DtSmoothOrient(void *usr, DtTime smoothT);

    void input(DtConstDcm orient, const DtDirectedRotRate &drr,
        DtTime simTime, int freeze=0);
    void tick(DtTime simTime);

    DtConstDcm orientation() const { return os; }

    // Set/Get the individual smoothing switch. In order for
    smoothing
    // to be enabled, the "master" smoothing-enabled switch must be
    on, AND
    // the individual smoothing switch (accessed through these func-
    tions),
    // must be on.
    virtual void setSmoothEnable(int on);
    virtual int smoothEnable() const;

    // Set/Get smoothPeriod. If set to -1, smooth period reverts to
    the
    // current default smooth period.
    virtual void setSmoothPeriod(DtTime t);
    virtual DtTime smoothPeriod() const;

public:

    // Set/Get default smooth period
    static void setDfltSmoothPeriod(DtTime t);
    static DtTime dfltSmoothPeriod();

    // Set/Get the value of the "master" smoothing-enabled switch.
    static void setMasterSmoothEnable(DtBoolean onOff);
    static DtBoolean masterSmoothEnable();

protected:

    virtual void dfltAction(DtTime simTime, void *usr, DtDcmRef os) =
    0;
    void init(void *u, DtTime t);
    void setT1data();

protected:

```

```
    int frozen;
    DtTime t0, t2, lastT;
    DtTime* myTau;
    DtTime myLocalTau;
    DtDcm os, o0;
    DtVector ws;
    int myEnable;
    void *usr;

protected:

    static DtTime theDfltTau;
    static DtBoolean theMasterSmoothEnable;
};
```


smUtil.h

```

#ifdef !DtHLA

#include "hostStructs.h"
#include "ackPdu.h"

/* options for DtAcknowledgePredicate */
class DT_DLL_VL DtAckPredOpt {
public:
    DtAckPredOpt() :
        negAckRetVal(0),
        verboseToOther(0),
        verboseFromOther(0),
        verboseOtherType(0),
        verboseNegAck(1),
        verbosePosAck(0) { }

    DtEntityIdentifier to;
    DtEntityIdentifier from;
    DtAcknowledgeFlag ackFlag;
    int negAckRetVal;
    char verboseToOther;
    char verboseFromOther;
    char verboseOtherType;
    char verboseNegAck;
    char verbosePosAck;
};
/* for use with Connection::DtReadUntil */
int DtAcknowledgePredicate(DtPdu *pdu, void *arg);

/* Return 1 if to, from, and ackFlag match between pkt and arg and
 * positive response. Return negAckRetVal if match w/ neg Reponse.
 * Return 0 otherwise.
 * If (from.DtEntity == DtDeferredEntityId and hosts match), then
 * from.DtEntity will be set to from.DtEntity in PDU.
 */
int DtAcknowledgePredicate(DtAcknowledgePdu *pdu, DtAckPredOpt
*opt);

#endif

```

Socket.h

```
/* DtSocket is a misnomer
   This provides net-level Packet IO
   Sometimes, it is implemented with sockets, sometimes not */

#ifndef _Socket_h
#define _Socket_h

#include "vlconfig.h"

#if DT_HAVE_STD_H
#include <std.h>
#endif
#include <stdio.h>

#include <sys/types.h>

#include "vlutil.h"
#include "vllist.h"

/*
 * IP + UDP headers are 20 + 8.
 * Ethernet MTU is 1500 (or 1492 for 802.3 encapsulation),
 * thus PDU limits will typically be 1472 or 1464
 * FDDI MTU is 4352 (PDU limit 4324)
 * IEEE 802.5 token ring MTU is 4464 (PDU limit of 4436)
 */
#define DtMaxPacketSize 4440 /* multiple of 8 bytes */
#define DtDEFAULT_PORT 3000
#define DtUSE_DEFAULT_IP_ADDR (DtInetAddr(0))

#define DtSOCKET_PACKET 1
#define DtSOCKET_NO_PACKET 0
#define DtSOCKET_SERVER_MSG -1
#define DtSOCKET_FILTERED_PACKET -2

enum DtTestOperator { Dto_EQ, Dto_NEQ } ;

enum DtFilterType { DtFtInetAddr, DtFtNetU8 };

class DT_DLL_VLUTIL DtFilterTest
{
public:
    DtFilterTest() { disableTest();}
    virtual ~DtFilterTest() {}

    void disableTest() { setTest(Dto_NEQ,
0, NULL); }
};
```

```

    void setTest(DtTestOperator op, DtU32 value) { setTest(op, 1,
&value);    }
    void extendTest(DtU32 value)                { extendTest (1,
&value);    }
    void reduceTest(DtU32 value)                { reduceTest (1,
&value);    }
    virtual void setOperator(DtTestOperator op);
    virtual void setTest(DtTestOperator op, int n, DtU32 *values);
    virtual void extendTest(int n, DtU32 *values);
    virtual void reduceTest(int n, DtU32 *values);

    virtual DtBoolean evalTest(char *buf, int offset, DtFilterType
type);

protected:

    virtual void clearList();
    DtTestOperator testOp;
    DtList values;
};

```

```

class DT_DLL_VLUTIL DtSocket
{
public:

    DtSocket();

    virtual ~DtSocket() {}

    virtual int DtSend(caddr_t packet, size_t size,
DtInetAddr addr = DtUSE_DEFAULT_IP_ADDR);
    virtual int sendTo(caddr_t packet, size_t size,
DtInetAddr addr = DtUSE_DEFAULT_IP_ADDR) = 0;

    /* Depending on the implementation in derived DtSockets, one of
the
    * forms of DtRecv may be faster than the other.
    *
    * NOTE: if a derived class complains "... hides virtual DtRecv()
...",
    * make sure you have BOTH overloads of DtRecv declared and
defined
    * in the subclass.
    *
    * DtRecv has an optional status argument that could more
elegantly be

```

```
    * implemented as different return codes, but this would break
    older code.
```

```
    */
    virtual int DtRecv(caddr_t *buffptr, int *status=NULL) = 0;
    virtual int DtRecv(caddr_t buff, size_t) = 0;
    /* for those that don't care about length: */
    caddr_t DtRecv() { caddr_t bp; DtRecv(&bp); return bp; }

    int flush();
    void setBundling(int maxSize);
    int isBundling() {return maxBundleSize;}
    virtual void setUnbundling(int) {};

    virtual DtInetAddr lastSourceInetAddr() { return 0L; }
    virtual int DtReportPktsRcvd(void) { return (pkts_rcvd); }
    virtual void DtBecomeNonBlocking(void) {};
    virtual void notifyMe() {};

    virtual int listenToMulticastGroup(DtInetAddr) {return 0;}
    virtual int dropMulticastGroup(DtInetAddr) {return 0;}
    virtual int setMcastTtl(int) {return 0;}

    virtual DtFilterTest *exerciseId()      { return & version; }
    virtual DtFilterTest *protocolVersion() { return & exercise; }
    virtual DtFilterTest *pduKind()         { return & kind;     }
    virtual DtFilterTest *ipAddr()          { return & addr;      }

    int blocking;
    int pkts_rcvd;

    /*
    * not all DtSocket have associated file descriptors
    * those that do not will have readFd()/writeFd() return -1
    */
    virtual int readFd() { return -1; }
    virtual int writeFd() { return -1; }
    virtual int isOk()   { return 1; }
```

```
protected:
```

```
    virtual DtBoolean accept(char *buf, DtInetAddr iaddr) {
    return (protocolVersion()->evalTest(buf, 0, DtFtNetU8) &&
           exerciseId()->evalTest      (buf, 1, DtFtNetU8) &&
           pduKind()->evalTest          (buf, 2, DtFtNetU8) &&
           ipAddr()->evalTest            ((char *)&iaddr, 0, DtFtIne-
tAddr))?)

        DtTrue : DtFalse; }
    char outBuff[DtMaxPacketSize];
    int outSize;
```

```
    int bundling;
    int maxBundleSize;

private:
    DtFilterTest version;
    DtFilterTest exercise;
    DtFilterTest kind;
    DtFilterTest addr;
};

DT_DLL_VLUTIL DtSocket *DtNewDfltSocket(int port, DtInetAddr
destAddr = 0);

#endif
```

soldierDec.h

```
#if DtHLA

#include "milEntDec.h"

// class DtSoldierDecoder:
//
// Instances of DtSoldierDecoder are used to decode a
// soldier entity state message into a state repository in an HLA
// (as opposed to DIS) environment.

class DT_DLL_VL DtSoldierDecoder : public DtMilitaryEntityDecoder
{
public:

    // constructor
    DtSoldierDecoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtSoldierDecoder();

    // copy constructor
    DtSoldierDecoder(const DtSoldierDecoder& orig);

    // assignment operator
    DtSoldierDecoder& operator=(const DtSoldierDecoder& orig);

public:

    // Create a DtSoldierDecoder.
    // User is responsible for deletion.
    static DtHlaStateDecoder* create(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

protected:

    // Add decoding functions for the attributes.
    // Note: Non-virtual since meant to be called from constructors
    to
    //      add decoders.
    void addMyDecoders();

    // static decode<AttributeName> member functions.
    DtDECLARE_ENTITY_ATTR_DECODER(PrimaryWeaponState);
    DtDECLARE_ENTITY_ATTR_DECODER(SecondaryWeaponState);
}
```

```
};  
// These macros are used to define simple attribute decoder func-  
// tions  
// (members of a derived DtHlaStateDecoder that decode  
// individual attributes). The simple decoding is just to  
// convert the value pulled out of the ahvps to a host type using  
// the cast expression castExpr, and pass it to the mutator func-  
// tion.  
#define DtDEFINE_SIMPLE_SOLDIER_ATTR_DECODER_WITH_CAST( \  
    attrName, netType, mutator, castExpr) \  
    DtDEFINE_SIMPLE_ATTR_DECODER_WITH_CAST( \  
        DtSoldierDecoder, DtEntityStateRepository, \  
        attrName, netType, mutator, castExpr)  
  
#define DtDEFINE_SIMPLE_SOLDIER_ATTR_DECODER( \  
    attrName, netType, inspector) \  
    DtDEFINE_SIMPLE_SOLDIER_ATTR_DECODER_WITH_CAST( \  
        attrName, netType, inspector, (netType))  
  
#endif
```

soldierEnc.h

```
#if DtHLA

#include "milEntEnc.h"
#include "entitySR.h"

// class DtSoldierEncoder:
//
// Instances of DtSoldierEncoder are used to encode a soldier
// entity's
// state repository into a state message in an HLA environment.

class DT_DLL_VL DtSoldierEncoder : public DtMilitaryEntityEncoder
{
public:

    // constructor
    DtSoldierEncoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtSoldierEncoder();

    // copy constructor
    DtSoldierEncoder(const DtSoldierEncoder& orig);

    // assignment operator
    DtSoldierEncoder& operator=(const DtSoldierEncoder& orig);

public:

    // Create a DtSoldierEncoder.
    // Caller is responsible for deletion.
    static DtHlaStateEncoder* create(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

protected:

    // static need<Attribute> member functions.
    DtDECLARE_ENTITY_ATTR_CHECKER(PrimaryWeaponState);
    DtDECLARE_ENTITY_ATTR_CHECKER(SecondaryWeaponState);

    // static decode<Attribute> member functions.
    DtDECLARE_ENTITY_ATTR_ENCODER(PrimaryWeaponState);
    DtDECLARE_ENTITY_ATTR_ENCODER(SecondaryWeaponState);
};
```



```
// This macro is used to define simple attribute encoder functions
// (members of a derived DtHlaStateEncoder that encode
// individual attributes). The simple encoding is just to implicitly
// convert the value returned by the inspector to a netType,
// and add it to the ahvps.
#define DtDEFINE_SIMPLE_SOLDIER_ATTR_ENCODER( \
    attrName, netType, inspector) \
    DtDEFINE_SIMPLE_ATTR_ENCODER(DtSoldierEncoder, \
        DtEntityStateRepository, \
        attrName, netType, inspector)

// This macro is used to define simple attribute checker functions
// (members of a derived DtHlaStateEncoder that checker whether
// a particular attribute is out of date. The simple implementation
// is to just compare the values returned by the two StateReps'
// inspector functions.
#define DtDEFINE_SIMPLE_SOLDIER_ATTR_CHECKER( \
    attrName, inspector) \
    DtDEFINE_SIMPLE_ATTR_CHECKER(DtSoldierEncoder, \
        DtEntityStateRepository, \
        attrName, inspector)

#endif
```

startPdu.h

```
#if !DtHLA

#include "hostStructs.h"
#include "simanPdu.h"

#define DtStartStaticSize 44

class DT_DLL_VL DtStartResumePdu;
typedef DtStartResumePdu DtStartResumeInteraction;
typedef void (*DtStartResumePduCb)(DtStartResumePdu *pdu, void
*usr);

class DT_DLL_VL DtStartResumePdu : public DtSimanPdu
{
public:

/***** STANDARD CONSTRUCTORS *****/

    DtStartResumePdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtStartResumePdu(const DtNetStartResumePdu *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

/***** ADDITIONAL CONSTRUCTORS *****/

    DtStartResumePdu(const DtEntityIdentifier &to,
        const DtEntityIdentifier &from,
        const DtTime realTime,
        const DtTime simTime,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    static DtPdu *create(const DtNetPacket *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    static void addCallback(DtExerciseConn *conn,
        DtStartResumePduCb cb, void *usr);
    static void removeCallback(DtExerciseConn *conn,
        DtStartResumePduCb cb, void *usr);

/***** INSPECTOR FUNCTIONS *****/

    DtTime realWorldTime() const;
    DtTime simulationTime() const;
#if DtDISVERS != 3
    int requestId() const;
#endif

/***** MUTATOR FUNCTIONS *****/
```

```
    void setRealWorldTime(const DtTime time);
    void setSimulationTime(const DtTime time);
#if DtDISVERS != 3
    void setRequestId(int id);
#endif

    void printData() const;

protected:
    DtPduKind internalGetPduKind() const {return DtStartResumePdu-
Kind;}
};
#endif
```

startResRPdu.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "hostStructs.h"
#include "simanPdu.h"
#include "packets/startResRP.h"
#include "exerciseConn.h"

// class DtStartResumeRPdu:
//
// Instances of DtStartResumeRPdu are used to send/receive
// reliable start/resume PDUs.

typedef void (*DtStartResumeRPduCb)(DtStartResumeRPdu* pdu, void*
usr);

class DT_DLL_VL DtStartResumeRPdu : public DtSimanPdu
{
public:

    // Constructors
    DtStartResumeRPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtStartResumeRPdu(const DtNetStartResumeRPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtStartResumeRPdu(const DtEntityIdentifier& to,
        const DtEntityIdentifier& from,
        const DtTime realTime,
        const DtTime simTime,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtStartResumeRPdu(const DtStartResumeRPdu& orig);

    // destructor
    virtual ~DtStartResumeRPdu();

    // Assignment operator
    DtStartResumeRPdu& operator=(const DtStartResumeRPdu& orig);

    // Set/Get real world time.
    virtual void setRealWorldTime(const DtTime time);
    virtual DtTime realWorldTime() const;

    // Set/Get simulation time.
    virtual void setSimulationTime(const DtTime time);
    virtual DtTime simulationTime() const;

    // Set/Get the required reliability service.
```

```

    virtual void setReqRelService(DtRequiredReliabilityService
value);
    virtual DtRequiredReliabilityService reqRelService() const;

    // Set/Get the request id.
    virtual void setRequestId(int id);
    virtual int requestId() const;

    // Print the data contained by this PDU
    virtual void printData() const;

public:

    // Create a DtStartResumeRPdu.
    // Caller is responsible for deletion.
    static DtPdu* create(const DtNetPacket* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // Add (register)/Remove (deregister) function to be called when
    // pdu is received.
    static void addCallback(DtExerciseConn* conn,
        DtStartResumeRPduCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtStartResumeRPduCb cb, void* usr);

protected:

    // Returns the internal PDU kind
    virtual DtPduKind internalGetPduKind() const;

protected:

    static int theStaticSize;

};
#endif
#endif

```

stateRep.h

```
#include "MachineTypes.h"
#include "chooseDefines.h"
#include "hostStructs.h"

// class DtStateRepository:
//
// DtStateRepository is an abstract base class for maintaining state
// for an object. These are used to decode data into the repository
// from the network and encode data from the repository to the
// network.

class DT_DLL_VL DtStateRepository
{
public:

    // destructor
    virtual ~DtStateRepository();

    // assignment operator
    DtStateRepository& operator=(const DtStateRepository& orig);

    // return an empty object of the same type as this
    virtual DtStateRepository* clone() const = 0;

    // Print state data
    virtual void printData() const = 0;

#ifdef DtHLA

    // Get/Set global identifier, or name. Set by VR-Link for
    // reflected
    // objects. Not necessary to set for local objects.
    virtual void setGlobalId(const DtGlobalObjectDesignator& id);
    virtual const DtGlobalObjectDesignator& globalId() const;

    // Get/Set local identifier, or handle. Set by VR-Link for
    // reflected
    // objects. Not necessary to set for local objects.
    virtual void setLocalId(const DtObjectId& id);
    virtual const DtObjectId& localId() const;

#endif

protected:

    // Constructors are protected since this is an abstract base
    // class.
```

```
// default constructor
DtStateRepository();

// copy constructor
DtStateRepository(const DtStateRepository& orig);

#if DtHLA
    DtGlobalObjectDesignator myGlobalId;
    DtObjectId myLocalId;
#endif
};
```

stIntCbInfo.h

```
#if DtHLA

#include "stIntCbList.h"
#include "RTI.hh"
#include "stateMsg.h"

// class DtStartStopInterCbInfo:
//
// Instances of DtStartStopInterCbInfo are used for providing
// a callback when the RTI's start or stopInteractionGeneration
// services are invoked.
// This class is a helper class to DtStartStopInterCbList.

class DT_DLL_VL DtStartStopInterCbInfo
{
public:

    // constructor
    DtStartStopInterCbInfo(
        RTI::InteractionClassHandle hand,
        DtStartStopInterCb cb,
        void* usr);

    // destructor
    virtual ~DtStartStopInterCbInfo();

    // copy constructor
    DtStartStopInterCbInfo(
        const DtStartStopInterCbInfo& orig);

    // assignment operator
    DtStartStopInterCbInfo& operator=(
        const DtStartStopInterCbInfo& orig);

    // Check whether one callback info is equal to another.
    virtual int match(const DtStartStopInterCbInfo* const other)
const;

    // Check whether one callback info is equal to another.
    virtual int operator==(const DtStartStopInterCbInfo& other)
const;

    // Get the inter class handle.
    virtual const RTI::InteractionClassHandle& interClass() const;

    // Get the callback function.
    virtual const DtStartStopInterCb& cbFunction() const;
```



```
    // Get the user data.  
    virtual void* userData() const;  
  
protected:  
  
    RTI::InteractionClassHandle myClass;  
    DtStartStopInterCb myFunc;  
    void* myUsr;  
};  
#endif
```

stIntCbList.h

```
#if DtHLA

#include "vLList.h"
#include "RTI.hh"
#include "chooseDefines.h"

// class DtStartStopInterCbList:
//
// Instances of DtStartStopInterCbList are used to provide
// a list of callbacks (DtStartStopInterCbInfo) which are
// called when the RTI's start or stopInteractionGeneration
// services are invoked.

typedef void (*DtStartStopInterCb)(
    RTI::InteractionClassHandle intClass,
    void* usr);

class DT_DLL_VL DtStartStopInterCbList
{
public:

    // default constructor
    DtStartStopInterCbList();

    // destructor
    virtual ~DtStartStopInterCbList();

    // copy constructor
    DtStartStopInterCbList(
        const DtStartStopInterCbList& orig);

    // assignment operator
    DtStartStopInterCbList& operator=(
        const DtStartStopInterCbList& orig);

    // Add/Remove a start updates callback to/from the list.
    // Returns whether add/remove took place.
    virtual int add(RTI::InteractionClassHandle hand,
        DtStartStopInterCb cb,
        void* usr);
    virtual int remove(RTI::InteractionClassHandle hand,
        DtStartStopInterCb cb,
        void* usr);

    // Remove all callbacks from the list.
    virtual void clear();


```

```
// Callbacks are for a particular inter class handle.
// Call any callbacks in the list for the inter class handle
// for the start updates inter.
virtual void apply(RTI::InteractionClassHandle interClass);

protected:

    // Remove all callbacks from the list.
    // Non-virtual since meant to be called from dtor, otherwise
    // should call clear().
    void dtorClear();

    // Copy another callback list into this inter's callback list.
    // Non-virtual since meant to be called from ctor.
    void ctorCopyList(const DtStartStopInterCbList& orig);

protected:

    DtList myList;
};
#endif
```

stopFrzRPdu.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "disEnums.h"
#include "hostStructs.h"
#include "simanPdu.h"
#include "packets/stopFrzRP.h"
#include "exerciseConn.h"

// class DtStopFreezeRPdu:
//
// Instances of DtStopFreezeRPdu are used to send/receive
// reliable stop/freeze PDUs.

