MÄK VR-Forces 2.1.1-ngc Release Notes

This release note contains the following release-specific information for VR-Forces Release 2.1.1-ngc:

Document ID: VRF-2.1.1-3-020613

Systems Supported

VR-Forces 2.1.1-ngc is available for the platforms listed in Table 1. For the availability of other platforms, contact your MÄK salesperson. For toolkit users, application code must be built with the indicated compilers in order to link to VR-Forces libraries.

Table 1: Platforms supported

Platform	Compiler
SGI IRIX6.5	MipsPro7.2.1 C++ compiler (available from SGI)
PC with Windows NT 4.0 SP6 or Windows 2000 SP2	Microsoft Visual C++ 6.0, Service Pack 5

VR-Link and MÄK RTI Compatibility

To build VR-Forces applications, you must link with VR-Link 3.7.2-ngc. If you are building for HLA and want to link with the MÄK RTI, use MÄK RTI 1.3.6-ngc.

System Requirements for PCs

Minimum system requirements are:

- Microsoft Windows NT 4.0 SP6/Windows 2000/Windows XP
- Intel Pentium compatible CPU, 500 Mhz or better
- RAM: 128MB. 256-512 recommended. VR-Forces loads a copy of the terrain database for each front-end and back-end loaded.
- Hard disk space - Approximately 160 MB for complete installation of application and toolkit. Approximately 90 MB for application alone. Most of this is for the terrain databases shipped with VR-Forces. The class reference documentation for the toolkit uses 59 MB. You may need considerably more space depending on the databases you plan to use.
- Two or more multiples of the amount of RAM for virtual memory. Running VR-Forces in HLA may require increased amounts of virtual memory.

Online Help

The online help is in HTML format and uses the default browser for your computer. For all online help features to work, you must enable Java and Javascript in your browser.

Network Compatibility

HLA only

VR-Forces 2.1.1-ngc was built against VR-Link 3.7.2-ngc and is compliant with:

- RPR-FOM 1.0 and 2.0
- MÄK RTI 1.3.6-ngc
- DMSO RTI NG v1, v2, v3.0, v3.1, v3.2, and 4.0.

VR-Forces 2.1.1-ngc is compatible with applications that use earlier versions of VR-Link if they support versions of the RPR FOM listed above.

DIS only

VR-Forces 2.1.1-ngc supports DIS 2.0.4, IEEE 1278.1, 2.1.4, and IEEE 1278.1a, and can therefore interoperate with DIS applications of any of these versions.

RPR FOM Versions Supported

VR-Forces 2.1.1-ngc has built-in support for versions 1.0 and 2.0 of the RPR FOM. It also supports VR-Link's ability to support alternative FOMs through the FOM Mapper. By default, VR-Forces 2.1.1-ngc uses RPR FOM 1.0.

If you are building an application with the VR-Forces toolkit and you want to specify a version of the RPR FOM through code, pass the version number (2.0) to the *DtRprFomMapper* constructor and pass the resulting object to the *DtExerciseConn* constructor. Also, make sure you are using a federation execution name that corresponds to the right FED file. For example:

```
DtExerciseConn conn("VR-Link20", "MyApp", new DtRprFomMapper(2.0));
```

RTI Support

The MÄK RTI is included on all MÄK product CDs. It is link-compatible with the DMSO RTI NG. VR-Forces 2.1.1-ngc has been tested with DMSO RTI NG v3.2, and 4.0.

If you use the MÄK RTI, be sure that VR-Forces can find your FED file, and optional *rid.mtl* file, by putting them in the directory from which you are running, or by setting the environment variable RTI_CONFIG to the directory that contains them.

RTI Library Names

VR-Forces 2.1.1-ngc expects library names that match those of DMSO RTI NG v4. If you run VR-Forces 2.1.1-ngc with DMSO RTI NG v4, both the UNIX and PC versions work without any modification. If you want to run VR-Forces 2.1.1-ngc with DMSO RTI NG v1 or v2, you must make the following changes:

- For UNIX, you must use the libraries *libRTI-NGv2.so* or *libRTI-NGv1.so*, which are supplied by MÄK in the *./lib* directory. These libraries correspond to DMSO RTI NG v1 and v2. To use VR-Forces 2.1.1-ngc with RTI NG v1 or v2, copy the appropriate library to the DMSO RTI NG *lib* directory and rename it *libRTI-NG.so*. (You might want to rename the v3.x *libRTI-NG.so* library first.)
- For PCs, rename the libraries that come with RTI NG v1 or v2 to the names used by the RTI NG v4 libraries (*libRTI-NG.lib* and *libfedtime.lib*).

New Features and Updates

This release of MÄK VR-Forces has the following updates and new features:

- Support for Identification Friend from Foe (IFF)
- New set data request for setting the entity concealment appearance attribute
- New object-changed callback process.

Support for IFF

VR-Forces 2.1.1-ngc adds support for IFF. The IFF Controller models a NATO IFF transponder with settable modes and codes. The IFF controller has been added to the generic fixed-wing platforms in the *vrfSim.opd* file.

```
(iff-controller
   (component-descriptor-type "iff-controller-descriptor")
   (component-type "iff-controller")
)
```

The following sections describe the components and parameters.

Setting IFF in the GUI

To turn on the IFF transponder for a fixed-wing entity that is configured with an IFF controller:

1. Choose Entities → Set → IFF. The Set IFF dialog box opens.
2. Complete the Set IFF dialog box. The parameters correspond to the parameters described in the following sections.

IFF Controller

The IFF controller maintains IFF mode settings and transmits IFF data onto the network for the entity it is attached to. This controller registers for Set IFF data request messages, and sets its internal mode states as indicated in the message. No network data is sent until at least one mode is turned on.

component-type:	iff-controller
component-descriptor-type:	iff-controller-descriptor
Attaches to:	None.
Mounts on:	Any entity that specifies a vrf-moving-object-state-repository or a state repository derived from vrf-moving-object-state-repository.
Port interfaces:	None.
Model:	NATO Transponder.
Parameters:	Used to set initial states at startup. See Table 2.

Table 2: IFF controller parameters (Sheet 1 of 2)

Parameter name	Values	Default
system-name-enum	0 - 7 (see *disEnums.h*)	7
mode-1-capable	True/False	True
mode-2-capable	True/False	True
mode-3-capable	True/False	True
mode-4-capable	True/False	True
mode-5-capable	True/False	True
mode-S-capable	True/False	True
system-on	True/False	True
system-operational	True/False	True
mode-1-on	True/False	False
mode-1-damage	True/False	False
mode-1-malfunction	True/False	False
mode-1-code	00 - 77 (