// DtStopFreezeRPdu callback function signature.
typedef void (*DtStopFreezeRPduCb)(DtStopFreezeRPdu* pdu, void*
usr);

class DT_DLL_VL DtStopFreezeRPdu : public DtSimanPdu
{
public:

    // Constructors
    DtStopFreezeRPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtStopFreezeRPdu(const DtNetStopFreezeRPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtStopFreezeRPdu(const DtEntityIdentifier& to,
        const DtEntityIdentifier& from,
        const DtTime realTime,
        DtStopReason reason,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtStopFreezeRPdu(const DtStopFreezeRPdu& orig);

    // destructor
    virtual ~DtStopFreezeRPdu();

    // Assignment operator
    DtStopFreezeRPdu& operator=(const DtStopFreezeRPdu& orig);

    // Set/Get real world time.
    virtual void setRealWorldTime(const DtTime time);
    virtual DtTime realWorldTime() const;

    // Set/Get the stop reason.
    virtual void setReason(DtStopReason reason);
    virtual DtStopReason reason() const;
```

```

    // Set/Get the frozen behavior.
    virtual void setFrozenBehavior(int behavior);
    virtual int frozenBehavior() const;

    // Set/Get the required reliability service.
    virtual void setReqRelService(DtRequiredReliabilityService
value);
    virtual DtRequiredReliabilityService reqRelService() const;

    // Set/Get the request id.
    virtual void setRequestId(int id);
    virtual int requestId() const;

    // Print the data contained by this PDU
    virtual void printData() const;

public:

    // Create a DtStopFreezeRPdu.
    // Caller is responsible for deletion.
    static DtPdu* create(const DtNetPacket* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // Add (register)/Remove (deregister) function to be called when
    // pdu is received.
    static void addCallback(DtExerciseConn* conn,
        DtStopFreezeRPduCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtStopFreezeRPduCb cb, void* usr);

protected:

    // Returns the internal PDU kind
    virtual DtPduKind internalGetPduKind() const;

protected:

    static int theStaticSize;

};
#endif
#endif

```

stopPdu.h

```
#if !DtHLA

#include "disEnums.h"
#include "hostStructs.h"
#include "simanPdu.h"

#define DtStopStaticSize 40

class DT_DLL_VL DtStopFreezePdu;
typedef DtStopFreezePdu DtStopFreezeInteraction;
typedef void (*DtStopFreezePduCb)(DtStopFreezePdu *pdu, void *usr);

class DT_DLL_VL DtStopFreezePdu : public DtSimanPdu
{
public:

/***** STANDARD CONSTRUCTORS *****/

    DtStopFreezePdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtStopFreezePdu(const DtNetStopFreezePdu *initial,
                    DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

/***** ADDITIONAL CONSTRUCTORS *****/

    DtStopFreezePdu(const DtEntityIdentifier &to,
                    const DtEntityIdentifier &from,
                    const DtTime realTime,
                    DtStopReason reason,
                    DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    static DtPdu *create(const DtNetPacket *initial,
                        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    static void addCallback(DtExerciseConn *conn,
                           DtStopFreezePduCb cb, void *usr);
    static void removeCallback(DtExerciseConn *conn,
                               DtStopFreezePduCb cb, void *usr);

/***** INSPECTOR FUNCTIONS *****/

    DtTime realWorldTime() const;
    DtStopReason reason() const;
#if DtDISVERS != 3
    int frozenBehavior() const;
    int requestId() const;
#endif

/***** MUTATOR FUNCTIONS *****/
```

```
    void setRealWorldTime(const DtTime time);
    void setReason(DtStopReason reason);
#if DtDISVERS != 3
    void setFrozenBehavior(int behavior);
    void setRequestId(int id);
#endif

    void printData() const;

protected:
    DtPduKind internalGetPduKind() const {return DtStopFreezePdu-
Kind;}
};
#endif
```

str.h

```
#include "MachineTypes.h"
#include "iostream.h"

// class DtString:
//
// Instances of DtString are

class DT_DLL_VLUTIL DtString
{
    friend DT_DLL_VLUTIL ostream& operator<<(ostream& os, const
DtString& str);
    friend DT_DLL_VLUTIL istream& operator>>(istream& is, DtString&
str);

public:

    // default constructor
    DtString();

    DtString(const char* str);

    // destructor
    virtual ~DtString();

    // copy constructor
    DtString(const DtString& orig);

    // assignment operators
    DtString& operator=(const DtString& orig);
    DtString& operator=(const char* orig);

    // cast operator
    virtual operator const char*() const;

    // comparison operators
    friend DT_DLL_VLUTIL DtBoolean operator==(const DtString& str1,
const DtString& str2);
    friend DT_DLL_VLUTIL DtBoolean operator<(const DtString& str1,
const DtString& str2);
    friend DT_DLL_VLUTIL DtBoolean operator<=(const DtString& str1,
const DtString& str2);
    friend DT_DLL_VLUTIL DtBoolean operator>(const DtString& str1,
const DtString& str2);
    friend DT_DLL_VLUTIL DtBoolean operator>=(const DtString& str1,
const DtString& str2);
    friend DT_DLL_VLUTIL DtBoolean operator!=(const DtString& str1,
const DtString& str2);
```

```

virtual DtBoolean operator==(const char* str) const;
virtual DtBoolean operator<(const char* str) const;
virtual DtBoolean operator<=(const char* str) const;
virtual DtBoolean operator>(const char* str) const;
virtual DtBoolean operator>=(const char* str) const;
virtual DtBoolean operator!=(const char* str) const;

// Case-insensitive string compare. Returns DtTrue if the
strings are
// equal except for differences in case.
virtual DtBoolean caseCompare(const DtString& str);

// concatenation operators
friend DT_DLL_VLUTIL DtString operator+(const DtString& str1,
const DtString& str2);
virtual DtString operator+(const char* str) const;
virtual DtString& operator+=(const char* str);
virtual DtString& operator+=(const DtString& str);

// index operator
virtual const char& operator[](int index) const;
virtual char& operator[](int index);

// truncates everything before or after the character, including
the
// character, respectively
// Normally, snipFront stops at the first occurrence of "ch", and
snipBack
// stops at the last occurrence of "ch". The second argument
flags reverse
// this. Return value is DtTrue iff "ch" was found in the string.
virtual DtBoolean snipFront(const char ch, DtBoolean toLastCh =
DtFalse);
virtual DtBoolean snipBack(const char ch, DtBoolean fromFirstCh =
DtFalse);

DtBoolean snipFrontToLast(const char ch) { return snipFront(ch,
DtTrue); }
DtBoolean snipBackToFirst(const char ch) { return snipBack(ch,
DtTrue); }

// returns the number of occurrences of the character
virtual int count(const char ch) const;

// return the length of the string
virtual int length() const;

// returns the char* internal representation of the string
const char* string() const;

```

```
    static const DtString& nullString();
protected:

    // deletes the string, and sets the myString pointer to NULL
    // non-virtual because only called by destructor, and non-virtual
members
    // myString is set to NULL, so it should be allocated some memory
if
    // operators are to be used on the string
    void clear();

    // Set/Get for myString - char pointer that holds the string
    // non-virtual because only called by constructor
    void setString(const char* str);

protected:

    char* myString;

    static const int theMaxInsertionStringSize;
};
```

strHashKey.h

```
#include "baseHashKey.h"

// class DtStringHashKey:
//
// Instances of DtStringHashKey are used as a hash key made from a
// string.

class DT_DLL_VLUTIL DtStringHashKey : public DtBaseHashKey
{
public:

    // default constructor
    DtStringHashKey(const DtString& str);

    // alternate constructor
    DtStringHashKey(const char* str);

    // destructor
    virtual ~DtStringHashKey();

    // copy constructor
    DtStringHashKey(const DtStringHashKey& orig);

    // assignment operator
    DtStringHashKey& operator=(const DtStringHashKey& orig);

    // make a copy of this hash key
    virtual DtBaseHashKey* clone() const;

protected:

    // get the string representation of this key
    virtual const DtString& stringRep() const;

    // compute the hashed value for this key
    virtual DtU32 computeHashValue();

protected:

    DtString myString;
};
```

strtUpdCbInfo.h

```
#if DtHLA

#if DtRTI103

#include "strtUpdCbList.h"
#include "RTI.hh"
#include "stateMsg.h"

// class DtUpdateControlCbInfo:
//
// Instances of DtUpdateControlCbInfo are used for providing
// a callback when the RTI update control services are invoked.
// This class is a helper class to DtUpdateControlCbList.

class DT_DLL_VL DtUpdateControlCbInfo
{
public:

    // constructor
    DtUpdateControlCbInfo(RTI::ObjectClassHandle h,
        DtUpdateControlCb cb,
        void* usr);

    // destructor
    virtual ~DtUpdateControlCbInfo();

    // copy constructor
    DtUpdateControlCbInfo(
        const DtUpdateControlCbInfo& orig);

    // assignment operator
    DtUpdateControlCbInfo& operator=(
        const DtUpdateControlCbInfo& orig);

    // Check whether one callback info is equal to another.
    virtual int match(const DtUpdateControlCbInfo* const other)
const;

    // Check whether one callback info is equal to another.
    virtual int operator==(const DtUpdateControlCbInfo& other) const;

    // Get the object class handle.
    virtual const RTI::ObjectClassHandle& objectClass() const;

    // Get the callback function.
    virtual const DtUpdateControlCb& cbFunction() const;
```

```
// Get the user data.
virtual void* userData() const;

protected:

    RTI::ObjectClassHandle myClass;
    DtUpdateControlCb myFunc;
    void* myUsr;
};
typedef DtUpdateControlCbInfo DtStartUpdatesCbInfo;
typedef DtUpdateControlCbInfo DtStopUpdatesCbInfo;

#endif
#endif
```

strtUpdCbList.h

```
#if DtHLA

#if DtRTI103

#include "vllist.h"
#include "RTI.hh"

// class DtUpdateControlCbList:
//
// Instances of DtUpdateControlCbList are used to provide
// a list of callbacks (DtUpdateControlCbInfo) which are
// called when an object starts updating.

typedef void (*DtUpdateControlCb)(
    RTI::ObjectClassHandle objectClass,
    const RTI::AttributeHandleSet& attributeSet,
    void* usr);

typedef DtUpdateControlCb DtStartUpdatesCb;
typedef DtUpdateControlCb DtStopUpdatesCb;

class DT_DLL_VL DtUpdateControlCbList
{
public:

    // default constructor
    DtUpdateControlCbList();

    // destructor
    virtual ~DtUpdateControlCbList();

    // copy constructor
    DtUpdateControlCbList(
        const DtUpdateControlCbList& orig);

    // assignment operator
    DtUpdateControlCbList& operator=(
        const DtUpdateControlCbList& orig);

    // Add/Remove a start updates callback to/from the list.
    // Returns whether add/remove took place.
    virtual int add(RTI::ObjectClassHandle h, DtUpdateControlCb cb,
        void* usr);
    virtual int remove(RTI::ObjectClassHandle h, DtUpdateControlCb
cb,
        void* usr);
```

```
// Remove all callbacks from the list.
virtual void clear();

// Callbacks are for a particular object class handle.
// Call any callbacks in the list for the object class handle.
virtual void apply(RTI::ObjectClassHandle objectClass,
    const RTI::AttributeHandleSet& hSet);
```

protected:

```
// Remove all callbacks from the list.
// Non-virtual since meant to be called from dtor, otherwise
// should call clear().
void dtorClear();

// Copy another callback list into this object's callback list.
// Non-virtual since meant to be called from ctor.
void ctorCopyList(const DtUpdateControlCbList& orig);
```

protected:

```
    DtList myList;
};
typedef DtUpdateControlCbList DtStartUpdatesCbList;
typedef DtUpdateControlCbList DtStopUpdatesCbList;

#endif
#endif
```

supEmisEsNozz.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"

#include "packets/supEmisEsP.h"

// class DtVectoringNozzleSystem:
//
// DtVectoringNozzleSystem is a class which contains the vectoring
// nozzle
// system data for the supplemental emissions/entity state PDU.

class DT_DLL_VL DtVectoringNozzleSystem
{
public:

    // Constructors
    DtVectoringNozzleSystem();
    DtVectoringNozzleSystem(const DtVectoringNozzleSystem& orig);

    // Assignment operator
    DtVectoringNozzleSystem& operator=(
        const DtVectoringNozzleSystem& orig);
    DtVectoringNozzleSystem& operator=(
        const DtNetVectoringNozzleSystem& orig);

    // Destructor
    virtual ~DtVectoringNozzleSystem();

    // Set/Get the horizontal deflection
    virtual void setHorDeflection(DtFloat32 value);
    virtual DtFloat32 horDeflection() const;

    // Set/Get the vertical deflection
    virtual void setVertDeflection(DtFloat32 value);
    virtual DtFloat32 vertDeflection() const;

    // Print the information contained in the record
    virtual void print() const;

    // Load/Unload data from a packet record
    virtual void unloadNetRecord(
        const DtNetVectoringNozzleSystem& record);
    virtual void loadNetRecord(
```



```
        DtNetVectoringNozzleSystem& record) const;

protected:

    // Data members:
    DtFloat32                myHorizontalDeflection;
    DtFloat32                myVerticalDeflection;

};
#endif
#endif
```

supEmisEsPdu.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "hostStructs.h"
#include "pdu.h"
#include "mathTypes.h"

#include "packets/supEmisEsP.h"

#include "supEmisEsNozz.h"
#include "supEmisEsProp.h"

// class DtSuppEmisEntStatePdu:
//
// Instances of DtSuppEmisEntStatePdu are used to send information
// about
// simulated supplemental emissions.

// DtSuppEmisEntStatePdu callback function signature.
typedef void (*DtSuppEmisEntStatePduCb)(DtSuppEmisEntStatePdu* pdu,
void* usr);

class DT_DLL_VL DtSuppEmisEntStatePdu : public DtPdu
{
public:

    // Constructors
    DtSuppEmisEntStatePdu(DtBufferPtr buffer =
DtUSE_INTERNAL_BUFFER);
    DtSuppEmisEntStatePdu(const DtNetSuppEmisEntStatePdu* initial,
DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtSuppEmisEntStatePdu(const DtSuppEmisEntStatePdu& orig);

    // destructor
    virtual ~DtSuppEmisEntStatePdu();

    // Assignment operator
    DtSuppEmisEntStatePdu& operator=(const DtSuppEmisEntStatePdu&
orig);

    // Set/Get originating entity identifier
    virtual void setOriginatingEntityID(const DtEntityIdentifier&
entity);
    virtual const DtEntityIdentifier& originatingEntityID() const;
```

```

// Set/Get infra-red signature representation index
virtual void setInfraRedSigRepIndex(DtU16 value);
virtual DtU16 infraRedSigRepIndex() const;

// Set/Get acoustic signature representation index
virtual void setAcousticSigRepIndex(DtU16 value);
virtual DtU16 acousticSigRepIndex() const;

// Set/Get radar signature representation index
virtual void setRadarSigRepIndex(DtU16 value);
virtual DtU16 radarSigRepIndex() const;

// Set/Get number of propulsion systems
virtual DtBoolean setNumPropulsionSystems(DtU16 value);
virtual DtU16 numPropulsionSystems() const;

// Set/Get number of nozzle systems
virtual DtBoolean setNumNozzleSystems(DtU16 value);
virtual DtU16 numNozzleSystems() const;

// Set/Get a propulsion system data record.
// Returns false if the record does not exist.
virtual DtBoolean setPropulsionSystem(int index,
    const DtPropulsionSystem& record);
virtual DtBoolean getPropulsionSystem(int index,
    DtPropulsionSystem& record) const;

// Set/Get a vectoring nozzle system data record.
// Returns false if the record does not exist.
virtual DtBoolean setNozzleSystem(int index,
    const DtVectoringNozzleSystem& record);
virtual DtBoolean getNozzleSystem(int index,
    DtVectoringNozzleSystem& record) const;

// Print the data contained by this PDU
virtual void printData() const;

```

public:

```

// Create a DtSuppEmisEntStatePdu.
// Caller is responsible for deletion.
static DtPdu* create(const DtNetPacket* initial,
    DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

// Add (register)/Remove (deregister) function to be called when
// pdu is received.
static void addCallback(DtExerciseConn* conn,
    DtSuppEmisEntStatePduCb cb, void* usr);
static void removeCallback(DtExerciseConn* conn,
    DtSuppEmisEntStatePduCb cb, void* usr);

```

protected:

```
// Returns the internal PDU kind
virtual DtPduKind internalGetPduKind() const;
```

```
// Casts the netPdu element to our network struct
virtual DtNetSuppEmisEntStatePdu* netPduStruct() const;
```

```
// Casts the 'this' pointer for changing elements in a const
function
```

```
// Non virtual since you can't override based on return type
DtSuppEmisEntStatePdu* self() const;
```

```
// Returns a pointer to the first byte of variable length data
virtual char* var() const;
```

```
// Returns a buffer offset to the propulsion system data record
virtual int propulsionSystemRecordOffset(int recordIndex) const;
```

```
// Returns a buffer offset to the given nozzle system data record
virtual int nozzleSystemRecordOffset(int recordIndex) const;
```

protected:

```
// Size of PDU with zero parameter records.
static int theStaticSize;
```

protected:

```
// Temporary objects for reference returns
DtEntityIdentifier          myEntityID;
```

```
};
#endif
#endif
```

supEmisEsProp.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"

#include "packets/supEmisEsP.h"

// class DtPropulsionSystem:
//
// DtPropulsionSystem is a class which contains the propulsion
// system data
// for the supplemental emissions/entity state PDU.

class DT_DLL_VL DtPropulsionSystem
{
public:

    // Constructors
    DtPropulsionSystem();
    DtPropulsionSystem(const DtPropulsionSystem& orig);

    // Assignment operator
    DtPropulsionSystem& operator=(
        const DtPropulsionSystem& orig);
    DtPropulsionSystem& operator=(
        const DtNetPropulsionSystem& orig);

    // Destructor
    virtual ~DtPropulsionSystem();

    // Set/Get the power setting
    virtual void setPowerSetting(DtFloat32 value);
    virtual DtFloat32 powerSetting() const;

    // Set/Get the engine RPM
    virtual void setEngineRPM(DtFloat32 value);
    virtual DtFloat32 engineRPM() const;

    // Print the information contained in the record
    virtual void print() const;

    // Load/Unload data from a packet record
    virtual void unloadNetRecord(
        const DtNetPropulsionSystem& record);
    virtual void loadNetRecord(

```

```
        DtNetPropulsionSystem& record) const;

protected:

    // Data members:
    DtFloat32          myPowerSetting;
    DtFloat32          myEngineRPM;

};
#endif
#endif
```

sysHashKey.h

```

#include "globalObjDes.h"

#if DtHLA
#include "strHashKey.h"
#else
#include "objIdHashKey.h"
#endif

// class DtSystemHashKey:
//
// Instances of DtSystemHashKey are used as hash keys for an emitter
// system.

class DT_DLL_VL DtSystemHashKey : public DtBaseHashKey
{
public:
    // default constructor
    DtSystemHashKey(const DtGlobalObjectDesignator& id, int
systemId);

    // destructor
    virtual ~DtSystemHashKey();

    // copy constructor
    DtSystemHashKey(const DtSystemHashKey& orig);

    // assignment operator
    DtSystemHashKey& operator=(const DtSystemHashKey& orig);

    // make a copy of this hash key
    virtual DtBaseHashKey* clone() const;

protected:
    // get the string representation of this key
    virtual const DtString& stringRep() const;

    // compute the hashed value for this key
    virtual DtU32 computeHashValue();

protected:
#if DtHLA
    DtStringHashKey myDesHashKey;
#else

```

```
    DtObjectIdHashKey myDesHashKey;  
#endif  
  
    DtString myStr;  
    int mySystemId;  
  
};
```


taitBryan.h

```
#include "MachineTypes.h"

class DT_DLL_MATRIX DtTaitBryan
{
public:
    DtTaitBryan();
    DtTaitBryan(double a, double b, double c);

    DtTaitBryan(const DtTaitBryan &tait);
    DtTaitBryan &operator=(const DtTaitBryan &tait);

    // Get whether another tait bryan equals or does not equal this
    one.
    int operator==(const DtTaitBryan& other) const;
    int operator!=(const DtTaitBryan& other) const;

    const char *string() const;

    DtFloat64 psi() const;
    DtFloat64 theta() const;
    DtFloat64 phi() const;

    void setPsi(DtFloat64 psi);
    void setTheta(DtFloat64 theta);
    void setPhi(DtFloat64 phi);

protected:

    DtFloat64 myPsi; /* about reference z */
    DtFloat64 myTheta; /* about intermediate y */
    DtFloat64 myPhi; /* about body x */
};
// Inline functions
```

tcpForwarder.h

```
#include "MachineTypes.h"
#include "NetTypes.h"
#include "Socket.h"
#include "vlutil.h"
#include "tcpSocketMgr.h"

class DT_DLL_VLUTIL DtTcpForwarder
{
public:
    DtTcpForwarder(int port=DtDEFAULT_PORT);

    virtual ~DtTcpForwarder();

    void sendToAll(caddr_t packet, size_t size);

    void readAndForward(DtTime maxTimeOverAll, DtTime maxTimePer-
Socket);

    DtTcpSocketMgr* socketMgr();

private:
    DtTcpSocketMgr* mySocketMgr;
    DtList* mySockList;
    DtTcpSocket* myBeginSocket;
    DtListItem* myBeginItem;
};
```

tcpSocket.h

```

#include <stdio.h>
#include <sys/types.h>
#ifdef !WIN32
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#endif
#include "MachineTypes.h"      /* needed for <sys/types.h> or relatives */
#include "NetTypes.h"
#include "Socket.h"
#include "vlutil.h"

class DT_DLL_VLUTIL DtTcpSocket : public DtSocket
{
public:

    // Server Constructor
    DtTcpSocket(int sockfd, struct sockaddr_in addr);

    // Client Constructor
    DtTcpSocket(DtInetAddr addr, int port=DtDEFAULT_PORT, int
        nonBlockingConnect=0);

    // Destructor
    virtual ~DtTcpSocket();

    int sendTo(caddr_t packet, size_t packetSize,
        DtInetAddr addr=DtUSE_DEFAULT_IP_ADDR);

    int DtRecv(caddr_t *buffptr, int *status=NULL);

    int DtRecv(caddr_t buff, size_t bufsize);

    int readFd();

    int writeFd();

    // designate the socket as non-blocking
    void DtBecomeNonBlocking(void);

    int isOk()    { return myOk; }

    // Determines if the tcp socket has connected successfully
    int connectionEstablished();

```

```
// Get/Set mySizeOffset
int sizeOffset() { return mySizeOffset; }
void setSizeOffset(int offset);

// Get/Set mySizeLength
int sizeLength() { return mySizeLength; }
void setSizeLength(int length);

private:
    int recvWithStatus(caddr_t *buffptr, int *status);

    char myPktBuff[2000];
#ifdef WIN32
    SOCKET mySockfd;
#else
    int mySockfd; // socket number file descriptor
#endif
    struct sockaddr_in myConnectAddr;
    int myOk;
    int mySizeOffset;
    int mySizeLength;
    static DtBoolean theAnyInstancesFlag;
};
```

tcpSocketMgr.h

```
#include "MachineTypes.h"
#include "NetTypes.h"
#include "Socket.h"
#include "vlutil.h"
#include "tcpSocket.h"

class DT_DLL_VLUTIL DtTcpSocketMgr
{
public:

    DtTcpSocketMgr(int port=DtDEFAULT_PORT);

    virtual ~DtTcpSocketMgr();

    void processConnections();

    int isOk()    { return myOk; }

    DtList* socketList();

private:
    int serverSockfd;
    struct sockaddr_in myServAddr;
    struct sockaddr_in myCliAddr;
    DtList* mySockList;
    int myOk;
};
```

thresholder.h

```
#include "MachineTypes.h"
#include "mathTypes.h"
#include "artPartThresh.h"

// class DtThresholder:
//
// Instances of DtThresholder store thresholding information used to
// determine what attributes of an entity need to be updated.

class DT_DLL_VL DtThresholder
{
public:

    // default constructor
    DtThresholder();

    // destructor
    virtual ~DtThresholder();

    // copy constructor
    DtThresholder(const DtThresholder& orig);

    // assignment operator
    DtThresholder& operator=(const DtThresholder& orig);

    // Set/Get rotation threshold in radians.
    // Negative number means use default.
    virtual void setRotationThreshold(DtReal thresh);
    virtual double rotationThreshold() const;

    // Returns the "RotationTrace" threshold
    virtual double rotationTraceThreshold() const;

    // Set/Get translation threshold in meters.
    // Negative number means use default.
    virtual void setTranslationThreshold(DtReal thresh);
    virtual double translationThreshold() const;

    // Set/Get aggregate dimension threshold. Thresh is a number
    // between
    // 0 and 1, indicating the percentage change that an aggregate
    // dimension
    // can undergo before causing an update to be generated.
    // Negative number means use default.
    // Applies only to aggregate entities.
    virtual void setAggDimensionThreshold(DtReal thresh);
    virtual double aggDimensionThreshold() const;
```

```
// Returns a pointer to an object that maintains thresholds
// for articulated parts.
virtual DtArtPartThresholder* artPartThresholder();
virtual const DtArtPartThresholder& artPartThresholder() const;

public:

    // Set/Get default translation threshold in radians.
    static void setDfltTranslationThreshold(DtReal thresh);
    static double dfltTranslationThreshold();

    // Set/Get default rotation threshold in radians.
    static void setDfltRotationThreshold(DtReal thresh);
    static double dfltRotationThreshold();

    // Returns the "RotationTrace" threshold
    static double dfltRotationTraceThreshold();

    // Set/Get aggregate dimension threshold. Thresh is a number
    // between
    // 0 and 1, indicating the percentage change that an aggregate
    // dimension
    // can undergo before causing an update to be generated.
    // Applies only to aggregate entities.
    static void setDfltAggDimensionThreshold(DtReal thresh);
    static double dfltAggDimensionThreshold();

protected:

    double myTransThresh;
    double myRotThresh;
    double myTraceRotThresh;
    double myAggDimThresh;
    DtBoolean myUsingDfltTrans;
    DtBoolean myUsingDfltRot;
    DtBoolean myUsingDfltAggDim;

    DtArtPartThresholder myArtPartThresh;

protected:

    static double theDfltTransThresh;
    static double theDfltRotThresh;
    static double theDfltTraceRotThresh;
    static double theDfltAggDimThresh;
};
```

tkJmDesLst.h

```
#if !DtHLA

// class DtTrackJamDesignatorList:
//
// Instances of DtTrackJamDesignatorList are list of globally
// significant object identifiers.

#include "hostStructs.h"
#include "trackJamDes.h"

class DT_DLL_VL DtTrackJamDesignatorList
{
public:

    // default constructor
    DtTrackJamDesignatorList();

    // destructor
    virtual ~DtTrackJamDesignatorList();

    // copy constructor
    DtTrackJamDesignatorList(const DtTrackJamDesignatorList& orig);

    // assignment operator
    DtTrackJamDesignatorList& operator=(
        const DtTrackJamDesignatorList& orig);

    // equivalence operators
    int operator==(const DtTrackJamDesignatorList& other) const;
    int operator!=(const DtTrackJamDesignatorList& other) const;

    // Set/Get number of objects. Setting to a number lower than
    // the current number will remove trailing objects from the list.
    virtual void setNumObjects(int num);
    virtual int numObjects() const;

    // Set index'th object designator.
    virtual DtBoolean setObject(int index,
        const DtTrackJamDesignator& des);

    // Get index'th object designator. Set valid to DtFalse if bad
    index,
    // DtTrue otherwise.
    virtual const DtTrackJamDesignator& object(
        int index, DtBoolean* valid = NULL) const;

};
```



```
// Print data.  
virtual void print() const;  
  
protected:  
  
    int myNum;  
    DtTrackJamDesignator* myObjs;  
};  
#endif
```

topoCoord.h

```
#include "geodCoord.h"
#include "LibMatrix.h"

// Any point on the globe can be used to specify the origin of a
// specific topographic coordinate system. In VR-Link, a topo-
// graphic
// frame is defined with the positive X axis pointing north, the
// positive Y axis pointing east, and the positive Z axis pointing
// down.

// From latitude and longitude, we can produce the rotation matrices
// that rotate between geocentric and topographic-down orientations.
// VR-Link uses post-multiplication for its rotation matrices. For
// example, the matrix produced by DtLatLon_to_TopoToGeoc can be
// used
// to rotate a topographic vector to a geocentric vector like this:
//      DtDcmVecMul(topoToGeoc, Vtopo, Vgeoc);
DT_DLL_VL void DtLatLon_to_TopoToGeoc(const DtGeodeticCoord& g,
DtDcmRef topoToGeoc);

// The matrix produced by DtLatLon_to_GeocToTopo can be used
// to rotate a geocentric vector to a topographic vector like this:
//      DtDcmVecMul(geocToTopo, Vgeoc, Vtopo);
DT_DLL_VL void DtLatLon_to_GeocToTopo(const DtGeodeticCoord& g,
DtDcmRef geocToTopo);

// This function can initialize a DtCoordTransform so that it can
// provide
// transformations between the geocentric coordinate system, and a
// particular topographic coordinate system defined by latitude and
// longitude.
DT_DLL_VL void DtGeocToTopoTransform(double latitude, double longi-
tude,
DtCoordTransform* geoc2topo);
```

topoView.h

```
#include "cartView.h"

// class DtTopoView:
//
// Instances of DtTopoView are used to create a Topographic
// coordinate view of the geocentric coordinates in an
// Entity State Repository.

class DT_DLL_VL DtTopoView : public DtCartesianView
{
public:

    // constructor
    DtTopoView(DtBaseEntityStateRepository* esr,
               double latitude, double longitude);

    // destructor
    virtual ~DtTopoView();

    // copy constructor
    DtTopoView(const DtTopoView& orig);

    // assignment operator
    DtTopoView& operator=(const DtTopoView& orig);
};
```

trackJamDes.h

```
#if !DtHLA

#include "globalObjDes.h"
#include "str.h"
#include "hostStructs.h"

// class DtTrackJamDesignator:
//
// Instances of DtTrackJamDesignator are

class DT_DLL_VL DtTrackJamDesignator : public DtGlobalObjectDesignator
{
public:

    // default constructor
    DtTrackJamDesignator();

    // default constructor
    DtTrackJamDesignator(
        const DtEntityIdentifier& entId,
        int emitterId,
        int beamId);

    // destructor
    virtual ~DtTrackJamDesignator();

    // copy constructor
    DtTrackJamDesignator(const DtTrackJamDesignator& orig);

    // assignment operator
    DtTrackJamDesignator& operator=(const DtTrackJamDesignator&
orig);

    // set/get the emitter Id
    virtual void setEmitterId(int id);
    virtual int emitterId() const;

    // set/get the beam Id
    virtual void setBeamId(int id);
    virtual int beamId() const;

    // comparison operators
    virtual DtBoolean operator==(const DtTrackJamDesignator& des)
const;
```

```
    virtual DtBoolean operator!=(const DtTrackJamDesignator& des)
const;

    // returns a string representation
    virtual const char* string() const;

protected:

    int myEmitterId;
    int myBeamId;
    DtString myStringRep;
};
#endif
```

transCtlPdu.h

```
#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "hostStructs.h"
#include "recordSetPdu.h"
#include "exerciseConn.h"

#include "packets/transCtlP.h"

// class DtTransferControlRequestPdu:
//
// Instances of DtTransferControlRequestPdu are used to initiate the
dynamic
// allocation and control of simulated entities.

// DtTransferControlRequestPdu callback function signature.
typedef void (*DtTransferControlRequestPduCb)
    (DtTransferControlRequestPdu* pdu, void* usr);

class DT_DLL_VL DtTransferControlRequestPdu : public DtRecordSetPdu
{
public:

    // Constructors
    DtTransferControlRequestPdu(DtBufferPtr buffer =
DtUSE_INTERNAL_BUFFER);
    DtTransferControlRequestPdu(const DtNetTransferControlRe-
questPdu* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtTransferControlRequestPdu(const DtTransferControlRequestPdu&
orig);

    // destructor
    virtual ~DtTransferControlRequestPdu();

    // Assignment operator
    DtTransferControlRequestPdu& operator=(
        const DtTransferControlRequestPdu& orig);

    // Set/Get originating entity identifier
    virtual void setOriginatingEntityID(const DtEntityIdentifier&
entity);
    virtual const DtEntityIdentifier& originatingEntityID() const;

    // Set/Get receiving entity identifier
```

```

    virtual void setReceivingEntityID(const DtEntityIdentifier&
entity);
    virtual const DtEntityIdentifier& receivingEntityID() const;

    // Set/Get request ID
    virtual void setRequestID(DtU32 id);
    virtual DtU32 requestID() const;

    // Set/Get required reliability service
    virtual void setReqRelService(enum DtRequiredReliabilityService
service);
    virtual DtRequiredReliabilityService reqRelService() const;

    // Set/Get transfer type
    virtual void setTransferType(DtTransferType type);
    virtual DtTransferType transferType() const;

    // Set/Get transferred entity identifier
    virtual void setTransferredEntityID(const DtEntityIdentifier&
entity);
    virtual const DtEntityIdentifier& transferredEntityID() const;

    // Print the data contained by this PDU
    virtual void printData() const;

public:

    // Create a DtTransferControlRequestPdu.
    // Caller is responsible for deletion.
    static DtPdu* create(const DtNetPacket* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // Add (register)/Remove (deregister) function to be called when
    // pdu is received.
    static void addCallback(DtExerciseConn* conn,
        DtTransferControlRequestPduCb cb, void* usr);
    static void removeCallback(DtExerciseConn* conn,
        DtTransferControlRequestPduCb cb, void* usr);

protected:

    // Returns a pointer to the beginning of the record set data
    struct.
    virtual DtNetRecordSetPdu* netRecordSetPduStruct() const;

    // Returns the internal PDU kind
    virtual DtPduKind internalGetPduKind() const;

    // Casts the netPdu element to our network struct
    // Non-virtual since you can't override based on return type

```

```
virtual DtNetTransferControlRequestPdu* netPduStruct() const;

// Casts the 'this' pointer for changing elements in a const
function
// Non virtual since you can't override based on return type
DtTransferControlRequestPdu* self() const;

protected:

// Size of PDU with zero data bytes.
static int theStaticSize;

protected:

// Temporary objects for reference returns
DtEntityIdentifier myEntityID;

};
#endif
#endif
```


transmitPdu.h

```

#if !DtHLA

#include "hostStructs.h"
#include "pdu.h"
#include "mathTypes.h"
#include "packets/transmitterP.h"
#include "beamAntRec.h"

#define DtTransmitterStaticSize 104

class DT_DLL_VL DtTransmitterPdu;
typedef void (*DtTransmitterPduCb)(DtTransmitterPdu *pdu, void
*usr);

#define DtFREQ_UPPER 0
#define DtFREQ_LOWER 1

class DT_DLL_VL DtModulationType
{
public:
    DtModulationType(int spectrum = 0,
        DtRadioMajorModulation major =
        DtRadioMajorModOther,
        int detail = 0,
        DtRadioSystem sys = DtRadioSystemOther)
    : Spectrum(spectrum), Major(major), Detail(detail), Sys(sys) {}

    unsigned int spectrum() const           {return Spectrum;}
    DtRadioMajorModulation majorModType() const {return Major;}
    int detail() const                       {return Detail;}
    DtRadioSystem systemType() const         {return Sys;}

    void setSpectrum(unsigned int s)         {Spectrum = s;}
    void setMajor(DtRadioMajorModulation m)   {Major = m;}
    void setDetail(int d)                     {Detail = d;}
    void setSystemType(DtRadioSystem s)       {Sys = s;}

    void printData() const;

private:
    unsigned int Spectrum;
    DtRadioMajorModulation Major;
    int Detail;
    DtRadioSystem Sys;
};
class DT_DLL_VL DtTransmitterPdu : public DtPdu
{

```

```
public:

/***** STANDARD CONSTRUCTORS *****/

    DtTransmitterPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtTransmitterPdu(const DtNetTransmitterPdu *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    static DtPdu *create(const DtNetPacket *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    static void addCallback(DtExerciseConn *conn,
        DtTransmitterPduCb cb, void *usr);
    static void removeCallback(DtExerciseConn *conn,
        DtTransmitterPduCb cb, void *usr);

/***** INSPECTOR FUNCTIONS *****/

    const DtEntityIdentifier &entityId() const;
    unsigned int radioId() const;
    #if DtDISVERS >= 5
        const DtRadioEntityType &radioEntityType() const;
    #else
        const DtEntityType &radioEntityType() const;
    #endif
    DtRadioTransmitState transmitState() const;
    DtRadioInputSource inputSource() const;
    DtConstVector worldAntennaLoc() const;
    DtConstVector relativeAntennaLoc() const;
    DtAntennaPatternType antennaPatternType() const;
    int antennaParamsLength() const;
    unsigned long frequency(int upperLower = DtFREQ_LOWER) const;
    float bandwidth() const;
    float power() const;
    const DtModulationType &modulationType() const;
    DtRadioCryptoSystem cryptoSystem() const;
    int cryptoKeyId() const;
    int modulationParamsLength() const;
    void getNetModulationParams(void *destination) const;
    void getNetAntennaParams(void *destination) const;
    const DtBeamAntennaPatternRecord &antennaParams() const;

/***** MUTATOR FUNCTIONS *****/

    void setEntityId(const DtEntityIdentifier &id);
    void setRadioId(unsigned int id);
    #if DtDISVERS >= 5
        void setRadioEntityType(const DtRadioEntityType &type);
    #endif
    void setRadioEntityType(const DtEntityType &type);
    void setTransmitState(DtRadioTransmitState state);
```

```

    void setInputSource(DtRadioInputSource source);
    void setWorldAntennaLoc(DtConstVector loc);
    void setRelativeAntennaLoc(DtConstVector loc);
    void setAntennaPatternType(DtAntennaPatternType type);
    void setFrequency(unsigned long upper, unsigned long lower);
    void setFrequency(unsigned long freq);
    void setBandwidth(float bw);
    void setPower(float power);
    void setModulationType(const DtModulationType &type);
    void setCryptoSystem(DtRadioCryptoSystem sys);
    void setCryptoKeyId(int key);
    void setModulationParams(void *networkRepresentation, int size);
    void setAntennaPatternParams(void *networkRepresentation, int
size);
    void setAntennaPatternParams(const DtBeamAntennaPatternRecord
&rec);

    void printData() const;
protected:
    DtPduKind internalGetPduKind() const {return DtTransmitterPdu-
Kind;}
    DtEntityIdentifier theId;
#if DtDISVERS >= 5
    DtRadioEntityType theType;
#else
    DtEntityType theType;
#endif
    DtVector wLoc;
    DtVector rLoc;
    DtModulationType modType;
    DtBeamAntennaPatternRecord pat;
    void setModulationParamsLength(int length);
    void setAntennaParamsLength(int length);
    void *antennaParamsAddr() const;
    void *modulationParamsAddr() const;
    virtual void printModulationParams() const;
    virtual void printAntennaParams() const;
};

#endif // !DtHLA

```

trkJamBeamDec.h

```
#if DtHLA

#ifndef DtTrackJamBeamDecoder_H_
#define DtTrackJamBeamDecoder_H_

#include "chooseDefines.h"
#include "baseEmitttrDec.h"

// This is used to declare attribute decoding functions
// which are members of the class.
#define DtDECLARE_TRKJAMBEAM_ATTR_DECODER(attrName) \
    DtDECLARE_ATTR_DECODER(DtEmitterBeamRepository, attrName)

// class DtTrackJamBeamDecoder:
//
// Instances of DtTrackJamBeamDecoder are

class DT_DLL_VL DtTrackJamBeamDecoder : public DtBaseEmitterBeamDe-
coder
{
public:

    // default constructor
    DtTrackJamBeamDecoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtTrackJamBeamDecoder();

    // copy constructor
    DtTrackJamBeamDecoder(const DtTrackJamBeamDecoder& orig);

    // assignment operator
    DtTrackJamBeamDecoder& operator=(const DtTrackJamBeamDecoder&
orig);

public:

    // Create a DtBaseEntityDecoder. Caller is responsible for dele-
    tion.
    static DtHlaStateDecoder* create(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

protected:
```

```

    // Add decoding functions for the attributes.
    // Note: Non-virtual since meant to be called from constructors
to
    //      add decoders.
    void addMyDecoders();

    // static decode<AttributeName> member functions.
    DtDECLARE_TRKJAMBEAM_ATTR_DECODER(JammingModeSequence);
    DtDECLARE_TRKJAMBEAM_ATTR_DECODER(TrackJamObjectIDs);
    DtDECLARE_TRKJAMBEAM_ATTR_DECODER(TrackOrJammer);
};
// These macros are used to define simple attribute decoder func-
tions
// (members of a derived DtHlaStateDecoder that decode
// individual attributes). The simple decoding is just to
// convert the value pulled out of the ahvps to a host type using
// the cast expression castExpr, and pass it to the mutator func-
tion.
#define DtDEFINE_SIMPLE_BASE_TRKJAMBEAM_ATTR_DECODER_WITH_CAST( \
    attrName, netType, mutator, castExpr) \
    DtDEFINE_SIMPLE_ATTR_DECODER_WITH_CAST( \
        DtTrackJamBeamDecoder, DtEmitterBeamRepository, \
        attrName, netType, mutator, castExpr)

#define DtDEFINE_SIMPLE_BASE_TRKJAMBEAM_ATTR_DECODER( \
    attrName, netType, inspector) \
    DtDEFINE_SIMPLE_BASE_TRKJAMBEAM_ATTR_DECODER_WITH_CAST( \
        attrName, netType, inspector, (netType))

#endif    //DtTrackJamBeamDecoder_H_

```

trkJamBeamEnc.h

```
#if DtHLA

#ifndef DtTrackJamBeamEncoder_H_
#define DtTrackJamBeamEncoder_H_

#include "chooseDefines.h"
#include "baseEmitttrEnc.h"

// This is used to declare static member functions for determining
// whether or not an attribute needs to be encoded.
#define DtDECLARE_BASE_TRKJAMBEAM_ATTR_CHECKER(attrName) \
    DtDECLARE_ATTR_CHECKER(DtEmitterBeamRepository, attrName)

// This is used to declare attribute encoding functions
// which are members of the class.
#define DtDECLARE_BASE_TRKJAMBEAM_ATTR_ENCODER(attrName) \
    DtDECLARE_ATTR_ENCODER(DtEmitterBeamRepository, attrName)

// class DtTrackJamBeamEncoder:
//
// Instances of DtTrackJamBeamEncoder are

class DT_DLL_VL DtTrackJamBeamEncoder : public DtBaseEmitterBeamEncoder
{
public:

    // default constructor
    DtTrackJamBeamEncoder(
        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtTrackJamBeamEncoder();

    // copy constructor
    DtTrackJamBeamEncoder(const DtTrackJamBeamEncoder& orig);

    // assignment operator
    DtTrackJamBeamEncoder& operator=(const DtTrackJamBeamEncoder&
orig);

public:

    // Create a DtBaseEntityEncoder. Caller is responsible for deletion.
    static DtHlaStateEncoder* create(
```

```

        DtExerciseConn* exConn,
        DtObjClassDesc* classDesc);

protected:
    DtDECLARE_BASE_TRKJAMBEAM_ATTR_ENCODER(JammingModeSequence);
    DtDECLARE_BASE_TRKJAMBEAM_ATTR_ENCODER(TrackJamObjectIDs);
    DtDECLARE_BASE_TRKJAMBEAM_ATTR_ENCODER(TrackOrJammer);

    DtDECLARE_BASE_TRKJAMBEAM_ATTR_CHECKER(JammingModeSequence);
    DtDECLARE_BASE_TRKJAMBEAM_ATTR_CHECKER(TrackJamObjectIDs);
    DtDECLARE_BASE_TRKJAMBEAM_ATTR_CHECKER(TrackOrJammer);
};
// This macro is used to define simple attribute encoder functions
// (members of a derived DtHlaStateEncoder that encode
// individual attributes). The simple encoding is just to implicitly
// convert the value returned by the inspector to a netType,
// and add it to the ahvps.
#define DtDEFINE_SIMPLE_BASE_TRKJAMBEAM_ATTR_ENCODER( \
    attrName, netType, inspector) \
    DtDEFINE_SIMPLE_ATTR_ENCODER(DtTrackJamBeamEncoder, \
        DtEmitterBeamRepository, \
        attrName, netType, inspector)

// This macro is used to define simple attribute checker functions
// (members of a derived DtHlaStateEncoder that check whether
// a particular attribute is out of date. The simple implementation
// is to just compare the values returned by the two StateReps'
// inspector functions.
#define DtDEFINE_SIMPLE_BASE_TRKJAMBEAM_ATTR_CHECKER( \
    attrName, inspector) \
    DtDEFINE_SIMPLE_ATTR_CHECKER(DtTrackJamBeamEncoder, \
        DtEmitterBeamRepository, \
        attrName, inspector)

#endif //DtTrackJamBeamEncoder_H_

```

tStampType.h

```
#include "chooseDefines.h"
```

```
enum DtTimeStampType  
{  
    DtTimeStampRelative,  
    DtTimeStampAbsolute  
};
```


undwaAcBeam.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"

#include "packets/undwaAcP.h"

// class DtUnderwaterAcousticBeam:
//
// DtUnderwaterAcousticBeam is a class which contains the beam
// fundamental
// data parameters for the underwater acoustics PDU

class DT_DLL_VL DtUnderwaterAcousticBeam
{
public:

    // Constructors
    DtUnderwaterAcousticBeam();
    DtUnderwaterAcousticBeam(
        const DtUnderwaterAcousticBeam& orig);

    // Assignment operator
    DtUnderwaterAcousticBeam& operator=(
        const DtUnderwaterAcousticBeam& orig);
    DtUnderwaterAcousticBeam& operator=(
        const DtNetUnderwaterAcousticBeam& orig);

    // Destructor
    virtual ~DtUnderwaterAcousticBeam();

    // Set/Get the beam data length
    virtual void setBeamDataLength(DtU8 length);
    virtual DtU8 beamDataLength() const;

    // Set/Get the beam ID number
    virtual void setBeamID_Number(DtU8 num);
    virtual DtU8 beamID_Number() const;

    // Set/Get the active emission parameter index
    virtual void setActiveEmissionParmIndex(
        DtUnderwaterAcousticActiveEmissionParmIndex ind);
    virtual DtUnderwaterAcousticActiveEmissionParmIndex
        activeEmissionParmIndex() const;

```

```
// Set/Get the acoustic scan pattern
virtual void setScanPattern(DtUnderwaterAcousticScanPattern
value);
virtual DtUnderwaterAcousticScanPattern scanPattern() const;

// Set/Get the beam center azimuth
virtual void setBeamCenterAzimuth(DtFloat32 value);
virtual DtFloat32 beamCenterAzimuth() const;

// Set/Get the azimuthal beamwidth
virtual void setAzimuthalBeamwidth(DtFloat32 value);
virtual DtFloat32 azimuthalBeamwidth() const;

// Set/Get the beam center D/E
virtual void setBeamCenterD_E(DtFloat32 value);
virtual DtFloat32 beamCenterD_E() const;

// Set/Get the D/E beamwidth
virtual void setD_E_Beamwidth(DtFloat32 value);
virtual DtFloat32 D_E_Beamwidth() const;

// Print the information contained in the record
virtual void print() const;

// Load/unload data from a packet record
virtual void unloadNetRecord(
    const DtNetUnderwaterAcousticBeam& record);
virtual void loadNetRecord(
    DtNetUnderwaterAcousticBeam& record) const;

protected:

// Data members:
DtU8                myBeamDataLength;
DtU8                myBeamID_Number;

DtUnderwaterAcousticActiveEmissionParmIndex
    myActiveEmissionParmIndex;

DtUnderwaterAcousticScanPattern myScanPattern;

DtFloat32            myBeamCenterAzimuth;
DtFloat32            myAzimuthalBeamwidth;
DtFloat32            myBeamCenterD_E;
DtFloat32            myD_E_Beamwidth;

};
#endif
#endif
```

undwaAcEmitSys.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"
#include "mathTypes.h"

#include "packets/undwaAcP.h"

// class DtUnderwaterAcousticEmitterSystem:
//
// DtUnderwaterAcousticEmitterSystem is a class which contains
// information
// describing the acoustic emitter systems in an underwater acous-
// tics PDU.

class DT_DLL_VL DtUnderwaterAcousticEmitterSystem
{
public:

    // Constructors
    DtUnderwaterAcousticEmitterSystem();
    DtUnderwaterAcousticEmitterSystem(
        const DtUnderwaterAcousticEmitterSystem& orig);

    // Assignment operator
    DtUnderwaterAcousticEmitterSystem& operator=(
        const DtUnderwaterAcousticEmitterSystem& orig);
    DtUnderwaterAcousticEmitterSystem& operator=(
        const DtNetUnderwaterAcousticEmitterSystem& orig);

    // Destructor
    virtual ~DtUnderwaterAcousticEmitterSystem();

    // Set/Get the data length
    virtual void setDataLength(DtU8 value);
    virtual DtU8 dataLength() const;

    // Set/Get the number of beams
    virtual void setNumberOfBeams(DtU8 value);
    virtual DtU8 numberOfBeams() const;

    // Set/Get the system name
    virtual void setAcousticName(DtUnderwaterAcousticSystemName
name);
    virtual DtUnderwaterAcousticSystemName acousticName() const;

```

```
// Set/Get the function
virtual void setFunction(DtUnderwaterAcousticFunction function);
virtual DtUnderwaterAcousticFunction function() const;

// Set/Get the acoustic ID number
virtual void setAcousticID(DtU8 id);
virtual DtU8 acousticID() const;

// Set/Get the location
virtual void setLocation(DtConstVector coords);
virtual DtConstVector location() const;

// Print the information contained in the record
virtual void print() const;

// Load/Unload data from a packet record
virtual void unloadNetRecord(
    const DtNetUnderwaterAcousticEmitterSystem& record);
virtual void loadNetRecord(
    DtNetUnderwaterAcousticEmitterSystem& record) const;
```

protected:

```
// Data members:
DtU8                                myDataLength;
DtU8                                myNumberOfBeams;
DtUnderwaterAcousticSystemName     myAcousticName;
DtUnderwaterAcousticFunction       myFunction;
DtU8                                myAcousticID;
DtVector                            myLocation;
```

```
};
#endif
#endif
```

undwaAcPasAct.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"

#include "packets/undwaAcP.h"

// class DtUnderwaterAcousticAddtnlPassiveAct:
//
// DtUnderwaterAcousticAddtnlPassiveAct is a class which contains
// the
// additional passive activity information for the underwater acous-
// tics PDU

class DT_DLL_VL DtUnderwaterAcousticAddtnlPassiveAct
{
public:

    // Constructors
    DtUnderwaterAcousticAddtnlPassiveAct();
    DtUnderwaterAcousticAddtnlPassiveAct(
        const DtUnderwaterAcousticAddtnlPassiveAct& orig);

    // Assignment operator
    DtUnderwaterAcousticAddtnlPassiveAct& operator=(
        const DtUnderwaterAcousticAddtnlPassiveAct& orig);
    DtUnderwaterAcousticAddtnlPassiveAct& operator=(
        const DtNetUnderwaterAcousticAddtnlPassiveAct& orig);

    // Destructor
    virtual ~DtUnderwaterAcousticAddtnlPassiveAct();

    // Set/Get the parameter index
    virtual void setParmIndex(
        DtUnderwaterAcousticAdditionalPassiveActivityParmIndex value);
    virtual DtUnderwaterAcousticAdditionalPassiveActivityParmIndex
        parmIndex() const;

    // Set/Get the APA value
    virtual void setAPA(DtInt16 value);
    virtual DtInt16 APA() const;

    // Print the information contained in the record
    virtual void print() const;

```

```
// Load/Unload data from a packet record
virtual void unloadNetRecord(
    const DtNetUnderwaterAcousticAddtnlPassiveAct& record);
virtual void loadNetRecord(
    DtNetUnderwaterAcousticAddtnlPassiveAct& record) const;

protected:

    // Data members:
    DtUnderwaterAcousticPassiveParmIndex myParmIndex;
    DtU16                                myAPA;

};
#endif
#endif
```

undwaAcPdu.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "hostStructs.h"
#include "pdu.h"
#include "exerciseConn.h"

#include "mathTypes.h"

#include "packets/undwaAcP.h"

#include "undwaAcBeam.h"
#include "undwaAcEmtSys.h"
#include "undwaAcPasAct.h"
#include "undwaAcShaft.h"

#include "chooseDefines.h"

// class DtUnderwaterAcousticPdu:
//
// Instances of DtUnderwaterAcousticPdu are used to send information
// about
// simulated underwater acoustic emissions.

// DtUnderwaterAcousticPdu callback function signature.
typedef void (*DtUnderwaterAcousticPduCb)
    (DtUnderwaterAcousticPdu* pdu, void* usr);

class DT_DLL_VL DtUnderwaterAcousticPdu : public DtPdu
{
public:
    // Constructors
    DtUnderwaterAcousticPdu(DtBufferPtr buffer =
DtUSE_INTERNAL_BUFFER);
    DtUnderwaterAcousticPdu(const DtNetUnderwaterAcousticPdu*
initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtUnderwaterAcousticPdu(const DtUnderwaterAcousticPdu& orig);

    // destructor
    virtual ~DtUnderwaterAcousticPdu();

    // Assignment operator

```

```
DtUnderwaterAcousticPdu& operator=(const DtUnderwaterAcousticPdu&
orig);

// Set/Get emitting entity identifier
virtual void setEmittingEntityID(const DtEntityIdentifier&
entity);
virtual const DtEntityIdentifier& emittingEntityID() const;

// Set/Get event identifier
virtual void setEventID(const DtEventID& event);
virtual const DtEventID& eventID() const;

// Set/Get state/change update indicator
virtual void setStateChangeUpdate(DtUnderwaterAcousticUpdate
upd);
virtual DtUnderwaterAcousticUpdate stateChangeUpdate() const;

// Set/Get passive parameter index
virtual void setPassiveParameterIndex(
DtUnderwaterAcousticPassiveParmIndex ind);
virtual DtUnderwaterAcousticPassiveParmIndex
passiveParameterIndex() const;

// Set/Get propulsion plant configuration
virtual void setPropulsionPlantConfig(
DtUnderwaterAcousticPropulsionPlantConfig config);
virtual DtUnderwaterAcousticPropulsionPlantConfig
propulsionPlantConfig() const;

// Set/Get number of shafts
// May return DtFalse if PDU size would exceed the maximum PDU
size
virtual DtBoolean setNumShafts(DtU8 num);
virtual DtU8 numShafts() const;

// Set/Get number of additional passive activities
// May return DtFalse if PDU size would exceed the maximum PDU
size
virtual DtBoolean setNumAdditionalPassiveAct(DtU8 num);
virtual DtU8 numAdditionalPassiveAct() const;

// Set/Get number of emitter systems
// May return DtFalse if PDU size would exceed the maximum PDU
size
virtual DtBoolean setNumEmitterSystems(DtU8 num);
virtual DtU8 numEmitterSystems() const;

// Set/Get number of beam data records for a particular emitter
system.
```



```

    // May return DtFalse if PDU size would exceed the maximum PDU
    size
    // or if the emitter system number is invalid.
    virtual DtBoolean setNumBeams(int emitterSystemIndex, DtU8 num);
    virtual DtU8 numBeams(int emitterSystemIndex) const;

    // Set/Get a shaft speed data record.
    // Returns false if the record does not exist.
    virtual DtBoolean setShaftSpeed(int index,
        const DtUnderwaterAcousticShaftSpeed& record);
    virtual DtBoolean getShaftSpeed(int index,
        DtUnderwaterAcousticShaftSpeed& record) const;

    // Set/Get an additional passive activity data record.
    // Returns false if the record does not exist.
    virtual DtBoolean setAddtnlPassiveAct(int index,
        const DtUnderwaterAcousticAddtnlPassiveAct& record);
    virtual DtBoolean getAddtnlPassiveAct(int index,
        DtUnderwaterAcousticAddtnlPassiveAct& record) const;

    // Set/Get an underwater acoustic emitter system.
    // Returns false if the record does not exist.
    // Data length field is automatically set by the PDU class.
    virtual DtBoolean setEmitterSystem(int index,
        const DtUnderwaterAcousticEmitterSystem& record);
    virtual DtBoolean getEmitterSystem(int index,
        DtUnderwaterAcousticEmitterSystem& record) const;

    // Set/Get beam fundamental data parameters for a beam.
    // Returns false if the record does not exist.
    // Data length field is automatically set by the PDU class.
    virtual DtBoolean setBeam(int emitterSystemIndex, int beamIndex,
        const DtUnderwaterAcousticBeam& record);
    virtual DtBoolean getBeam(int emitterSystemIndex, int beamIndex,
        DtUnderwaterAcousticBeam& record) const;

    // Print the data contained by this PDU
    virtual void printData() const;

public:

    // Create a DtUnderwaterAcousticPdu.
    // Caller is responsible for deletion.
    static DtPdu* create(const DtNetPacket* initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    // Add (register)/Remove (deregister) function to be called when
    // pdu is received.
    static void addCallback(DtExerciseConn* conn,
        DtUnderwaterAcousticPduCb cb, void* usr);

```

```
static void removeCallback(DtExerciseConn* conn,
                          DtUnderwaterAcousticPduCb cb, void* usr);
```

protected:

```
// Returns the internal PDU kind
virtual DtPduKind internalGetPduKind() const;
```

```
// Casts the netPdu element to our network struct
virtual DtNetUnderwaterAcousticPdu* netPduStruct() const;
```

```
// Returns a pointer to the first byte of variable length data
virtual char* var() const;
```

```
// Casts the 'this' pointer for changing elements in a const
function
```

```
// Non virtual since you can't override based on return type
DtUnderwaterAcousticPdu* self() const;
```

```
// Returns a buffer offset to the given shaft speed record
virtual int shaftSpeedRecordOffset(int recordIndex) const;
```

```
// Returns a buffer offset to the given additional passive
activity record
virtual int passiveActRecordOffset(int recordIndex) const;
```

```
// Returns a buffer offset to the given acoustic emitter system
virtual int emitterSystemRecordOffset(int recordIndex) const;
```

```
// Returns a buffer offset to the given beam fundamental data
parameter of
```

```
// a particular emitter system.
virtual int beamDataRecordOffset(
    int emitterSystemIndex, int beamIndex) const;
```

protected:

```
// Size of PDU with zero parameter records.
static int theStaticSize;
```

protected:

```
// Temporary objects for reference returns
DtEntityIdentifier          myEntityID;
DtEventID                  myEventID;
```

```
};
#endif
#endif
```

undwaAcShaft.h

```

#if !DtHLA
#if DtDISVERS >= 5

#include "MachineTypes.h"
#include "disEnums.h"

#include "packets/undwaAcP.h"

// class DtUnderwaterAcousticShaftSpeed:
//
// DtUnderwaterAcousticShaftSpeed is a class which contains the
// shaft speed
// information for the underwater acoustics PDU.

class DT_DLL_VL DtUnderwaterAcousticShaftSpeed
{
public:

    // Constructors
    DtUnderwaterAcousticShaftSpeed();
    DtUnderwaterAcousticShaftSpeed(const DtUnderwaterAcoustic-
ShaftSpeed& orig);

    // Assignment operator
    DtUnderwaterAcousticShaftSpeed& operator=(
        const DtUnderwaterAcousticShaftSpeed& orig);
    DtUnderwaterAcousticShaftSpeed& operator=(
        const DtNetUnderwaterAcousticShaftSpeed& orig);

    // Destructor
    virtual ~DtUnderwaterAcousticShaftSpeed();

    // Set/Get the current shaft RPM
    virtual void setCurrentShaftRPM(DtU16 value);
    virtual DtU16 currentShaftRPM() const;

    // Set/Get the ordered shaft RPM
    virtual void setOrderedShaftRPM(DtU16 value);
    virtual DtU16 orderedShaftRPM() const;

    // Set/Get the shaft RPM rate of change
    virtual void setShaftRPM_RateOfChange(DtU32 value);
    virtual DtU32 shaftRPM_RateOfChange() const;

    // Print the information contained in the record
    virtual void print() const;

```

```
// Load/Unload data from a packet record
virtual void unloadNetRecord(
    const DtNetUnderwaterAcousticShaftSpeed& record);
virtual void loadNetRecord(
    DtNetUnderwaterAcousticShaftSpeed& record) const;

protected:

    // Data members:
    DtU16          myCurrentShaftRPM;
    DtU16          myOrderedShaftRPM;
    DtU32          myShaftRPM_RateOfChange;

};
#endif
#endif
```

unknownHObj.h

```

#if DtHLA

#include "hlaObj.h"

// class DtUnknownHlaObject:
//
// Instances of DtUnknownHlaObject.

class DT_DLL_VL DtUnknownHlaObject : public DtHlaObject
{
public:

    // Constructor for local objects.
    DtUnknownHlaObject(
        DtExerciseConn* exConn,
        const char* name,
        DtObjClassDesc* classDesc);

    // Constructor for reflected objects.
    DtUnknownHlaObject(
        DtExerciseConn* exConn,
        RTI::ObjectHandle id,
        const char* name,
        DtObjClassDesc* classDesc);

    // destructor
    virtual ~DtUnknownHlaObject();

    // Does nothing. Use the version of update that takes an ahvps
    // to send data for a DtUnknownHlaObject.
    virtual void update();

    // Sends the ahvps through our DtExerciseConn. Does not look
    // at the contents of ahvps. It is the caller's responsibility to
    // insure that all attributes in the message are in the set of
    // attributes needed by the federation (obtained using
    // attributesNeededByFederation). It is also the responsibility
of the
    // user of DtUnknownHlaObject to insure that requested attributes
    // (obtained via requestedAttributes) get updated in a timely
manner.
    // clearRequestedAttributes should be called after each time
requested
    // attributes are updated.
    virtual void update(const RTI::AttributeHandleValuePairSet&
ahvps);

```

protected:

```
    // copy constructor
    DtUnknownHlaObject(const DtUnknownHlaObject& orig);

    // assignment operator
    DtUnknownHlaObject& operator=(const DtUnknownHlaObject& orig);

    // Does nothing. You can access information from incoming
    // update messages by using DtHlaObject::addPostReflectCb to
    // register a function that will be called whenever an update
    // msg is received.
    virtual void reflect(const DtStateMsg& stateMsg);
};
#endif
```

unknownPdu.h

```
#if !DtHLA

#include "pdu.h"
#define DtUnknownStaticSize 12

class DT_DLL_VL DtUnknownPdu;
typedef void (*DtUnknownPduCb)(DtUnknownPdu *pdu, void *usr);

class DT_DLL_VL DtUnknownPdu : public DtPdu
{
public:

/***** STANDARD CONSTRUCTORS *****/

    DtUnknownPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtUnknownPdu(const DtNetPduHeader *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

    static DtPdu *create(const DtNetPacket *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    static void addCallback(DtPduKind kind, DtExerciseConn *conn,
        DtUnknownPduCb cb, void *usr);
    static void removeCallback(DtPduKind kind, DtExerciseConn *conn,
        DtUnknownPduCb cb, void *usr);

/***** MUTATOR FUNCTIONS *****/

    void setKind(DtPduKind kind);

protected:
    DtPduKind internalGetPduKind() const;
};
#endif
```

updCtlCbInfo.h

```
#if DtHLA

#if !DtRTI103

#include "updCtlCbList.h"
#include "RTI.hh"
#include "stateMsg.h"

// class DtUpdateControlCbInfo:
//
// Instances of DtUpdateControlCbInfo are used for providing
// a callback when the RTI update control services are invoked.
// This class is a helper class to DtUpdateControlCbList.

class DT_DLL_VL DtUpdateControlCbInfo
{
public:

    // constructor
    DtUpdateControlCbInfo(RTI::ObjectHandle hand,
        DtUpdateControlCb cb,
        void* usr);

    // destructor
    virtual ~DtUpdateControlCbInfo();

    // copy constructor
    DtUpdateControlCbInfo(
        const DtUpdateControlCbInfo& orig);

    // assignment operator
    DtUpdateControlCbInfo& operator=(
        const DtUpdateControlCbInfo& orig);

    // Check whether one callback info is equal to another.
    virtual int match(const DtUpdateControlCbInfo* const other)
const;

    // Check whether one callback info is equal to another.
    virtual int operator==(const DtUpdateControlCbInfo& other) const;

    // Get the object handle.
    virtual const RTI::ObjectHandle& objectHandle() const;

    // Get the callback function.
    virtual const DtUpdateControlCb& cbFunction() const;
```



```
// Get the user data.
virtual void* userData() const;

protected:

    RTI::ObjectHandle myHand;
    DtUpdateControlCb myFunc;
    void* myUsr;
};
typedef DtUpdateControlCbInfo DtUpdatesOnCbInfo;
typedef DtUpdateControlCbInfo DtUpdatesOnCbInfo;

#endif
#endif
```

updCtlCbList.h

```
#if DtHLA

#if !DtRTI103

#include "vllist.h"
#include "RTI.hh"
#include "chooseDefines.h"

// class DtUpdateControlCbList:
//
// Instances of DtUpdateControlCbList are used to provide
// a list of callbacks (DtUpdateControlCbInfo) which are
// called when an object starts updating.

typedef void (*DtUpdateControlCb)(
    RTI::ObjectHandle handle,
    const RTI::AttributeHandleSet& attributeSet,
    void* usr);

typedef DtUpdateControlCb DtUpdatesOnCb;
typedef DtUpdateControlCb DtUpdatesOffCb;

class DT_DLL_VL DtUpdateControlCbList
{
public:

    // default constructor
    DtUpdateControlCbList();

    // destructor
    virtual ~DtUpdateControlCbList();

    // copy constructor
    DtUpdateControlCbList(
        const DtUpdateControlCbList& orig);

    // assignment operator
    DtUpdateControlCbList& operator=(
        const DtUpdateControlCbList& orig);

    // Add/Remove a start updates callback to/from the list.
    // Returns whether add/remove took place.
    virtual int add(RTI::ObjectHandle hand, DtUpdateControlCb cb,
        void* usr);
    virtual int remove(RTI::ObjectHandle hand, DtUpdateControlCb cb,
        void* usr);
```

```
// Remove all callbacks from the list.
virtual void clear();

// Callbacks are for a particular object class handle.
// Call any callbacks in the list for the object class handle.
virtual void apply(RTI::ObjectHandle handle,
    const RTI::AttributeHandleSet& hSet);
```

protected:

```
// Remove all callbacks from the list.
// Non-virtual since meant to be called from dtor, otherwise
// should call clear().
void dtorClear();

// Copy another callback list into this object's callback list.
// Non-virtual since meant to be called from ctor.
void ctorCopyList(const DtUpdateControlCbList& orig);
```

protected:

```
    DtList myList;
};
typedef DtUpdateControlCbList DtUpdatesOnCbList;
typedef DtUpdateControlCbList DtUpdatesOffCbList;

#endif
#endif
```

utmCoord.h

```
#include "vlVector.h"
#include "geodCoord.h"

// class DtUtmCoord:
//
// Instances of DtUtmCoord are used to represent UTM coordinates.
// Universal Transverse Mercator projection coordinate with respect
// to
// reference set by DtUtmInit.

class DT_DLL_VL DtUtmCoord : public DtVector
{
    friend DT_DLL_VL void DtGeodToUtm(const DtGeodeticCoord& gd,
DtUtmCoord& utm);
    friend DT_DLL_VL void DtGeocToUtm(DtConstVector gd, DtUtmCoord&
utm);

public:

    //
    // Note: For coordinate conversions, DtUtmInit must be called
    first.
    //

    // default constructor
    DtUtmCoord();

    // constructor
    DtUtmCoord(double east, double north, double up);

    // constructor, construct UTM from geodetic
    DtUtmCoord(const DtGeodeticCoord& geodetic);

    // destructor
    virtual ~DtUtmCoord();

    // copy constructor
    DtUtmCoord(const DtUtmCoord& orig);

    // assignment operator
    DtUtmCoord& operator=(const DtUtmCoord& orig);

    // assignment operator, convert geodetic to UTM.
    DtUtmCoord& operator=(const DtGeodeticCoord& geodetic);

    // Set from geodetic coordinate
    // (convert geodetic to UTM).
```

```

// Can also use assignment operator.
virtual DtUtmCoord& setGeodetic(const DtGeodeticCoord& geodetic);

// Get geodetic coordinate equivalent
// (convert UTM to geodetic).
virtual void getGeodetic(DtGeodeticCoord& geodetic) const;

// Get geodetic coordinate equivalent
// (convert UTM to geodetic).
// Note that this is not returning a reference.
virtual DtGeodeticCoord geodetic() const;

// Set from geocentric coordinate
// (convert geocentric to UTM).
// Can also use assignment operator.
virtual DtUtmCoord& setGeocentric(DtConstVector geocentric);

// Get geocentric coordinate equivalent
// (convert UTM to geocentric).
virtual void getGeocentric(DtVectorRef geocentric) const;

// Get geocentric coordinate equivalent
// (convert UTM to geocentric).
// Note that this is not returning a reference.
virtual DtVector geocentric() const;

// Set/Get east component.
virtual void setEast(DtFloat64 east);
virtual DtFloat64 east() const;

// Set/Get north component.
virtual void setNorth(DtFloat64 north);
virtual DtFloat64 north() const;

// Set/Get up component.
virtual void setUp(DtFloat64 up);
virtual DtFloat64 up() const;

// Get the heading that northing is off of north
virtual double gridDeclination() const;
};
// Coordinate conversions. DtUtmInit must be called first.

// Use of these functions is discouraged in favor of DtUtmCoord
// member functions which perform conversions.
DT_DLL_VL void DtGeodToUtm(const DtGeodeticCoord& in, DtUtmCoord&
out);
DT_DLL_VL void DtUtmToGeod(const DtUtmCoord& utm, DtGeodeticCoord&
out);

```

```
// Use of these functions is discouraged in favor of DtUtmCoord
// member functions which perform conversions.
DT_DLL_VL void DtGeocToUtm(DtConstVector in, DtUtmCoord& out);
DT_DLL_VL void DtUtmToGeoc(const DtUtmCoord& in, DtVectorRef out);

DT_DLL_VL DtConstDcm DtNedToEnuDcm();
DT_DLL_VL DtConstDcm DtEnuToNedDcm();

// Use of these functions is discouraged in favor of DtUtmCoord
// member functions which perform conversions.
DT_DLL_VL double DtUtmGridDeclination(const DtUtmCoord& utm);
DT_DLL_VL double DtUtmGridDeclination(double lat, double lon);

DT_DLL_VL void DtGridDeclinationDcm(double gridDecl, DtDcmRef
utm2ned);

enum DtPolarDirection { DtEast, DtWest, DtNorth, DtSouth };

typedef struct
{
    double deg, min, sec;
    DtPolarDirection direction;
} DtDegMinSec;

// Sets reference point for all DtUtmCoords
// Vrlink presently supports only one UTM reference at a time,
// but the reference point may be reinitialized at any point.
//
// If standardOffsets is true, the standard false easting (500,000
m)
// and false northing (10,000,000 m only when south of equator) will
// be applied.
//
// DtUtmInit expects lonRef between +/- 180.
// The UTM zone will be set by the zone that the reference point
falls
// in, where zone N extends from (N*6 - 186) to (N*6 - 180).

DT_DLL_VL void DtUtmInit(const DtDegMinSec& latRef,
    const DtDegMinSec& lonRef,
    int standardOffsets=1,
    int verbose=1);

// Some constants:

// Origin of the IITSEC '92 terrain database
const DtDegMinSec DtUtmLatRef1_1 = { 35.0, 15.0, 0.0, DtNorth};
const DtDegMinSec DtUtmLonRef1_1 = {122.0, 3.0, 59.0, DtWest};

// Origin of the IITSEC '94 & '95 terrain database
```

```
const DtDegMinSec DtUtmLatRef_IITSEC_93 = { 35.0, 15.0, 0.0,
DtNorth};
const DtDegMinSec DtUtmLonRef_IITSEC_93 = {122.0, 4.0, 0.0,
DtWest};

// The standard UTM reference for a zone is the point along the
// zone's central meridian at the equator. Standard UTM positions
// also use the standard false easting and false easting (see DtUtmInit)
//
// This is the standard UTM reference position for UTM zone 10

const DtDegMinSec DtUtmLatRefZone10 = { 0.0, 0.0, 0.0, DtNorth};
const DtDegMinSec DtUtmLonRefZone10 = {123.0, 0.0, 0.0, DtWest};

// inline functions
```

utmView.h

```
#include "LibMatrix.h"
#include "utmCoord.h"
// class DtUtmView:
//
// Instances of DtUtmView are used to create a UTM coordinate view
// of the geocentric coordinates in an Entity State Repository.

class DT_DLL_VL DtUtmView
{
public:

    // constructor
    DtUtmView(DtBaseEntityStateRepository* esr);

    // destructor
    virtual ~DtUtmView();

    // copy constructor
    DtUtmView(const DtUtmView& orig);

    // assignment operator
    DtUtmView& operator=(const DtUtmView& orig);

    // Set/Get the location in UTM coordinates.
    virtual void setLocation(const DtUtmCoord& aLocation);
    virtual const DtUtmCoord& location() const;

    // Set/Get the velocity.
    virtual void setVelocity(const DtUtmCoord& aVelocity);
    virtual const DtUtmCoord& velocity() const;

    // Set/Get the orientation.
    virtual void setOrientation(const DtTaitBryan& orient);
    virtual const DtTaitBryan& orientation() const;

    // Get the body to local coordinate matrix. This is just
    // a matrix representation of the euler angles returned by
    // orientation().
    virtual DtConstDcm bodyToLocal() const;

    // Set/Get the acceleration.
    virtual void setAcceleration(const DtUtmCoord& anAcceleration);
    virtual const DtUtmCoord& acceleration() const;

    // Set/Get the rotational velocity.
    virtual void setRotationalVelocity(DtConstVector bodyRates);
    virtual DtConstVector rotationalVelocity() const;
```


protected:

```
// Cast off const-ness for trivial internal change.
// Non-virtual since can't override on return type.
DtUtmView* self() const;

// Get the matrix which transforms from UTM to geocentric.
virtual DtConstDcm utmToGeocMatrix(DtConstVector geocLoc) const;

// Get the matrix which transforms form geocentric to UTM.
virtual DtConstDcm geocToUtmMatrix(DtConstVector geocLoc) const;
```

protected:

```
DtBaseEntityStateRepository* myESR;

// used as return values by inspectors. Do not necessarily
// hold current state.
DtUtmCoord myLocalLoc;
DtUtmCoord myLocalVel;
DtUtmCoord myLocalAccel;
DtTaitBryan myLocalOrient;
DtDcm myBodyToLocal;

DtDcm myUtmToGeocMatrix;
DtVector myGeocLocForUtmToGeoc;

DtDcm myGeocToUtmMatrix;
DtVector myGeocLocForGeocToUtm;
};
```

varDatumRec.h

```
#include "disEnums.h"
#include "datumIdType.h"
#include "packets/dataP.h"

// class DtVariableDatumRecord:
//
// Instances of DtVariableDatumRecord are used to hold a single
// variable datum
// record and to provide access to its network representation.

class DT_DLL_VL DtVariableDatumRecord
{
public:

    // default constructor
    DtVariableDatumRecord();

    // destructor
    virtual ~DtVariableDatumRecord();

    // copy constructor
    DtVariableDatumRecord(const DtVariableDatumRecord& orig);

    // assignment operator
    DtVariableDatumRecord& operator=(const DtVariableDatumRecord&
orig);

    // comparison operators
    int operator==(const DtVariableDatumRecord& otherguy) const;
    int operator!=(const DtVariableDatumRecord& otherguy) const;

    // Set the working buffer to an externally supplied buffer
    virtual void setExternalBuffer(char* bufferPtr);

    // Set/Get the datum length in bits. This method will re-allocate memory
    // if the buffer is an internal buffer.
    virtual void setDatumBitLength(DtU32 newDatumBitLength);
    virtual DtU32 datumBitLength() const;

    // Get the datum length in bytes (padded to an 8 byte boundary)
    virtual int datumByteLength() const;

    // Set/Get the variable datum ID
    virtual void setDatumID(DtDatumParam value);
    virtual DtDatumParam datumID() const;
```

```
// Set/Get datum value as a DtFloat64
// Set will set the datum bit length to 64
virtual void setDatumValFloat64(DtFloat64 value);
virtual DtFloat64 datumValFloat64() const;

// Set/Get datum value as a DtFloat32
// Set will set the datum bit length to 32
virtual void setDatumValFloat32(DtFloat32 value);
virtual DtFloat32 datumValFloat32() const;

// Set/Get datum value as a DtInt32
// Set will set the datum bit length to 32
virtual void setDatumValInt32(DtInt32 value);
virtual DtInt32 datumValInt32() const;

// Set/Get datum value as a DtU32
// Set will set the datum bit length to 32
virtual void setDatumValU32(DtU32 value);
virtual DtU32 datumValU32() const;

// Set/Get datum value as a DtInt16
// Set will set the datum bit length to 16
virtual void setDatumValInt16(DtInt16 value);
virtual DtInt16 datumValInt16() const;

// Set/Get datum value as a DtU16
// Set will set the datum bit length to 16
virtual void setDatumValU16(DtU16 value);
virtual DtU16 datumValU16() const;

// Set/Get datum value as a DtU8
// Set will set the datum bit length to 8
virtual void setDatumValU8(DtU8 value);
virtual DtU8 datumValU8() const;

// Set/Get datum value as a byte array. Data is copied to and
from
// the network data buffer according to the specified length of
the record.
virtual void setDatumValByteArray(const DtU8* setFrom);
virtual void getDatumValByteArray(DtU8* copyTo) const;

// Get a pointer to the datum value inside of the record
virtual char* datumValPtr();

// Get a const pointer to the datum value inside of the record
virtual const char* constDatumValPtr() const;

// Get the network byte length of the entire datum record
```

```
    virtual int networkByteLength() const;

    // Get a pointer to the buffer which stores the network representation of
    // the datum record
    virtual const char* networkDataBuffer() const;

    // Print the contents of the record
    virtual void print() const;

protected:

    // Cast the buffer to a DtNetVarDatum struct
    virtual DtNetVarDatum* netStruct();

    // Cast the buffer to a const DtNetVarDatum struct
    virtual const DtNetVarDatum* constNetStruct() const;

    // Delete the internal buffer if there is no external buffer
    virtual void deleteIfNoExternal();

    // Compute the total datum size in bytes from the datum bit length
    virtual int datumByteSize(int datumBitLength) const;

    // Compute the total record size in bytes from the datum bit length
    virtual int recordByteSize(int datumBitLength) const;

protected:

    char* myRecordBuffer;
    char* myExternalBuffer;

};
```

vcPdu.h

```

#ifndef DtViewControlPdu_H_
#define DtViewControlPdu_H_

#include "chooseDefines.h"
#include "hostStructs.h"
#include "pdu.h"
#include "packets/viewControlP.h"
#include "mathTypes.h"
#include "globalObjDes.h"

#if !DtHLA

class DT_DLL_VL DtExerciseConn;
class DT_DLL_VL DtViewControlPdu;
typedef DtViewControlPdu DtViewControlInteraction;
typedef void (
    *DtViewControlInteractionCb)(DtViewControlInteraction *inter,
    void *usr);
#include "interaction.h"

#endif // !DtHLA

#if DtHLA
class DT_DLL_VL DtViewControlPdu : public DtPdu
#else
class DT_DLL_VL DtViewControlPdu : public DtInteraction
#endif
{
public:

    static DtPdu *create(const DtNetPacket *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

#if !DtHLA

    static void addCallback(DtExerciseConn *conn,
        DtViewControlInteractionCb cb, void *usr);
    static void removeCallback(DtExerciseConn *conn,
        DtViewControlInteractionCb cb, void *usr);

#endif

    /**
     * ***** INSPECTOR FUNCTIONS *****
     */
    const DtGlobalObjectDesignator& viewerId() const;
    const DtGlobalObjectDesignator& senderId() const;
    DtViewerCommand viewerCommand() const;

```

```
    unsigned long ignoreFlags() const;

/***** MUTATOR FUNCTIONS *****/

    void setViewerId(const DtGlobalObjectDesignator& id);
    void setSenderId(const DtGlobalObjectDesignator& id);
    void setIgnoreFlags(unsigned long flags);

    virtual void setFromString(const char *) {}
    virtual char *string() const {return "";}

    void printData() const;

protected:
    DtPduKind internalGetPduKind() const {return DtViewControlPdu-
Kind;}
    DtGlobalObjectDesignator retId;
    void setViewerCommand(DtViewerCommand com);
};

DtViewControlPdu *DtNewViewControlPdu(DtViewerCommand command);

class DT_DLL_VL DtAbsoluteVCPdu : public DtViewControlPdu
{
public:

/***** STANDARD CONSTRUCTORS *****/

    DtAbsoluteVCPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtAbsoluteVCPdu(const DtNetViewControlPdu *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

/***** INSPECTOR FUNCTIONS *****/

    DtConstVector worldLocation() const;
    const DtTaitBryan &orientation() const;

/***** MUTATOR FUNCTIONS *****/

    void setWorldLocation(DtConstVector loc);
    void setOrientation(const DtTaitBryan &ori);

    void setFromString(const char *str);
    char *string() const;
    void printData() const;

protected:
    DtVector theVec;
    DtTaitBryan theAngles;
};
```

```

class DT_DLL_VL DtTetherVCPdu : public DtViewControlPdu
{
public:

/***** STANDARD CONSTRUCTORS *****/

    DtTetherVCPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtTetherVCPdu(const DtNetViewControlPdu *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

/***** INSPECTOR FUNCTIONS *****/

    DtConstVector topoUpOffset() const;
    const DtTaitBryan &orientation() const;
    const DtGlobalObjectDesignator& attachId() const;

/***** MUTATOR FUNCTIONS *****/

    void setTopoUpOffset(DtConstVector offset);
    void setOrientation(const DtTaitBryan &ori);
    void setAttachId(const DtGlobalObjectDesignator& id);

    void setFromString(const char *str);
    char *string() const;
    void printData() const;

protected:
    DtVector theVec;
    DtTaitBryan theAngles;
    DtGlobalObjectDesignator theId;
};

class DT_DLL_VL DtCompassVCPdu : public DtViewControlPdu
{
public:

/***** STANDARD CONSTRUCTORS *****/

    DtCompassVCPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtCompassVCPdu(const DtNetViewControlPdu *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

/***** INSPECTOR FUNCTIONS *****/

    float radius() const;
    float bodyHeading() const;
    float pitch() const;
    float headingOffset() const;
    float pitchOffset() const;

```

```
    float roll() const;
    const DtGlobalObjectDesignator& attachId() const;

/***** MUTATOR FUNCTIONS *****/

    void setRadius(float r);
    void setBodyHeading(float h);
    void setPitch(float p);
    void setHeadingOffset(float o);
    void setPitchOffset(float p);
    void setRoll(float r);
    void setAttachId(const DtGlobalObjectDesignator& id);

    void setFromString(const char *str);
    char *string() const;
    void printData() const;

protected:
    DtGlobalObjectDesignator theId;
};

class DT_DLL_VL DtMimicVCPdu : public DtViewControlPdu
{
public:

/***** STANDARD CONSTRUCTORS *****/

    DtMimicVCPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtMimicVCPdu(const DtNetViewControlPdu *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

/***** INSPECTOR FUNCTIONS *****/

    DtConstVector offset() const;
    const DtTaitBryan &bodyOrientation() const;
    const DtGlobalObjectDesignator& attachId() const;

/***** MUTATOR FUNCTIONS *****/

    void setOffset(DtConstVector offset);
    void setBodyOrientation(const DtTaitBryan &ori);
    void setAttachId(const DtGlobalObjectDesignator& id);

    void setFromString(const char *str);
    char *string() const;
    void printData() const;

protected:
    DtGlobalObjectDesignator theId;
    DtVector theVec;
```



```

    DtTaitBryan theAngles;
};

class DT_DLL_VL DtOrbitVCPdu : public DtViewControlPdu
{
public:

/***** STANDARD CONSTRUCTORS *****/

    DtOrbitVCPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtOrbitVCPdu(const DtNetViewControlPdu *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

/***** INSPECTOR FUNCTIONS *****/

    float radius() const;
    float azimuthRate() const;
    float heading() const;
    float pitch() const;
    float headingOffset() const;
    float pitchOffset() const;
    float roll() const;
    const DtGlobalObjectDesignator& attachId() const;

/***** MUTATOR FUNCTIONS *****/

    void setRadius(float r);
    void setAzimuthRate(float a);
    void setHeading(float h);
    void setPitch(float p);
    void setHeadingOffset(float o);
    void setPitchOffset(float p);
    void setRoll(float r);
    void setAttachId(const DtGlobalObjectDesignator& id);

    void setFromString(const char *str);
    char *string() const;
    void printData() const;

protected:
    DtGlobalObjectDesignator theId;
};

class DT_DLL_VL DtTrackVCPdu : public DtViewControlPdu
{
public:

/***** STANDARD CONSTRUCTORS *****/

    DtTrackVCPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

```

```
DtTrackVCPdu(const DtNetViewControlPdu *initial,
             DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

/***** INSPECTOR FUNCTIONS *****/

DtConstVector worldLocation() const;
const DtGlobalObjectDesignator& trackId() const;

/***** MUTATOR FUNCTIONS *****/

void setWorldLocation(DtConstVector loc);
void setTrackId(const DtGlobalObjectDesignator& id);

void setFromString(const char *str);
char *string() const;
void printData() const;

protected:
    DtVector theVec;
    DtGlobalObjectDesignator theId;
};

class DT_DLL_VL DtTetherTrackVCPdu : public DtViewControlPdu
{
public:

/***** STANDARD CONSTRUCTORS *****/

    DtTetherTrackVCPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtTetherTrackVCPdu(const DtNetViewControlPdu *initial,
                      DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

/***** INSPECTOR FUNCTIONS *****/

    DtConstVector offset() const;
    const DtGlobalObjectDesignator& attachId() const;
    const DtGlobalObjectDesignator& trackId() const;

/***** MUTATOR FUNCTIONS *****/

    void setOffset(DtConstVector loc);
    void setAttachId(const DtGlobalObjectDesignator& id);
    void setTrackId(const DtGlobalObjectDesignator& id);

    void setFromString(const char *str);
    char *string() const;
    void printData() const;

protected:
    DtVector theVec;
```

```

    DtGlobalObjectDesignator theId;
};

class DT_DLL_VL DtMimicTrackVCPdu : public DtViewControlPdu
{
public:

/***** STANDARD CONSTRUCTORS *****/

    DtMimicTrackVCPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtMimicTrackVCPdu(const DtNetViewControlPdu *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

/***** INSPECTOR FUNCTIONS *****/

    DtConstVector offset() const;
    const DtGlobalObjectDesignator& attachId() const;
    const DtGlobalObjectDesignator& trackId() const;

/***** MUTATOR FUNCTIONS *****/

    void setOffset(DtConstVector loc);
    void setAttachId(const DtGlobalObjectDesignator& id);
    void setTrackId(const DtGlobalObjectDesignator& id);

    void setFromString(const char *str);
    char *string() const;
    void printData() const;

protected:
    DtVector theVec;
    DtGlobalObjectDesignator theId;
};

class DT_DLL_VL DtGroupVCPdu : public DtViewControlPdu
{
public:

/***** STANDARD CONSTRUCTORS *****/

    DtGroupVCPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtGroupVCPdu(const DtNetViewControlPdu *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

/***** INSPECTOR FUNCTIONS *****/

    float radialOffset() const;
    float heading() const;
    float pitch() const;
    float headingOffset() const;

```

```
float pitchOffset() const;
int groupCmd() const;
int numEntitiesToRemove() const;
int numEntitiesToAdd() const;
const DtGlobalObjectDesignator& entityToRemove(int n) const;
const DtGlobalObjectDesignator& entityToAdd(int n) const;

/***** MUTATOR FUNCTIONS *****/

void setRadialOffset(float o);
void setHeading(float h);
void setPitch(float p);
void setHeadingOffset(float o);
void setPitchOffset(float o);
void setGroupCmd(int c);
void setNumEntitiesToRemove(int n);
void setNumEntitiesToAdd(int n);
void setEntityToRemove(int n, const DtGlobalObjectDesignator&
id);
void setEntityToAdd(int n, const DtGlobalObjectDesignator& id);

void setFromString(const char *str);
char *string() const;
void printData() const;

protected:
    DtGlobalObjectDesignator theId;
};

class DT_DLL_VL DtModelessVCPdu : public DtViewControlPdu
{
public:

/***** STANDARD CONSTRUCTORS *****/

    DtModelessVCPdu(DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
    DtModelessVCPdu(const DtNetViewControlPdu *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);

/***** INSPECTOR FUNCTIONS *****/

    const DtGlobalObjectDesignator& primary() const;
    const DtGlobalObjectDesignator& secondary() const;
    int numEntitiesToRemove() const;
    int numEntitiesToAdd() const;
    const DtGlobalObjectDesignator& entityToRemove(int n) const;
    const DtGlobalObjectDesignator& entityToAdd(int n) const;

/***** MUTATOR FUNCTIONS *****/
```

```

    void setPrimary(const DtGlobalObjectDesignator& id);
    void setSecondary(const DtGlobalObjectDesignator& id);
    void setNumEntitiesToRemove(int n);
    void setNumEntitiesToAdd(int n);
    void setEntityToRemove(int n, const DtGlobalObjectDesignator&
id);
    void setEntityToAdd(int n, const DtGlobalObjectDesignator& id);

    void printData() const;

protected:
    DtGlobalObjectDesignator theId;
};

class DT_DLL_VL DtUnknownVCPdu : public DtViewControlPdu
{
public:

    /***** STANDARD CONSTRUCTORS *****/

    DtUnknownVCPdu(const DtNetViewControlPdu *initial,
        DtBufferPtr buffer = DtUSE_INTERNAL_BUFFER);
};

#define DtIGNORE_EAST                (1<<0)
#define DtIGNORE_NORTH               (1<<1)
#define DtIGNORE_HEIGHT              (1<<2)
#define DtIGNORE_HEADING             (1<<3)
#define DtIGNORE_PITCH               (1<<4)
#define DtIGNORE_ROLL                (1<<5)
#define DtIGNORE_EAST_OFFSET         (1<<6)
#define DtIGNORE_NORTH_OFFSET        (1<<7)
#define DtIGNORE_HEIGHT_OFFSET       (1<<8)
#define DtIGNORE_RADIUS              (1<<9)
#define DtIGNORE_BODY_HEADING        (1<<10)
#define DtIGNORE_HEADING_OFFSET      (1<<11)
#define DtIGNORE_PITCH_OFFSET        (1<<12)
#define DtIGNORE_FORWARD             (1<<13)
#define DtIGNORE_RIGHT               (1<<14)
#define DtIGNORE_UP                  (1<<15)
#define DtIGNORE_BODY_PITCH          (1<<16)
#define DtIGNORE_BODY_ROLL           (1<<17)
#define DtIGNORE_AZIMUTH_RATE        (1<<18)
#define DtIGNORE_RADIAL_OFFSET       (1<<19)
#define DtIGNORE_PRIMARY_ID          (1<<20)
#define DtIGNORE_SECONDARY_ID        (1<<21)
#define DtIGNORE_GROUP_IDS           (1<<22)

#endif // DtViewControlPdu_H_

```

vList.h

```
#ifndef DtList_H_
#define DtList_H_

#include <stddef.h>
#include "MachineTypes.h"

// DtListItem is an envelope class used by DtList.  When you add an
// element to a DtList, it creates a DtListItem to contain the
// element.
// The DtListItem stores the next and previous pointer, as well as
// the
// data element itself.

class DT_DLL_VLUTIL DtListItem
{
public:

    friend class DtList;

public:

    // Destructor
    virtual ~DtListItem();

    // Return previous (next) item.  If you hit the beginning (end)
    // of the list, returns NULL.
    DtListItem* prev() const;
    DtListItem* next() const;

    // Returns the data element contained in the DtListItem envelope.
    void* data() const;

protected:

    // Constructor - called only by DtList
    DtListItem(void* d);

protected:

    DtListItem* myPrev;
    DtListItem* myNext;
    void*      myData;
};

// A DtList is a list of elements of arbitrary type.  Data elements
// are
```

```

// added to, and extracted from a DtList as void*. DtList encloses
// each
// element in a DtListItem object. You can iterate through the
// elements
// in a DtList like this:
// for (DtListItem* item = list.first(); item; item = item->next())
// {
//     void* dataElement = item->data();
// }

class DT_DLL_VLUTIL DtList
{
public:
    // Constructor
    DtList();

    // Destructor - does NOT free memory of data associated
    // with the items. It only frees the items.
    virtual ~DtList();

    // Copy constructor. Shallow copy. Copies the pointers you've
    // added
    // to the list, but not the items that are pointed to.
    DtList(const DtList& orig);

    // Assignment operator. Shallow copy. Copies the pointers
    // you've added
    // to the list, but not the items that are pointed to. Old list
    // items
    // are removed from the list, and the elements they contain are
    // not deleted.
    DtList& operator=(const DtList& orig);

    // Return first and last item on the list. DtListItem is an envelope
    // for your data. The data can be obtained using DtListItem::data().
    DtListItem* first() const;
    DtListItem* last() const;

    // Returns the number of elements in the list.
    unsigned count() const;

    // Add element to the end of the list. Returns the DtListItem
    // envelope
    // object that contains your new element.
    DtListItem* add(void* data);

```

```
// Add element to the beginning of the list. Returns the DtListItem
// envelope object that contains your new element.
DtListItem* addToStart(void* data);

// Add element to the end of the list. Returns the DtListItem
envelope
// object that contains your new element.
DtListItem* addToEnd(void* data);

// Add element to the list just before the indicated item.
// Returns the DtListItem envelope object that contains your new
element.
DtListItem* addBefore (void* data, DtListItem* before);

// Add element to the list just after the indicated item.
// Returns the DtListItem envelope object that contains your new
element.
DtListItem* addAfter(void* data, DtListItem* after);

// Remove an element from the list. Item is the envelope
containing the
// data element you would like to remove.
// Note: remove does not free memory of data associated
// with the item. It only frees the item. The data is
// returned to the caller for deletion if required.
void* remove(DtListItem* item);

// Calls remove for all elements in the list. It does NOT delete
the
// data in the list. Merely removes the data from the list.
void removeAll();

// Returns the DtListItem envelope object containing the indi-
cated
// data element.
DtListItem* itemLookup(void* data) const;

protected:

// Add item to the list after after.
virtual void addToList(DtListItem* item, DtListItem* after);

// Remove item from the list. Does NOT delete the data contained
in
// item.
virtual void removeFromList(DtListItem* item);

protected:
```



```

    DtListItem* myFirst;
    DtListItem* myLast;
    unsigned    myCount;
};

// An intrusive list is a list that assumes that the elements that
// you add are pointers to instances of classes derived from
// DtListedObject. When an intrusive list is use, you can get
previous
// and next pointers directly from the listed elements, (through the
base
// DtListedObject class), without having to go through the DtList
itself.

class DT_DLL_VLUTIL DtIntrusiveList : public DtList
{
public:

    // Override of virtual function. Saves pointer to the list and to
    // the item in the DtListedObj.
    virtual void addToList(DtListItem* item, DtListItem* after);

    // Override of virtual function. Removes the item from this
list,
    // and cleans up the DtListedObj's item and list pointers.
    virtual void removeFromList(DtListItem* item);
};

// This is a base class from which elements of a DtIntrusiveList
must
// be derived. Previous and next pointers can be obtained directly
from
// the DtListedObj, without going through the DtList to which it has
been
// added.

class DT_DLL_VLUTIL DtListedObj
{
public:

    friend class DtIntrusiveList;

public:

    // Constructor
    DtListedObj();

    // Destructor
    virtual ~DtListedObj();

```

```
// Return next (previous) elements on the list, or NULL if hit
// the end (beginning) of the list.
void* next() const;
void* prev() const;

// Return next (previous) elements on the list, but wrap to the
beginning
// (end) of the list when you hit the end (beginning). Only
returns
// NULL when the list is totally empty.
void* wrapNext() const;
void* wrapPrev() const;

protected:

    DtList* myList;
    DtListItem* myItem;
};

// Inline Functions

inline DtListItem* DtListItem::prev() const
{
    return myPrev;
}

inline DtListItem* DtListItem::next() const
{
    return myNext;
}

inline void* DtListItem::data() const
{
    return myData;
}

inline DtListItem* DtList::first() const
{
    return myFirst;
}

inline DtListItem* DtList::last() const
{
    return myLast;
}

inline unsigned DtList::count() const
{
    return myCount;
}
```

```
inline DtListItem* DtList::add(void* data)
{
    return addAfter(data, myLast);
}

inline DtListItem* DtList::addToStart(void* data)
{
    return addAfter(data, NULL);
}

inline DtListItem* DtList::addToEnd(void* data)
{
    return addAfter(data, myLast);
}

inline DtListItem* DtList::addBefore (void* data, DtListItem*
before)
{
    return addAfter(data, before ? before->myPrev : (DtListItem*)
NULL);
}

inline void* DtListedObj::next() const
{
    return (myItem && myItem->next()) ? myItem->next()->data() :
NULL;
}

inline void* DtListedObj::prev() const
{
    return (myItem && myItem->prev()) ? myItem->prev()->data() :
NULL;
}

inline void* DtListedObj::wrapNext() const
{
    return next() ? next() :
        ((myList && myList->first()) ? myList->first()->data() : NULL);
}

inline void* DtListedObj::wrapPrev() const
{
    return prev() ? prev() :
        ((myList && myList->last()) ? myList->last()->data() : NULL);
}

// Macro to loop through the list.
#define looplist(theData, list) \
for (DtListItem* item=(list)->first(); \
```

```
    theData = (item ? item->data() : NULL); \
    item = item->next())

// typed looplist for cfront (ANSI C and GCC don't require casting
// from void *)
#define looptlist(theData, type, list) \
for (DtListItem* item=(list)->first(); \
    theData = (type)(item ? item->data() : NULL); \
    item = item->next())

// looptlist for multiple loops in same scope with C++ without
// proper scope of loop vars
#define looptlist2(theData, type, list) \
for (item=(list)->first(); \
    theData = (type)(item ? item->data() : NULL); \
    item = item->next())

#endif // DtList_H_
```

vlprint.h

```

#include "MachineTypes.h"
/* vlprint.h
 * These functions provide a simple error and print facility for
 * vlrink
 */

/*
 * DtNotifyLevel is used to control the level of diagnostics to one
 * of
 * the following levels. Each level includes messages for all lower
 * levels as well. The default level is DtNlInfo, i.e. all messages
 * except debug statements.
 */

/* Notification Levels */
#define DtNlFatal 0
#define DtNlWarn 1
#define DtNlInfo 2
#define DtNlVerbose 3

extern DT_DLL_VLUTIL int DtNotifyLevel; /* default is DtNlInfo */

/* These functions print messages using stdio depending on the
 * notify level.
 * Fatal and Warn print to stderr.
 * Info prints to stdout.
 * Fatal funcs call DtAbort at end, even if (notifyLevel <
DtNlFatal)
 * Fatal prints are preceded by "***FATAL ERROR**": "
 * the DtFatalError print is followed by \n
 * Perror prints are followed by a call to perror()
 */
DT_DLL_VLUTIL void DtFatalError(const char *s, ...);
DT_DLL_VLUTIL void DtFatalPerror(const char *s, ...);
DT_DLL_VLUTIL void DtWarnPerror(const char *s, ...);
DT_DLL_VLUTIL void DtFatalNetPerror(const char *s, ...);
DT_DLL_VLUTIL void DtWarnNetPerror(const char *s, ...);
DT_DLL_VLUTIL void DtWarn(const char *s, ...);
DT_DLL_VLUTIL void DtInfo(const char *s, ...);
DT_DLL_VLUTIL void DtVerbose(const char *s, ...);
/* DtAbort() exits the program.
 * The user can provide for different behaviour [e.g. calling
 * abort()]
 * by linking with a another definition DtAbort [i.e. It is in its
 * own
 * object file.], but the user's DtAbort() should not return.
 */

```

```
DT_DLL_VLUTIL void DtAbort(void);
```

```
#if WIN32
```

```
DT_DLL_VLUTIL void setFileOutput(const char *filename, int con=0,  
int son=0);
```

```
#endif
```

vTime.h

```
#include "MachineTypes.h"

typedef DtReal DtTime;
const double DtHour2Sec = 3600;
const double DtSec2Hour = 1./3600;

// Initialize all VR-Link time functionality
// This must be called before any other time-related functions
DT_DLL_VLUTIL void DtTimeInit(void);

// Simulation Time Calls

// Returns the last value passed to DtSetSimTime - the current value
// of VR-Link simulation time.
DT_DLL_VLUTIL DtTime DtGetSimTime(void);

// Set the current value of VR-Link simulation time. Should be
// called once per simulation frame to let VR-Link know the value
// of simulation time for the frame.
DT_DLL_VLUTIL void DtSetSimTime(DtTime NewSimTime);

// Abort if DtSetSimTime was never called
DT_DLL_VLUTIL void DtIsSimTimeSet(void);

// Return real seconds elapsed since DtTimeInit was called
DT_DLL_VLUTIL DtTime DtGetElapsedRealTime(void);

// Obsolete. This function exists for backwards compatibility with
// older versions of VR-Link. Use DtGetElapsedRealTime instead.
DT_DLL_VLUTIL DtTime DtTimeNow(void);

// Returns the value of DtGetElapsedRealTime at the time that
// DtSetSimTime was last invoked.
DT_DLL_VLUTIL DtTime DtGetApproximateElapsedRealTime(void);

// Return current wallclock time - seconds since Jan 1, 1970
DT_DLL_VLUTIL DtTime DtAbsRealTime(void);

extern int DtUseSimTimeForApproxRealTime;
```

vlutil.h

```
#include "vlconfig.h"

#ifdef WIN32
#include <winsock.h>
#endif

#ifdef DT_HAVE_UNISTD_H
#include <unistd.h>
#endif

#ifdef PCFTP
#include <4bsddefs.h>
#endif

extern "C" {
#include <stdlib.h>
#include <sys/types.h>
}
#include "vlTime.h"
#include "math.h"

#define DtDELETE(x) \
    if (x) \
    { \
        delete x; \
        x = NULL; \
    }

#ifndef DtISON
#define DtISON(a, x) (a & x)
#define DtISOFF(a, x) (!DtISON(a, x))
#define DtTURNON(a, x) (a |= x)
#define DtTURNOFF(a, x) (a &= ~x)
#endif

#define DtMIN(x, y) ((x)<(y) ? (x) : (y))
#define DtMAX(x, y) ((x)>(y) ? (x) : (y))
#define DtLIMIT(low, x, hi) DtMAX(DtMIN(x, hi), low)
#define DtABS(x) ((x) < 0 ? -(x) : (x))
#define DtSQR(x) ((x)*(x))

#define DtNUMBER(foo) (sizeof(foo)/sizeof(foo[0]))
#define DtOFFSET(type, field) ((size_t)&((type *)NULL)->field)
#define DtSIZEOF(type, field) sizeof(((type *)NULL)->field)

// Rounds num up to a multiple of mult.
/* some versions of strtol returns INT_MAX for input larger than
```



```

    0x7fffffff. As long as strtoul accepts and properly processes
    negative input, this macro will work for both unsigned and signed
    ints (as other strtol do). */

#ifdef DT_HAVE_STRTOUL
#define Dtatoi(s) ((int)(strtoul(s, 0, 0)))
#else
#define Dtatoi(s) (int)strtol(s, 0, 0)
#endif

DT_DLL_VLUTIL char *DtDeg2DmsString(double val, char *buff, int
prec);
DT_DLL_VLUTIL char *DtDeg2DmsString(double val);

// Returns -1 if num is not in lst. Otherwise, returns the index
// in the list at which num is first found. lst must be a zero-
// terminated
// array of integers.
DT_DLL_VLUTIL int DtIsInList(const int *lst, int num);

#ifdef DT_FD_ZERO_USES_MEMSET
#include <string.h>
#else /* need bzero */

#ifdef DT_HAVE_BZERO_PROTO_IN_MEMORY_H
#include <memory.h>
#elif DT_HAVE_BZERO_PROTO_IN_CC_LIBC_H
#include <CC/libc.h>
#elif DT_HAVE_BZERO_PROTO_IN_BSTRING_H
#include <bstring.h>
#else
extern "C" { void bzero(char *b, int length); }
#endif

#endif /* DT_FD_ZERO_USES_MEMSET */

#ifdef __SUNPRO_CC
#include <sys/file.h>
#endif

DT_DLL_VLUTIL int DtSelect(int n, fd_set *i, fd_set *o, fd_set *e,
DtTime t);

#ifdef WIN32
#else
#endif

class DT_DLL_VLUTIL DtSelectParam
{
public:

```

```
DtSelectParam() {
    FD_ZERO(&read0);
    read = read0;
    maxfd = -1;
}
int select(DtTime timeout) {
    read = read0;
    return DtSelect(maxfd+1, &read, NULL, NULL, timeout);
}
void addRead(int fd) {
    FD_SET(fd, &read0);
    maxfd = DtMAX(maxfd, fd);
}
void delRead(int fd) {
    FD_CLR((unsigned int) fd, &read0);
    /* should recalc maxfd */
}
int canRead(int fd) {
    return ((int)(FD_ISSET(fd, &read)));
}
private:
    int maxfd;
    fd_set read0, read;
};

// Strdup wrapper. Returns NULL if str is NULL.
// Otherwise returns a newed char* containing a copy of str.
// Caller deletes using delete[].
DT_DLL_VLUTIL char* DtStrdup(const char* str);

// Convert str to all lowercase.
DT_DLL_VLUTIL char* DtToLower(char* str);

// Convert str to all uppercase.
DT_DLL_VLUTIL char* DtToUpper(char* str);

/* ----- inet utils ----- */

typedef DtU32 DtInetAddr;

const DtInetAddr DtInetAddrAny = 0;
const DtInetAddr loopbackAddr = 0x7f000001; /* 127.0.0.1 */
/* converts ulong to internet dot notation (returns internal static
buffer) */
DT_DLL_VLUTIL const char *DtInetAddrString(DtInetAddr addr);

/* converts to internet dot notation to ulong in host order */
#if _PowerUX
DT_DLL_VLUTIL DtInetAddr DtStringToInetAddr(char *str);
#else
```

```
DT_DLL_VLUTIL DtInetAddr DtStringToInetAddr(const char *str);
#endif

DT_DLL_VLUTIL DtInetAddr DtInetBroadcastOfDevice(const char
*device);
DT_DLL_VLUTIL DtInetAddr DtInetAddrOfDevice(const char *device);
DT_DLL_VLUTIL DtInetAddr DtNetMaskOfDevice(const char *device);
DT_DLL_VLUTIL DtInetAddr DtFirstHostInetAddr();
DT_DLL_VLUTIL int DtIsMulticastAddr(DtInetAddr addr);
```

v/Vector.h

```
#include "MachineTypes.h"

enum DtVectorOffset
{
    DtX,
    DtY,
    DtZ
};

typedef const DtVector &DtConstVector;
typedef DtVector &DtVectorRef;

// class DtVector:
//
// No virtual functions, because we want the size of a DtVector
// to 3 * 8 bytes - so that you can make the same assumptions about it
// that we used to make about its size and layout. This is espe-
// cially
// important in DtDcm, where we use an array of 3 DtVectors as the
// myRep.
// If DtVector had any virtual functions, there would be 4 bytes
// between
// vector reps.
class DT_DLL_MATRIX DtVector
{
public:

    // default constructor
    DtVector();

    // construct from three doubles.
    DtVector(double x, double y, double z);

    // destructor
    ~DtVector();

    // copy constructor
    DtVector(const DtVector& vec);

    // assignment operator
    DtVector& operator=(const DtVector& vec);

    // Convert the vector to a pointer to double.
    operator double* ();
    operator const double* () const;

    // Get the value for the specified subscript. (int version)
```

```
double& operator[](int ind);
double operator[](int ind) const;

// Get the value for the specified subscript. (enum version)
double& operator[](DtVectorOffset ind);
double operator[](DtVectorOffset ind) const;

// Get whether another vector equals this vector.
int operator==(const DtVector& vec) const;

// Get a string representation of the vector.
const char* string() const;

public:

    // Get a zero'ed vector.
    static DtConstVector zero();

    // Get a (1,0,0) vector.
    static DtConstVector i();

    // Get a (0,1,0) vector.
    static DtConstVector j();

    // Get a (0,0,1) vector.
    static DtConstVector k();

    // Get a vector set to all ones.
    static DtConstVector ones();

protected:
    double myRep[3];
};
// Inline DtVector functions
```

vrlFedAmb.h

```
#if DtHLA

#if DtRTI103
#include "vFedAmb103.h"
#else

#include "RTI.hh"
#include "nullFedAmb.h"
#include "callbacks.h"

#include "chooseDefines.h"

// class DtVrlFederateAmbassador:
//
// Instances of DtVrlFederateAmbassador are used to manage the
// interface between the RTI and the VR-Link Federate.

class DT_DLL_VL DtVrlFederateAmbassador : public DtNullFederateAm-
bassador
{
public:

    // default constructor
    DtVrlFederateAmbassador();

    // destructor
    virtual ~DtVrlFederateAmbassador();

    // Initialize the ambassador.
    // Needs to be called after construction.
    virtual void init(DtReflectedHlaObjectManager* reflectedObjMgr,
        DtLocalHlaObjectManager* localObjMgr,
        DtInteractionManager* interMgr);

    // Add/Remove (register/deregister) function to be called when
    // the RTI's reflectAttributeValues service is invoked.
    virtual void addReflectAttributeValuesCb(RTI::ObjectHandle id,
        DtReflectAttributeValuesCb cb,
        void* usr);
    virtual void removeReflectAttributeValuesCb(RTI::ObjectHandle id,
        DtReflectAttributeValuesCb cb,
        void* usr);

    // Add/Remove (register/deregister) function to be called when
    // the RTI's discoverObject service is invoked.
    virtual void addDiscoverObjectCb(RTI::ObjectClassHandle hand,
        DtDiscoverObjectCb cb,
```

```

    void* usr);
virtual void removeDiscoverObjectCb(RTI::ObjectClassHandle hand,
    DtDiscoverObjectCb cb,
    void* usr);

// Add/Remove (register/deregister) function to be called when
// the RTI's removeObject service is invoked.
virtual void addRemoveObjectCb(RTI::ObjectHandle id,
    DtRemoveObjectCb, void* usr);
virtual void removeRemoveObjectCb(RTI::ObjectHandle id,
    DtRemoveObjectCb, void* usr);

// Add/Remove (register/deregister) function to be called when
// the RTI's startRegistration service is invoked.
virtual void addStartRegCb(RTI::ObjectClassHandle hand,
    DtStartRegCb, void* usr);
virtual void removeStartRegCb(RTI::ObjectClassHandle hand,
    DtStartRegCb, void* usr);

// Add/Remove (register/deregister) function to be called when
// the RTI's stopReg service is invoked.
virtual void addStopRegCb(RTI::ObjectClassHandle hand,
    DtStopRegCb, void* usr);
virtual void removeStopRegCb(RTI::ObjectClassHandle hand,
    DtStopRegCb, void* usr);

// Add/Remove (register/deregister) function to be called when
// the RTI's turnUpdatesOn service is invoked.
virtual void addUpdatesOnCb(RTI::ObjectHandle hand,
    DtUpdatesOnCb, void* usr);
virtual void removeUpdatesOnCb(RTI::ObjectHandle hand,
    DtUpdatesOnCb, void* usr);

// Add/Remove (register/deregister) function to be called when
// the RTI's turnUpdatesOff service is invoked.
virtual void addUpdatesOffCb(RTI::ObjectHandle hand,
    DtUpdatesOffCb, void* usr);
virtual void removeUpdatesOffCb(RTI::ObjectHandle hand,
    DtUpdatesOffCb, void* usr);

// Add/Remove (register/deregister) function to be called when
// the RTI's startInteractionGeneration service is invoked.
virtual void addStartInteractionsCb(RTI::InteractionClassHandle
hand,
    DtStartStopInterCb, void* usr);
    virtual void removeStartInteractionsCb(RTI::InteractionClass-
Handle hand,
    DtStartStopInterCb, void* usr);

// Add/Remove (register/deregister) function to be called when

```

```
// the RTI's stopInteractionGeneration service is invoked.
virtual void addStopInteractionsCb(RTI::InteractionClassHandle
hand,
    DtStartStopInterCb, void* usr);
virtual void removeStopInteractionsCb(RTI::InteractionClassHandle
hand,
    DtStartStopInterCb, void* usr);

// Add/Remove (register/deregister) function to be called when
// the RTI's receiveInteraction service is invoked.
virtual void addReceiveInteractionCb(RTI::InteractionClassHandle
hand,
    DtReceiveInteractionCb cb, void* usr);
virtual void removeReceiveInteractionCb(RTI::InteractionClass-
Handle hand,
    DtReceiveInteractionCb cb, void* usr);

protected:

// copy constructor - not implemented
DtVrlFederateAmbassador(const DtVrlFederateAmbassador& orig);

// assignment operator - not implemented
DtVrlFederateAmbassador& operator=(const DtVrlFederateAmbassador&
orig);

virtual void startRegistrationForObjectClass (
    RTI::ObjectClassHandle theClass)
    throw (
        RTI::ObjectClassNotPublished,
        RTI::FederateInternalError);

virtual void stopRegistrationForObjectClass (
    RTI::ObjectClassHandle theClass)
    throw (
        RTI::ObjectClassNotPublished,
        RTI::FederateInternalError);

virtual void turnInteractionsOn (
    RTI::InteractionClassHandle theHandle)
    throw (
        RTI::InteractionClassNotPublished,
        RTI::FederateInternalError);

virtual void turnInteractionsOff (
    RTI::InteractionClassHandle theHandle)
    throw (
        RTI::InteractionClassNotPublished,
        RTI::FederateInternalError);
```



```

virtual void discoverObjectInstance (
    RTI::ObjectHandle theObject,
    RTI::ObjectClassHandle theObjectClass,
    const char* theObjectName)
    throw (
        RTI::CouldNotDiscover,
        RTI::ObjectClassNotKnown,
        RTI::FederateInternalError);

virtual void reflectAttributeValues (
    RTI::ObjectHandle theObject,
    const RTI::AttributeHandleValuePairSet& theAttributes,
    const RTI::FedTime& theTime,
    const char* theTag,
    RTI::EventRetractionHandle theHandle)
    throw (
        RTI::ObjectNotKnown,
        RTI::AttributeNotKnown,
        RTI::FederateOwnsAttributes,
        RTI::InvalidFederationTime,
        RTI::FederateInternalError);

virtual void reflectAttributeValues (
    RTI::ObjectHandle theObject,
    const RTI::AttributeHandleValuePairSet& theAttributes,
    const char* theTag)
    throw (
        RTI::ObjectNotKnown,
        RTI::AttributeNotKnown,
        RTI::FederateOwnsAttributes,
        RTI::FederateInternalError);

virtual void receiveInteraction (
    RTI::InteractionClassHandle theInteraction,
    const RTI::ParameterHandleValuePairSet& theParameters,
    const RTI::FedTime& theTime,
    const char* theTag,
    RTI::EventRetractionHandle theHandle)
    throw (
        RTI::InteractionClassNotKnown,
        RTI::InteractionParameterNotKnown,
        RTI::InvalidFederationTime,
        RTI::FederateInternalError);

virtual void receiveInteraction (
    RTI::InteractionClassHandle theInteraction,
    const RTI::ParameterHandleValuePairSet& theParameters,
    const char* theTag)
    throw (
        RTI::InteractionClassNotKnown,

```

```
        RTI::InteractionParameterNotKnown,
        RTI::FederateInternalError);

virtual void removeObjectInstance (
    RTI::ObjectHandle theObject,
    const RTI::FedTime& theTime,
    const char* theTag,
    RTI::EventRetractionHandle theHandle)
    throw (
        RTI::ObjectNotKnown,
        RTI::InvalidFederationTime,
        RTI::FederateInternalError);

virtual void removeObjectInstance (
    RTI::ObjectHandle theObject,
    const char* theTag)
    throw (
        RTI::ObjectNotKnown,
        RTI::FederateInternalError);

virtual void provideAttributeValueUpdate (
    RTI::ObjectHandle theObject,
    const RTI::AttributeHandleSet& theAttributes)
    throw (
        RTI::ObjectNotKnown,
        RTI::AttributeNotKnown,
        RTI::AttributeNotOwned,
        RTI::FederateInternalError);

virtual void turnUpdatesOnForObjectInstance (
    RTI::ObjectHandle theObject,
    const RTI::AttributeHandleSet& theAttributes)
    throw (
        RTI::ObjectNotKnown,
        RTI::AttributeNotOwned,
        RTI::FederateInternalError);

virtual void turnUpdatesOffForObjectInstance (
    RTI::ObjectHandle theObject,
    const RTI::AttributeHandleSet& theAttributes)
    throw (
        RTI::ObjectNotKnown,
        RTI::AttributeNotOwned,
        RTI::FederateInternalError);

public:

    // returns a new DtVrIFedAmbassador
    static DtVrIFederateAmbassador* create();
```

protected:

```
DtInteractionManager* myInteractionManager;  
DtReflectedHlaObjectManager* myReflectedObjectManager;  
DtLocalHlaObjectManager* myLocalObjectManager;  
  
DtReflectAttributeValuesCbList myRavCbList;  
DtUpdatesOnCbList myUpdatesOnCbList;  
DtUpdatesOffCbList myUpdatesOffCbList;  
DtStartRegCbList myStartRegCbList;  
DtStopRegCbList myStopRegCbList;  
DtStartStopInterCbList myStartIntCbList;  
DtStartStopInterCbList myStopIntCbList;  
};  
#endif  
#endif
```

vrlFedTime.h

```
// This file is based on MyFedTime.h, which appears in the DMSO RTI
// MyFedTime example in RTI_HOME/lang/C++/demo/MyFedTime.
```

```
#ifndef DtVrlFedTime_H_
#define DtVrlFedTime_H_
```

```
#if DtHLA
```

```
#include <RTI.hh>
#include <sys/types.h>
#include <string.h>
#include "MachineTypes.h"
```

```
class DT_DLL_VL DtVrlFedTime : public RTI::FedTime {
```

```
//-----
// Constructors and Destructors
```

```
//-----
public:
```

```
    DtVrlFedTime();
    DtVrlFedTime(const double&);
    DtVrlFedTime(const DtVrlFedTime&);
    virtual ~DtVrlFedTime();
```

```
//-----
// Overloaded functions from RTI::FedTime
```

```
//-----
public:
```

```
    virtual void          setZero();
    virtual RTI::Boolean   isZero();
    virtual void          setEpsilon();
    virtual void          setPositiveInfinity();
    virtual RTI::Boolean   isPositiveInfinity();
    virtual int            encodedLength() const;
    virtual void          encode(char *buff) const;
    virtual int            getPrintableLength() const;
    virtual void          getPrintableString(char*);
```

```
//-----
// Overloaded operators from RTI::FedTime
```

```
//-----
public:
```

```
    virtual RTI::FedTime& operator+= (const RTI::FedTime&)
        throw (RTI::InvalidFederationTime);
```

```
    virtual RTI::FedTime& operator-= (const RTI::FedTime&)
        throw (RTI::InvalidFederationTime);
```

```

virtual RTI::Boolean operator<= (const RTI::FedTime&) const
    throw (RTI::InvalidFederationTime);

virtual RTI::Boolean operator< (const RTI::FedTime&) const
    throw (RTI::InvalidFederationTime);

virtual RTI::Boolean operator>= (const RTI::FedTime&) const
    throw (RTI::InvalidFederationTime);

virtual RTI::Boolean operator> (const RTI::FedTime&) const
    throw (RTI::InvalidFederationTime);

virtual RTI::Boolean operator== (const RTI::FedTime&) const
    throw (RTI::InvalidFederationTime);

virtual RTI::FedTime& operator= (const RTI::FedTime&)
    throw (RTI::InvalidFederationTime);

//-----
// Implementation functions
//-----
public:
    virtual double          getTime() const;

//-----
// Implementation operators
//-----
virtual RTI::Boolean operator== (const double&) const
    throw (RTI::InvalidFederationTime);

virtual RTI::Boolean operator!= (const RTI::FedTime&) const
    throw (RTI::InvalidFederationTime);

virtual RTI::Boolean operator!= (const double&) const
    throw (RTI::InvalidFederationTime);

virtual RTI::FedTime& operator= (const DtVrlFedTime&)
    throw (RTI::InvalidFederationTime);

virtual RTI::FedTime& operator= (const double&)
    throw (RTI::InvalidFederationTime);

virtual RTI::FedTime& operator*= (const RTI::FedTime&)
    throw (RTI::InvalidFederationTime);

virtual RTI::FedTime& operator/= (const RTI::FedTime&)
    throw (RTI::InvalidFederationTime);

virtual RTI::FedTime& operator+= (const double&)
    throw (RTI::InvalidFederationTime);

```

```
virtual RTI::FedTime& operator-= (const double&)
    throw (RTI::InvalidFederationTime);

virtual RTI::FedTime& operator*= (const double&)
    throw (RTI::InvalidFederationTime);

virtual RTI::FedTime& operator/= (const double&)
    throw (RTI::InvalidFederationTime);

virtual DtVrlFedTime operator+ (const RTI::FedTime&)
    throw (RTI::InvalidFederationTime);

virtual DtVrlFedTime operator+ (const double&)
    throw (RTI::InvalidFederationTime);

virtual DtVrlFedTime operator- (const RTI::FedTime&)
    throw (RTI::InvalidFederationTime);

virtual DtVrlFedTime operator- (const double&)
    throw (RTI::InvalidFederationTime);

virtual DtVrlFedTime operator* (const RTI::FedTime&)
    throw (RTI::InvalidFederationTime);

virtual DtVrlFedTime operator* (const double&)
    throw (RTI::InvalidFederationTime);

virtual DtVrlFedTime operator/ (const RTI::FedTime&)
    throw (RTI::InvalidFederationTime);

virtual DtVrlFedTime operator/ (const double&)
    throw (RTI::InvalidFederationTime);

//-----
// Implementation friends
//-----
public:
    friend ostream& operator<< (ostream&, RTI::FedTime&);

//-----
// Implementation member variables
//-----
private:
    double          _fedTime;
    double          _zero;
    double          _epsilon;
    double          _positiveInfinity;
};
//-----
```

```
// Global operators
//-----

DtVrlFedTime operator+ (const double&, const RTI::FedTime&);
DtVrlFedTime operator- (const double&, const RTI::FedTime&);
DtVrlFedTime operator* (const double&, const RTI::FedTime&);
DtVrlFedTime operator/ (const double&, const RTI::FedTime&);

#endif
```

vrIRtiAmb.h

```
#if DtHLA

#if DtRTI103
#include "vRtiAmb103.h"
#else

#ifndef DtVrIRtiAmbassador_H_
#define DtVrIRtiAmbassador_H_

#include "RTI.hh"
#include "chooseDefines.h"
#include "MachineTypes.h"

// class DtVrIRtiAmbassador:
//
// DtVrIRtiAmbassador is a purely virtual class, designed to provide
// a foundation for different ways to send RTI messages.

class DT_DLL_VL DtVrIRtiAmbassador
{
public:

    // default constructor
    DtVrIRtiAmbassador();

    // destructor
    virtual ~DtVrIRtiAmbassador();

    // copy constructor
    DtVrIRtiAmbassador(const DtVrIRtiAmbassador& orig);

    // assignment operator
    DtVrIRtiAmbassador& operator=(const DtVrIRtiAmbassador& orig);

    // Get the real RTI ambassador, if the implementation of the
    derived
    // object permits. Otherwise returns NULL.
    virtual RTI::RTIambassador* rtiAmb();

    // Set/Get whether to reflect back local objects and interac-
    tions, if
    // supported by the derived class. The set call's return value
    indicates
    // the value of the flag as a result of the call.
    virtual DtBoolean setReflecting(DtBoolean onOff);
    virtual DtBoolean reflecting() const;
```



```
virtual void createFederationExecution (
    const char *executionName,
    const char *FED) = 0;

virtual void destroyFederationExecution (
    const char *executionName) = 0;

virtual RTI::FederateHandle joinFederationExecution (
    const char* yourName,
    const char* executionName,
    RTI::FederateAmbassadorPtr federateAmbassadorReference) = 0;

virtual void resignFederationExecution(
    RTI::ResignAction theAction) = 0;

virtual void registerFederationSynchronizationPoint(
    const char* label,
    const char* theTag) = 0;

virtual void registerFederationSynchronizationPoint(
    const char* label,
    const char* theTag,
    const RTI::FederateHandleSet& syncSet) = 0;

virtual void synchronizationPointAchieved(
    const char* label) = 0;

virtual void requestFederationSave (
    const char* label,
    const RTI::FedTime& theTime) = 0;

virtual void requestFederationSave (
    const char *label) = 0;

virtual void federateSaveBegun() = 0;

virtual void federateSaveComplete() = 0;

virtual void federateSaveNotComplete() = 0;

virtual void requestFederationRestore(const char* label) = 0;

virtual void federateRestoreComplete() = 0;

virtual void federateRestoreNotComplete() = 0;

virtual void publishObjectClass(
    RTI::ObjectClassHandle theClass,
    const RTI::AttributeHandleSet& attributeList) = 0;
```

```
virtual void unpublishObjectClass(
    RTI::ObjectClassHandle theClass) = 0;

virtual void publishInteractionClass(
    RTI::InteractionClassHandle theInteraction) = 0;

virtual void unpublishInteractionClass(
    RTI::InteractionClassHandle theInteraction) = 0;

virtual void subscribeObjectClassAttributes(
    RTI::ObjectClassHandle theClass,
    const RTI::AttributeHandleSet& attributeList,
    RTI::Boolean active = RTI::RTI_TRUE) = 0;

virtual void unsubscribeObjectClass(
    RTI::ObjectClassHandle theClass) = 0;

virtual void subscribeInteractionClass(
    RTI::InteractionClassHandle theClass,
    RTI::Boolean active = RTI::RTI_TRUE) = 0;

virtual void unsubscribeInteractionClass(
    RTI::InteractionClassHandle theClass) = 0;

virtual RTI::ObjectHandle registerObjectInstance(
    RTI::ObjectClassHandle theClass, const char* theObject) = 0;

virtual RTI::ObjectHandle registerObjectInstance(
    RTI::ObjectClassHandle theClass) = 0;

virtual RTI::EventRetractionHandle updateAttributeValues(
    RTI::ObjectHandle theObject,
    const RTI::AttributeHandleValuePairSet& theAttributes,
    const RTI::FedTime& theTime,
    const char* theTag) = 0;

virtual void updateAttributeValues(
    RTI::ObjectHandle theObject,
    const RTI::AttributeHandleValuePairSet& theAttributes,
    const char* theTag) = 0;

virtual RTI::EventRetractionHandle sendInteraction(
    RTI::InteractionClassHandle theInteraction,
    const RTI::ParameterHandleValuePairSet& theParameters,
    const RTI::FedTime& theTime,
    const char* theTag) = 0;

virtual void sendInteraction(
    RTI::InteractionClassHandle theInteraction,
    const RTI::ParameterHandleValuePairSet& theParameters,
```

```
    const char* theTag) = 0;

virtual RTI::EventRetractionHandle deleteObjectInstance(
    RTI::ObjectHandle theObject,
    const RTI::FedTime& theTime,
    const char* theTag) = 0;

virtual void deleteObjectInstance(
    RTI::ObjectHandle theObject,
    const char* theTag) = 0;

virtual void localDeleteObjectInstance(
    RTI::ObjectHandle theObject) = 0;

virtual void changeAttributeTransportationType (
    RTI::ObjectHandle theObject,
    const RTI::AttributeHandleSet& theAttributes,
    RTI::TransportationHandle theType) = 0;

virtual void changeInteractionTransportationType (
    RTI::InteractionClassHandle theClass,
    RTI::TransportationHandle theType) = 0;

virtual void requestObjectAttributeValueUpdate (
    RTI::ObjectHandle theObject,
    const RTI::AttributeHandleSet& theAttributes) = 0;

virtual void requestClassAttributeValueUpdate (
    RTI::ObjectClassHandle theClass,
    const RTI::AttributeHandleSet& theAttributes) = 0;

virtual void unconditionalAttributeOwnershipDivestiture(
    RTI::ObjectHandle theObject,
    const RTI::AttributeHandleSet& theAttributes) = 0;

virtual void negotiatedAttributeOwnershipDivestiture(
    RTI::ObjectHandle theObject,
    const RTI::AttributeHandleSet& theAttributes,
    const char* theTag) = 0;

virtual void attributeOwnershipAcquisition(
    RTI::ObjectHandle theObject,
    const RTI::AttributeHandleSet& desiredAttributes,
    const char* theTag) = 0;

virtual void attributeOwnershipAcquisitionIfAvailable(
    RTI::ObjectHandle theObject,
    const RTI::AttributeHandleSet& desiredAttributes) = 0;
```

```
virtual RTI::AttributeHandleSet* attributeOwnershipReleaseResponse(
    RTI::ObjectHandle theObject,
    const RTI::AttributeHandleSet& theAttributes) = 0;

virtual void cancelNegotiatedAttributeOwnershipDivestiture(
    RTI::ObjectHandle theObject,
    const RTI::AttributeHandleSet& theAttributes) = 0;

virtual void cancelAttributeOwnershipAcquisition(
    RTI::ObjectHandle theObject,
    const RTI::AttributeHandleSet& theAttributes) = 0;

virtual void queryAttributeOwnership(
    RTI::ObjectHandle theObject,
    RTI::AttributeHandle theAttribute) = 0;

virtual RTI::Boolean isAttributeOwnedByFederate(
    RTI::ObjectHandle theObject,
    RTI::AttributeHandle theAttribute) = 0;

virtual void enableTimeRegulation (
    const RTI::FedTime& theFederateTime,
    const RTI::FedTime& theLookahead) = 0;

virtual void disableTimeRegulation() = 0;

virtual void enableTimeConstrained() = 0;

virtual void disableTimeConstrained() = 0;

virtual void timeAdvanceRequest(
    const RTI::FedTime& theTime) = 0;

virtual void timeAdvanceRequestAvailable (
    const RTI::FedTime& theTime) = 0;

virtual void nextEventRequest(
    const RTI::FedTime& theTime) = 0;

virtual void nextEventRequestAvailable (
    const RTI::FedTime& theTime) = 0;

virtual void flushQueueRequest (
    const RTI::FedTime& theTime) = 0;

virtual void enableAsynchronousDelivery() = 0;

virtual void disableAsynchronousDelivery() = 0;
```

```
virtual void queryLBTS(RTI::FedTime& theTime) = 0;

virtual void queryFederateTime(RTI::FedTime& theTime) = 0;

virtual void queryMinNextEventTime (
    RTI::FedTime& theTime) = 0;

virtual void modifyLookahead (
    const RTI::FedTime& theLookahead) = 0;

virtual void queryLookahead (
    RTI::FedTime& theTime) = 0;

virtual void retract(
    RTI::EventRetractionHandle theHandle) = 0;

virtual void changeAttributeOrderType(
    RTI::ObjectHandle theObject,
    const RTI::AttributeHandleSet& theAttributes,
    RTI::OrderingHandle theType) = 0;

virtual void changeInteractionOrderType(
    RTI::InteractionClassHandle theClass,
    RTI::OrderingHandle theType) = 0;

virtual void createRegion (
    RTI::SpaceHandle theSpace,
    RTI::ULong numberOfExtents) = 0;

virtual void notifyAboutRegionModification (
    RTI::Region &theRegion) = 0;

virtual void deleteRegion(RTI::Region *theRegion) = 0;

virtual RTI::ObjectHandle registerObjectInstanceWithRegion(
    RTI::ObjectClassHandle theClass,
    const char* theObject,
    RTI::AttributeHandle theAttributes[],
    RTI::Region* theRegions[],
    RTI::ULong theNumberOfHandles) = 0;

virtual RTI::ObjectHandle registerObjectInstanceWithRegion(
    RTI::ObjectClassHandle theClass,
    RTI::AttributeHandle theAttributes[],
    RTI::Region* theRegions[],
    RTI::ULong theNumberOfHandles) = 0;

virtual void associateRegionForUpdates (
    RTI::Region& theRegion,
    RTI::ObjectHandle theObject,
```

```
    const RTI::AttributeHandleSet& theAttributes) = 0;

virtual void unassociateRegionForUpdates(
    RTI::Region& theRegion,
    RTI::ObjectHandle theObject) = 0;

virtual void subscribeObjectClassAttributesWithRegion(
    RTI::ObjectClassHandle theClass,
    RTI::Region& theRegion,
    const RTI::AttributeHandleSet &attributeList,
    RTI::Boolean active = RTI::RTI_TRUE) = 0;

virtual void unsubscribeObjectClassWithRegion(
    RTI::ObjectClassHandle theClass,
    RTI::Region& theRegion) = 0;

virtual void subscribeInteractionClassWithRegion (
    RTI::InteractionClassHandle theClass,
    RTI::Region& theRegion,
    RTI::Boolean active = RTI::RTI_TRUE) = 0;

virtual void unsubscribeInteractionClassWithRegion (
    RTI::InteractionClassHandle theClass,
    RTI::Region& theRegion) = 0;

virtual RTI::EventRetractionHandle sendInteractionWithRegion(
    RTI::InteractionClassHandle theInteraction,
    const RTI::ParameterHandleValuePairSet &theParameters,
    const RTI::FedTime& theTime,
    const char* theTag,
    const RTI::Region& theRegion) = 0;

virtual void sendInteractionWithRegion(
    RTI::InteractionClassHandle theInteraction,
    const RTI::ParameterHandleValuePairSet &theParameters,
    const char* theTag,
    const RTI::Region& theRegion) = 0;

virtual void requestClassAttributeValueUpdateWithRegion(
    RTI::ObjectClassHandle theClass,
    const RTI::AttributeHandleSet& theAttributes,
    const RTI::Region& theRegion) = 0;

virtual RTI::ObjectClassHandle getObjectClassHandle (
    const char* theName) = 0;

virtual char* getObjectClassName (
    RTI::ObjectClassHandle theHandle) = 0;

virtual RTI::AttributeHandle getAttributeHandle (
```

```
    const char* theName,
    RTI::ObjectClassHandle whichClass) = 0;

virtual char* getAttributeName (
    RTI::AttributeHandle theHandle,
    RTI::ObjectClassHandle whichClass) = 0;

virtual RTI::InteractionClassHandle getInteractionClassHandle (
    const char* theName) = 0;

virtual char* getInteractionClassName (
    RTI::InteractionClassHandle theHandle) = 0;

virtual RTI::ParameterHandle getParameterHandle (
    const char* theName,
    RTI::InteractionClassHandle whichClass) = 0;

virtual char* getParameterName (
    RTI::ParameterHandle theHandle,
    RTI::InteractionClassHandle whichClass) = 0;

virtual RTI::ObjectHandle getObjectInstanceHandle(
    const char* theName) = 0;

virtual char* getObjectInstanceName(
    RTI::ObjectHandle theHandle) = 0;

virtual RTI::SpaceHandle getRoutingSpaceHandle (
    const char* theName) = 0;

virtual const char* getRoutingSpaceName (
    const RTI::SpaceHandle theHandle) = 0;

virtual RTI::DimensionHandle getDimensionHandle(
    const char* theName,
    RTI::SpaceHandle whichSpace) = 0;

virtual char* getDimensionName(
    RTI::DimensionHandle theHandle,
    RTI::SpaceHandle whichSpace) = 0;

virtual RTI::SpaceHandle
    getAttributeRoutingSpaceHandle(
        RTI::AttributeHandle theHandle,
        RTI::ObjectClassHandle whichClass) = 0;

virtual RTI::ObjectClassHandle getObjectClass (
    RTI::ObjectHandle theObject) = 0;

virtual RTI::SpaceHandle getInteractionRoutingSpaceHandle(
```

```
    RTI::InteractionClassHandle theHandle) = 0;

virtual RTI::TransportationHandle getTransportationHandle(
    const char *theName) = 0;

virtual char* getTransportationName(
    RTI::TransportationHandle theHandle) = 0;

virtual RTI::OrderingHandle getOrderingHandle(
    const char *theName) = 0;

virtual char* getOrderingName(
    RTI::OrderingHandle theHandle) = 0;

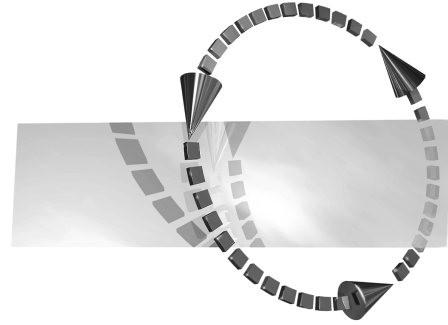
virtual void enableClassRelevanceAdvisorySwitch() = 0;
virtual void disableClassRelevanceAdvisorySwitch() = 0;
virtual void enableAttributeRelevanceAdvisorySwitch() = 0;
virtual void disableAttributeRelevanceAdvisorySwitch() = 0;
virtual void enableAttributeScopeAdvisorySwitch() = 0;
virtual void disableAttributeScopeAdvisorySwitch() = 0;
virtual void enableInteractionRelevanceAdvisorySwitch() = 0;
virtual void disableInteractionRelevanceAdvisorySwitch() = 0;

virtual RTI::Boolean tick() = 0;

virtual RTI::Boolean tick(
    RTI::TickTime minimum,
    RTI::TickTime maximum) = 0;

virtual RTI::RegionToken getRegionToken(RTI::Region*) = 0;

virtual RTI::Region* getRegion(RTI::RegionToken) = 0;
};
#endif // DtVrIRtiAmbassador_H_
#endif // DtRTI103
```

Index

A

absNotifyObs.h header file 3-2
ackInter.h header file 1-2
ackPdu.h header file 3-3
ackRPdu.h header file 3-5
actionPdu.h header file 3-7
actReqInter.h header file 1-2
actReqRPdu.h header file 3-10
actRespInter.h header file 1-2
actRespRPdu.h header file 3-12
 address
 MÄK Technologies xviii
 Adobe Acrobat xv
aggAggList.h header file 3-14
aggEntDec.h header file 3-16
aggEntEnc.h header file 3-18
aggEntList.h header file 3-20
aggPub.h header file 1-2
aggregateSR.h header file 3-29
aggStatePdu.h header file 3-22
antPatRec.h header file 3-34
approximator.h header file 3-36
apSpec.h header file 3-38
arealObjStPdu.h header file 3-39
artPart.h header file 3-43
artPartList.h header file 3-47
artPartThresh.h header file 3-51
attPartList.h header file 3-54

B

baseEmbedDec.h header file 3-57
baseEmbedEnc.h header file 3-59
baseEmitttrDec.h header file 3-61
baseEmitttrEnc.h header file 3-63
baseEntDec.h header file 3-66
baseEntEnc.h header file 3-69
baseEntitySR.h header file 3-72

baseExConn.h header file 3-77
baseHashKey.h header file 3-79
baseMsg.h header file 3-81
baseReflAgg.h header file 3-83
baseReflEnt.h header file 3-85
beamAntRec.h header file 3-86

C

cartView.h header file 3-89
 class
 DtAcknowledgePdu 2-2
 DtActionRequestPdu 2-4
 DtActionResponsePdu 2-6
 DtCollisionPdu 2-8
 DtCommentPdu 2-10
 DtCreateEntityPdu 2-12
 DtDataPdu 2-13
 DtDataQueryPdu 2-14
 DtDesignatorPdu 2-16
 DtDetonationPdu 2-18
 DtEmissionPdu 2-21
 DtEntityStatePdu 2-26
 DtEventReportPdu 2-32
 DtFirePdu 2-33
 DtLaserPdu 2-35
 DtMessagePdu 2-36
 DtPduWithData 2-41
 DtReceiverPdu 2-48
 DtRemoveEntityPdu 2-49
 DtRepairCompletePdu 2-50
 DtRepairResponsePdu 2-51
 DtResupplyCancelPdu 2-52
 DtResupplyOfferPdu 2-53
 DtResupplyReceivedPdu 2-55
 DtSetDataPdu 2-57
 DtSignalPdu 2-58
 DtSimanPdu 2-60
 DtStartResumePdu 2-62

DtStopFreezePdu 2-64
DtTransmitterPdu 2-66
DtViewControlPdu 2-70
 listed alphabetically 1-4
 listed by header file 1-23
ClientSocket.h header file 3-91
cliServer.h header file 3-94
collideInter.h file 1-2
collidePdu.h header file 3-94
collisElasPdu.h header file 3-96
commentInter.h header file 1-2
commentPdu.h header file 3-99
commentRPdu.h header file 3-101
 conventions
 manual xix
createEntInt.h header file 1-2
createEntRPdu.h header file 3-103
createPdu.h header file 3-105

D

dAggDec.h header file 3-107
dAggEnc.h header file 3-108
dAggPub.h header file 3-110
dataInter.h header file 1-2
dataPdu.h header file 3-111
dataQInter.h header file 1-2
dataQueryRPdu.h header file 3-112
dataRPdu.h header file 3-114
datumIdType.h header file 3-116
datumMap.h header file 3-117
datumRecSet.h header file 3-119
dcm.h header file 3-126
deadReckoner.h header file 3-127
decFactory.h header file 3-131
dEntityDec.h header file 3-134
dEntityEnc.h header file 3-136
dEntityPub.h header file 3-138
designatePdu.h header file 3-139
designatorDec.h header file 3-141
designatorEnc.h header file 3-143
designatorSR.h header file 3-145
detonateInter.h header file 1-2
detonatePdu.h header file 3-147
dExerciseConn.h header file 3-150
dInteraction.h header file 3-156
directRotRate.h header file 3-157
dirRtiAmb.h header file 3-158
 DIS
 PDUs 2-1

disEnums.h header file 3-170
disPduFactory.h header file 3-215
dRefEmSysList.h header file 3-216
dRefAggList.h header file 3-218
dReflectedAgg.h header file 3-221
dReflectedEnt.h header file 3-224
dReflectedObj.h header file 3-227
dRefEmitrSys.h header file 3-229
dRefEntList.h header file 3-231
dRefObjList.h header file 3-234
dscvObjCbInfo.h header file 3-237
dStateMsg.h header file 3-239
DtAcknowledgePdu class 2-2
DtActionRequestPdu class 2-4
DtActionResponsePdu class 2-6
DtCollisionPdu class 2-8
DtCommentPdu class 2-10
DtCreateEntityPdu class 2-12
DtDataPdu class 2-13
DtDataQueryPdu class 2-14
DtDesignatorPdu class 2-16
DtDetonationPdu class 2-18
DtEmissionPdu class 2-21
DtEntityStatePdu class 2-26
DtEventReportPdu class 2-32
DtFirePdu class 2-33
DtLaserPdu class 2-35
DtMessagePdu class 2-36
DtPduWithData class 2-41
DtReceiverPdu class 2-48
DtRemoveEntityPdu class 2-49
DtRepairCompletePdu class 2-50
DtRepairResponsePdu class 2-51
DtResupplyCancelPdu class 2-52
DtResupplyOfferPdu class 2-53
DtResupplyReceivedPdu class 2-55
DtSetDataPdu class 2-57
DtSignalPdu class 2-58
DtSimanPdu class 2-60
DtStartResumePdu class 2-62
DtStopFreezePdu class 2-64
DtTransmitterPdu class 2-66
DtViewControlPdu class 2-70

E

elecEmitPdu.h header file 3-241, 3-384
embedSystemSR.h header file 3-245
emissionPdu.h header file 3-247
emitterSysDec.h header file 3-251

emitterSysEnc.h header file 3-253
emitterSysSR.h header file 3-255
emitttrBeamSR.h header file 3-257
emptyFomMap.h header file 3-261
encFactory.h header file 3-263
entityPub.h header file 1-2
entitySR.h header file 3-266
EntityTypes.h header file 3-271
enum2string.h header file 3-292
envDec.h header file 3-293
envEnc.h header file 3-295
enviroTypeRec.h header file 3-297
envProcPdu.h header file 3-298
esPdu.h header file 3-301
esUpdatePdu.h header file 3-304
Euler.h header file 3-308
eventId.h header file 3-309
eventRepInter.h header file 1-2
eventReptPdu.h header file 3-312
eventReptRPdu.h header file 3-313
exerciseConn.h header file 1-2

F

fax
 MÄK Technologies xviii
fedReader.h header file 3-315
fedTarget.h header file 3-317
fireInter.h header file 1-2
firePdu.h header file 3-321
fom.h header file 3-323
fomMapper.h header file 3-326

G

gedBasFixWing.h header file 3-332
gedBasRotWing.h header file 3-335
gedBasSoldier.h header file 3-338
gedBasVehicle.h header file 3-341
gedEnhFixWing.h header file 3-344
gedEnhRotWing.h header file 3-346
gedEnhSoldier.h header file 3-348
gedEnhVehicle.h header file 3-350
gedLogVehicle.h header file 3-352
genericDec.h header file 3-355
genericEnc.h header file 3-356
genericPub.h header file 3-357
genericSR.h header file 3-358
geodCoord.h header file 3-360
globalObjDes.h header file 3-363

globIdHLst.h header file 3-364
gridAxiRecFix.h header file 3-366
gridAxiRecVar.h header file 3-369
gridDataPdu.h header file 3-371
gridDataRec.h header file 3-376
gridDataRecT0.h header file 3-378
gridDataRecT1.h header file 3-380
gridDataRecT2.h header file 3-382

H

hAckDec.h header file 3-384
hAckEnc.h header file 3-386
hAckInter.h header file 3-387
hActReqDec.h header file 3-389
hActReqEnc.h header file 3-391
hActReqInter.h header file 3-392
hActReqObjDec.h header file 3-394
hActReqObjEnc.h header file 3-396
hActReqObjInt.h header file 3-397
hActRespDec.h header file 3-399
hActRespEnc.h header file 3-401
hActRespInter.h header file 3-403
hActRspObjDec.h header file 3-405
hActRspObjEnc.h header file 3-407
hActRspObjInt.h header file 3-409
hAggPub.h header file 3-411
hAppRadSigDec.h header file 3-413
hAppRadSigEnc.h header file 3-415
hashItem.h header file 3-417
hashlist.h header file 3-418
hAttChgReqDec.h header file 3-422
hAttChgReqEnc.h header file 3-423
hAttChgReqInt.h header file 3-424
hAttChgResDec.h header file 3-427
hAttChgResEnc.h header file 3-429
hAttChgResInt.h header file 3-431
hCollideDec.h header file 3-434
hCollideEnc.h header file 3-436
hCollideInter.h header file 3-438
hDbiRadSigDec.h header file 3-471
hDbiRadSigEnc.h header file 3-473
hDesignatrPub.h header file 3-475
hDetDec.h header file 3-477
hDetEnc.h header file 3-479
hDetInter.h header file 3-481
 header file 3-1
 fireInter.h 1-2
 reflectedEnt.h 1-3
 stateMsg.h 1-3

vcInter.h 1-3
absNotifyObs.h 3-2
ackInter.h 1-2
ackPdu.h 3-3
ackRPdu.h 3-5
actionPdu.h 3-7
actReqInter.h 1-2
actReqRPdu.h 3-10
actRespInter.h 1-2
actRespRPdu.h 3-12
aggAggList.h 3-14
aggEntDec.h 3-16
aggEntEnc.h 3-18
aggEntList.h 3-20
aggPub.h 1-2
aggregateSR.h 3-29
aggStatePdu.h 3-22
antPatRec.h 3-34
approximator.h 3-36
apSpec.h 3-38
arealObjStPdu.h 3-39
artPart.h 3-43
artPartList.h 3-47
artPartThresh.h 3-51
attPartList.h 3-54
baseEmbedDec.h 3-57
baseEmbedEnc.h 3-59
baseEmittRDec.h 3-61
baseEmittREnc.h 3-63
baseEntDec.h 3-66
baseEntEnc.h 3-69
baseEntitySR.h 3-72
baseExConn.h 3-77
baseHashKey.h 3-79
baseMsg.h 3-81
baseReflAgg.h 3-83
baseReflEnt.h 3-85
beamAntRec.h 3-86
cartView.h 3-89
ClientSocket.h 3-91
cliServer.h 3-94
collideInter.h 1-2
collidePdu.h 3-94
collisElasPdu.h 3-96
commentInter.h 1-2
commentPdu.h 3-99
commentRPdu.h 3-101
createEntInt.h 1-2
createEntRPdu.h 3-103
createPdu.h 3-105

cross referenced to classes 1-4
dAggDec.h 3-107
dAggEnc.h 3-108
dAggPub.h 3-110
dataInter.h 1-2
dataPdu.h 3-111
dataQInter.h 1-2
dataQueryRPdu.h 3-112
dataRPdu.h 3-114
datumIdType.h 3-116
datumMap.h 3-117
datumRecSet.h 3-119
dcm.h 3-126
deadReckoner.h 3-127
decFactory.h 3-131
dEntityDec.h 3-134
dEntityEnc.h 3-136
dEntityPub.h 3-138
designatePdu.h 3-139
designatorDec.h 3-141
designatorEnc.h 3-143
designatorSR.h 3-145
detonateInter.h 1-2
detonatePdu.h 3-147
dExerciseConn.h 3-150
dInteraction.h 3-156
directRotRate.h 3-157
dirRtiAmb.h 3-158
disEnums.h 3-170
disPduFactory.h 3-215
dRefEmSysList.h 3-216
dReflAggList.h 3-218
dReflectedAgg.h 3-221
dReflectedEnt.h 3-224
dReflectedObj.h 3-227
dReflEmitrSys.h 3-229
dReflEntList.h 3-231
dRefObjList.h 3-234
dscvObjCbInfo.h 3-237
dStateMsg.h 3-239
elecEmitPdu.h 3-241, 3-384
embedSystemSR.h 3-245
emissionPdu.h 3-247
emitterSysDec.h 3-251
emitterSysEnc.h 3-253
emitterSysSR.h 3-255
emittRBeamSR.h 3-257
emptyFomMap.h 3-261
encFactory.h 3-263
entityPub.h 1-2

entitySR.h 3-266
 EntityTypes.h 3-271
 enum2string.h 3-292
 envDec.h 3-293
 envEnc.h 3-295
 enviroTypeRec.h 3-297
 envProcPdu.h 3-298
 esPdu.h 3-301
 esUpdatePdu.h 3-304
 Euler.h 3-308
 eventId.h 3-309
 eventRepInter.h 1-2
 eventReptPdu.h 3-312
 eventReptRPdu.h 3-313
 exerciseConn.h 1-2
 fedReader.h 3-315
 fedTarget.h 3-317
 firePdu.h 3-321
 fom.h 3-323
 fomMapper.h 3-326
 gedBasFixWing.h 3-332
 gedBasRotWing.h 3-335
 gedBasSoldier.h 3-338
 gedBasVehicle.h 3-341
 gedEnhFixWing.h 3-344
 gedEnhRotWing.h 3-346
 gedEnhSoldier.h 3-348
 gedEnhVehicle.h 3-350
 gedLogVehicle.h 3-352
 genericDec.h 3-355
 genericEnc.h 3-356
 genericPub.h 3-357
 genericSR.h 3-358
 geodCoord.h 3-360
 globalObjDes.h 3-363
 globIdHLst.h 3-364
 gridAxiRecFix.h 3-366
 gridAxiRecVar.h 3-369
 gridDataPdu.h 3-371
 gridDataRec.h 3-376
 gridDataRecT0.h 3-378
 gridDataRecT1.h 3-380
 gridDataRecT2.h 3-382
 for grouping other header files 1-2
 hAckDec.h 3-384
 hAckEnc.h 3-386
 hAckInter.h 3-387
 hActReqDec.h 3-389
 hActReqEnc.h 3-391
 hActReqInter.h 3-392
 hActReqObjDec.h 3-394
 hActReqObjEnc.h 3-396
 hActReqObjInt.h 3-397
 hActRespDec.h 3-399
 hActRespEnc.h 3-401
 hActRespInter.h 3-403
 hActRspObjDec.h 3-405
 hActRspObjEnc.h 3-407
 hActRspObjInt.h 3-409
 hAggPub.h 3-411
 hAppRadSigDec.h 3-413
 hAppRadSigEnc.h 3-415
 hashItem.h 3-417
 hashlist.h 3-418
 hAttChgReqDec.h 3-422
 hAttChgReqEnc.h 3-423
 hAttChgReqInt.h 3-424
 hAttChgResDec.h 3-427
 hAttChgResEnc.h 3-429
 hAttChgResInt.h 3-431
 hCollideDec.h 3-434
 hCollideEnc.h 3-436
 hCollideInter.h 3-438
 hDbiRadSigDec.h 3-471
 hDbiRadSigEnc.h 3-473
 hDesignatrPub.h 3-475
 hDetDec.h 3-477
 hDetEnc.h 3-479
 hDetInter.h 3-481
 hEmitrBeamPub.h 3-485
 hEmittrSysPub.h 3-487
 hEncARSigDec.h 3-489
 hEncARSigEnc.h 3-491
 hEntityPub.h 3-493
 hExerciseConn.h 3-499
 hFireDec.h 3-508
 hFireEnc.h 3-510
 hFireInter.h 3-512
 hierAdpt.h 3-515
 hInteraction.h 3-515
 hInterFactory.h 3-519
 hInterWData.h 3-521
 hIntWEncDec.h 3-527
 hlaObjFact.h 3-537
 hlaObj.h 3-530
 hlaObjWithSR.h 3-539
 hLcDec.h 3-542
 hLcEnc.h 3-543
 hLcInter.h 3-544
 hObjPub.h 3-546

hostStructs.h 3-550
hRadioRcvrPub.h 3-557
hRadioXmitPub.h 3-559
hRawBinRSDec.h 3-561
hRawBinRSEnc.h 3-563
hRefBeamList.h 3-565
hRefDesigList.h 3-567
hRefEmSysList.h 3-569
hRefIAggList.h 3-573
hRefIBeam.h 3-576
hRefIDesigntr.h 3-578
hReflectedAgg.h 3-580
hReflectedEnt.h 3-582
hRefIEmitrSys.h 3-593
hRefIEntList.h 3-595
hRefIRadRecvr.h 3-598
hRefIRadXmitr.h 3-600
hRefIRadRvList.h 3-602
hRefIRadXmList.h 3-604
hSignalInter.h 3-658
hStartDec.h 3-661
hStartEnc.h 3-663
hStartInter.h 3-664
hStateDecoder.h 3-666
hStateEncoder.h 3-670
hStateMsg.h 3-675
hStopDec.h 3-678
hStopEnc.h 3-680
hStopInter.h 3-681
hUnknownInter.h 3-684
hVcDec.h 3-686
hVcEnc.h 3-687
hVcInter.h 3-688
iffAtcNavBeam.h 3-690
iffAtcNavLayr.h 3-692
iffAtcNavOper.h 3-694
iffAtcNavOpr2.h 3-697
iffAtcNavParm.h 3-699
iffAtcNavPdu.h 3-701
iffAtcNavSyID.h 3-705
intClassDesc.h 3-707
intDecFact.h 3-710
interaction.h 1-3
intercmCtlPdu.h 3-715
intercmEntDst.h 3-719
intercmGrpAsg.h 3-721
intercmGrpDst.h 3-723
intercmParRec.h 3-725
intercmSigPdu.h 3-727
interDecoder.h 3-730
interEncoder.h 3-733
interMgr.h 3-736
intHashKey.h 3-738
isGroupOfPdu.h 3-739
isPartOfPdu.h 3-744
jammerBeamDec.h 3-747
jammerBeamEnc.h 3-749
laserPdu.h 3-751
lcEnd.h 3-753
lcEOFAction.h 3-754
lcHeartbeat.h 3-755
lcInter.h 1-3
lcPause.h 3-756
lcPdu.h 3-757
lcPlayAction.h 3-761
lcPlayFile.h 3-762
lcPlay.h 3-760
lcQuit.h 3-763
lcRecAction.h 3-764
lcRecFile.h 3-765
lcRecord.h 3-766
lcSetTime.h 3-767
lcSkipTime.h 3-768
lcStart.h 3-769
lcStop.h 3-770
lcTimeout.h 3-772
lcTimeScale.h 3-771
leAppearPdu.h 3-773
leArtPartsPdu.h 3-778
leDeadReck.h 3-781
leDetonPdu.h 3-783
leEntityID.h 3-788
leEntityOri.h 3-790
leEntityVel.h 3-792
leFirePdu.h 3-794
leLocEntity.h 3-799
leLocWorld.h 3-801
leOriError.h 3-803
lePosError.h 3-805
leTspiPdu.h 3-807
LibMatrix.h 3-811
linObjStPdu.h 3-817
locObjMgr.h 3-823
mathTypes.h 3-827
matricInc.h 1-2
milEntDec.h 3-831
milEntEnc.h 3-833
milPlatDec.h 3-835
milPlatEnc.h 3-837
mineDataPdu.h 3-839

minePerimeter.h 3-847
mineQueryPdu.h 3-849
mineRsNackPdu.h 3-853
mineStatePdu.h 3-856
mtlInc.h 1-2
munEntDec.h 3-860
munEntEnc.h 3-862
netBoolean.h 3-864
NetSocket.h 3-866
NetStructs.h 3-871
netTimeStamp.h 3-880
NetTypes.h 3-882
objClassDesc.h 3-893
objDesList.h 3-897
objectType.h 3-899
objIdHashKey.h 3-901
objIdHashLst.h 3-902
objRefCbInfo.h 3-904
pairSock.h 3-906
pduCbInfo.h 3-910
pduCbMgr.h 3-911
pduDatRecSet.h 3-913
pduFactory.h 3-915
pdu.h 3-907
pduWithData.h 3-917
physEntDec.h 3-923
physEntEnc.h 3-926
pointObjStPdu.h 3-929
printUtil.h 3-933
ProcControl.h 3-934
queryPdu.h 3-935
radarBeamDec.h 3-937
radarBeamEnc.h 3-939
radioRecvrDec.h 3-941
radioRecvrEnc.h 3-943
radioRecvrSR.h 3-945
radioXmitDec.h 3-947
radioXmitEnc.h 3-949
radioXmitSR.h 3-952
receivePdu.h 3-953
recIntrCbInfo.h 3-955
recordPdu.h 3-957
recordSet.h 3-960
recordSetPdu.h 3-964
recQueryPdu.h 3-967
refAttrCbInfo.h 3-970
refAttrCbList.h 3-972
refEmitLst.h 1-3
refGenericObj.h 3-974
refGenList.h 3-976
reflAggList.h 1-3, 3-978
reflectedAgg.h 1-3
reflectedObj.h 1-3
refEmittr.h 1-3
refEntList.h 1-3
refObjList.h 1-3
refObjMgr.h 3-980
regCtlCbInfo.h 3-983
regCtlCbList.h 3-985
remEntInter.h 1-3
remObjCbInfo.h 3-987
removeEntRPdu.h 3-989
removePdu.h 3-991
repairPdu.h 3-993
repCompInter.h 1-3
repRespInter.h 1-3
resupplyInter.h 1-3
resupplyPdu.h 3-995
rprDecFact.h 3-999
rprEncFact.h 3-1000
rprFomMap.h 3-1001
rprIDecFact.h 3-1004
rprIEncFact.h 3-1005
rprInterFact.h 3-1006
rprObjFact.h 3-1007
script.h 3-1008
servReqInter.h 1-3
servReqPdu.h 3-1009
setDataInter.h 1-3
setDataPdu.h 3-1011
setDataRPdu.h 3-1012
setRecordPdu.h 3-1014
signalInter.h 1-3
signalPdu.h 3-1017
simanPdu.h 3-1019
simRprFomMap.h 3-1020
smoothUtil.h 3-1026
smUtil.h 3-1030
Socket.h 3-1031
soldierDec.h 3-1035
soldierEnc.h 3-1037
startInter.h 1-3
startPdu.h 3-1039
startResRPdu.h 3-1041
stateRep.h 3-1043
stIntCbInfo.h 3-1045
stIntCbList.h 3-1047
stopFrzRPdu.h 3-1049
stopInter.h 1-3
stopPdu.h 3-1051

-
- str.h* 3-1053
 - strtUpdCbInfo.h* 3-1057
 - strtUpdCbList.h* 3-1059
 - supEmisEsNozz.h* 3-1061
 - supEmisEsPdu.h* 3-1063
 - supEmisEsProp.h* 3-1066
 - taitBryan.h* 3-1070
 - tcpForwarder.h* 3-1071
 - tcpSocket.h* 3-1072
 - tcpSocketMgr.h* 3-1074
 - thresolder.h* 3-1075
 - topoCoord.h* 3-1079
 - topoView.h* 3-1080
 - transCtlPdu.h* 3-1083
 - transmitPdu.h* 3-1086
 - trkJamBeamDec.h* 3-1089
 - trkJamBeamEnc.h* 3-1091
 - undwaAcBeam.h* 3-1094
 - undwaAcEmitSys.h* 3-1096
 - undwaAcPasAct.h* 3-1098
 - undwaAcPdu.h* 3-1100
 - undwaAcShaft.h* 3-1104
 - unknownHObj.h* 3-1106
 - unknownPdu.h* 3-1108
 - updCtlCbInfo.h* 3-1109
 - updCtlCbList.h* 3-1111
 - utmCoord.h* 3-1113
 - utmView.h* 3-1117
 - varDatumRec.h* 3-1119
 - vcPdu.h* 3-1122
 - vlInc.h* 1-2
 - vlList.h* 3-1131
 - vlprint.h* 3-1138
 - vlTime.h* 3-1140
 - vlutil.h* 3-1141
 - vlutilInc.h* 1-2
 - vlVector.h* 3-1145
 - vrlFedAmb.h* 3-1147
 - vrlFedTime.h* 3-1153
 - vrlRtiAmb.h* 3-1157
 - hEmitrBeamPub.h* header file 3-485
 - hEmitrSysPub.h* header file 3-487
 - hEncARSigDec.h* header file 3-489
 - hEncARSigEnc.h* header file 3-491
 - hEntityPub.h* header file 3-493
 - hExerciseConn.h* header file 3-499
 - hFireDec.h* header file 3-508
 - hFireEnc.h* header file 3-510
 - hFireInter.h* header file 3-512
 - hierAdpt.h* header file 3-515
 - hInteraction.h* header file 3-515
 - hInterFactory.h* header file 3-519
 - hInterWData.h* header file 3-521
 - hIntWEncDec.h* header file 3-527
 - hlaObjFact.h* header file 3-537
 - hlaObj.h* header file 3-530
 - hlaObjWithSR.h* header file 3-539
 - hLcDec.h* header file 3-542
 - hLcEnc.h* header file 3-543
 - hLcInter.h* header file 3-544
 - hObjPub.h* header file 3-546
 - hostStructs.h* header file 3-550
 - hRadioRcvrPub.h* header file 3-557
 - hRadioXmitPub.h* header file 3-559
 - hRawBinRSDec.h* header file 3-561
 - hRawBinRSEnc.h* header file 3-563
 - hRefBeamList.h* header file 3-565
 - hRefDesigList.h* header file 3-567
 - hRefEmSysList.h* header file 3-569
 - hReflAggList.h* header file 3-573
 - hReflBeam.h* header file 3-576
 - hReflDesigntr.h* header file 3-578
 - hReflectedAgg.h* header file 3-580
 - hReflectedEnt.h* header file 3-582
 - hReflEmitrSys.h* header file 3-593
 - hReflEntList.h* header file 3-595
 - hRefRadRecvr.h* header file 3-598
 - hRefRadXmitr.h* header file 3-600
 - hRefRadRvList.h* header file 3-602
 - hRefRadXmList.h* header file 3-604
 - hSignalInter.h* header file 3-658
 - hStartDec.h* header file 3-661
 - hStartEnc.h* header file 3-663
 - hStartInter.h* header file 3-664
 - hStateDecoder.h* header file 3-666
 - hStateEncoder.h* header file 3-670
 - hStateMsg.h* header file 3-675
 - hStopDec.h* header file 3-678
 - hStopEnc.h* header file 3-680
 - hStopInter.h* header file 3-681
 - hUnknownInter.h* header file 3-684
 - hVcDec.h* header file 3-686
 - hVcEnc.h* header file 3-687
 - hVcInter.h* header file 3-688
- I**
- iffAtcNavBeam.h* header file 3-690
 - iffAtcNavLayr.h* header file 3-692
 - ifNtch*

iffAtcNavOpr2.h header file 3-697
iffAtcNavParm.h header file 3-699
iffAtcNavPdu.h header file 3-701
iffAtcNavSyID.h header file 3-705
intClassDesc.h header file 3-707
intDecFact.h header file 3-710
interaction.h header file 1-3
intercmCtlPdu.h header file 3-715
intercmEntDst.h header file 3-719
intercmGrpAsg.h header file 3-721
intercmGrpDst.h header file 3-723
intercmParRec.h header file 3-725
intercmSigPdu.h header file 3-727
interDecoder.h header file 3-730
interEncoder.h header file 3-733
interMgr.h header file 3-736
intHashKey.h header file 3-738
isGroupOfPdu.h header file 3-739
isPartOfPdu.h header file 3-744

J

jammerBeamDec.h header file 3-747
jammerBeamEnc.h header file 3-749

L

laserPdu.h header file 3-751
lcEnd.h header file 3-753
lcEofAction.h header file 3-754
lcHeartbeat.h header file 3-755
lcInter.h header file 1-3
lcPause.h header file 3-756
lcPdu.h header file 3-757
lcPlayAction.h header file 3-761
lcPlayFile.h header file 3-762
lcPlay.h header file 3-760
lcQuit.h header file 3-763
lcRecAction.h header file 3-764
lcRecFile.h header file 3-765
lcRecord.h header file 3-766
lcSetTime.h header file 3-767
lcSkipTime.h header file 3-768
lcStart.h header file 3-769
lcStop.h header file 3-770
lcTimeout.h header file 3-772
lcTimeScale.h header file 3-771
leAppearPdu.h header file 3-773
leArtPartsPdu.h header file 3-778

leDeadReck.h header file 3-781
leDetonPdu.h header file 3-783
leEntityID.h header file 3-788
leEntityOri.h header file 3-790
leEntityVel.h header file 3-792
leFirePdu.h header file 3-794
leLocEntity.h header file 3-799
leLocWorld.h header file 3-801
leOriError.h header file 3-803
lePosError.h header file 3-805
leTspiPdu.h header file 3-807
LibMatrix.h header file 3-811
linObjStPdu.h header file 3-817
locObjMgr.h header file 3-823
 Logger xvii

M

maintenance agreement xviii
 MÄK Logger xvii
 MÄK Plan View Display xvii
 MÄK RTI xvii
 MÄK Stealth xvii
 MÄK Technical Support xviii
 manual
 conventions xix
mathTypes.h header file 1-2, 3-827
matrixInc.h file 1-2
milEntDec.h header file 3-831
milEntEnc.h header file 3-833
milPlatDec.h header file 3-835
milPlatEnc.h header file 3-837
mineDataPdu.h header file 3-839
minePerimeter.h header file 3-847
mineQueryPdu.h header file 3-849
mineRsNackPdu.h header file 3-853
mineStatePdu.h header file 3-856
mtlInc.h header file 1-2
munEntDec.h header file 3-860
munEntEnc.h header file 3-862

N

netBoolean.h header file 3-864
NetSocket.h header file 3-866
NetStructs.h header file 3-871
netTimeStamp.h header file 3-880
NetTypes.h header file 3-882

O

objClassDesc.h header file 3-893
objDesList.h header file 3-897
objectType.h header file 3-899
objIdHashKey.h header file 3-901
objIdHashLst.h header file 3-902
objRefCbInfo.h header file 3-904

P

pairSock.h header file 3-906
PDU
 class reference 2-1
pduCbInfo.h header file 3-910
pduCbMgr.h header file 3-911
pduDatRecSet.h header file 3-913
pduFactory.h header file 3-915
pdu.h header file 3-907
pdus.h grouping file 1-2
pduWithData.h header file 3-917
phone number
 MÄK Technologies xviii
physEntDec.h header file 3-923
physEntEnc.h header file 3-926
Plan View Display xvii
pointObjStPdu.h header file 3-929
Portable Document Format (PDF) xv to xvi
printUtil.h header file 3-933
ProcControl.h header file 3-934
products
 Logger xvii
 Plan View Display xvii
 RTI xvii
 Stealth xvii
 technical support xviii
 upgrades xviii

Q

queryPdu.h header file 3-935

R

radarBeamDec.h header file 3-937
radarBeamEnc.h header file 3-939
radioRecvrDec.h header file 3-941
radioRecvrEnc.h header file 3-943
radioRecvrSR.h header file 3-945
radioXmitDec.h header file 3-947

radioXmitEnc.h header file 3-949
radioXmitSR.h header file 3-952
receivePdu.h header file 3-953
recIntrCbInfo.h header file 3-955
recordPdu.h header file 3-957
recordSet.h header file 3-960
recordSetPdu.h header file 3-964
recQueryPdu.h header file 3-967
refAttrCbInfo.h header file 3-970
refAttrCbList.h header file 3-972
refEmitLst.h header file 1-3
refGenericObj.h header file 3-974
refGenList.h header file 3-976
reflAggList.h header file 1-3, 3-978
reflectedAgg.h header file 1-3
reflectedEnt.h header file 1-3
reflectedObj.h header file 1-3
reflEmitttr.h header file 1-3
reflEntList.h header file 1-3
reflObjList.h header file 1-3
refObjMgr.h header file 3-980
regCtlCbInfo.h header file 3-983
regCtlCbList.h header file 3-985
remEntInter.h header file 1-3
remObjCbInfo.h header file 3-987
removeEntRPdu.h header file 3-989
removePdu.h header file 3-991
repairPdu.h header file 3-993
repCompInter.h header file 1-3
repRespInter.h header file 1-3
resupplyInter.h header file 1-3
resupplyPdu.h header file 3-995
rprDecFact.h header file 3-999
rprEncFact.h header file 3-1000
rprFomMap.h header file 3-1001
rprIDecFact.h header file 3-1004
rprIEncFact.h header file 3-1005
rprInterFact.h header file 3-1006
rprObjFact.h header file 3-1007
RTI xvii

S

script.h header file 3-1008
servReqInter.h header file 1-3
servReqPdu.h header file 3-1009
setDataInter.h header file 1-3
setDataPdu.h header file 3-1011
setDataRPdu.h header file 3-1012
setRecordPdu.h header file 3-1014

signalInter.h header file 1-3
signalPdu.h header file 3-1017
simanPdu.h header file 3-1019
simRprFomMap.h header file 3-1020
smoothUtil.h header file 3-1026
smUtil.h header file 3-1030
Socket.h header file 3-1031
soldierDec.h header file 3-1035
soldierEnc.h header file 3-1037
startInter.h header file 1-3
startPdu.h header file 3-1039
startResRPdu.h header file 3-1041
stateMsg.h header file 1-3
stateRep.h header file 3-1043
Stealth xvii
stIntCbInfo.h header file 3-1045
stIntCbList.h header file 3-1047
stopFrzRPdu.h header file 3-1049
stopInter.h header file 1-3
stopPdu.h header file 3-1051
str.h header file 3-1053
strtUpdCbInfo.h header file 3-1057
strtUpdCbList.h header file 3-1059
supEmisEsNozz.h header file 3-1061
supEmisEsPdu.h header file 3-1063
supEmisEsProp.h header file 3-1066
support, technical xviii

T

taitBryan.h header file 3-1070
tcpForwarder.h header file 3-1071
tcpSocket.h header file 3-1072
tcpSocketMgr.h header file 3-1074
technical support xviii
threshold.h header file 3-1075
topoCoord.h header file 3-1079

topoView.h header file 3-1080
transCtlPdu.h header file 3-1083
transmitPdu.h header file 3-1086
trkJamBeamDec.h header file 3-1089
trkJamBeamEnc.h header file 3-1091

U

undwaAcBeam.h header file 3-1094
undwaAcEmtSys.h header file 3-1096
undwaAcPasAct.h header file 3-1098
undwaAcPdu.h header file 3-1100
undwaAcShaft.h header file 3-1104
unknownHObj.h header file 3-1106
unknownPdu.h header file 3-1108
updCtlCbInfo.h header file 3-1109
updCtlCbList.h header file 3-1111
upgrades xviii
utmCoord.h header file 3-1113
utmView.h header file 3-1117

V

varDatumRec.h header file 3-1119
vcInter.h header file 1-3
vcPdu.h header file 3-1122
vlInc.h header file 1-2
vlList.h header file 3-1131
vlprint.h header file 3-1138
vlTime.h header file 3-1140
vlutil.h header file 3-1141
vlutilInc.h header file 1-2
vlVector.h header file 3-1145
vrlFedAmb.h header file 3-1147
vrlFedTime.h header file 3-1153
vrlRtiAmb.h header file 3-1157



MÄK Technologies
185 Alewife Brook Parkway
Cambridge, MA 02138
617.876.8085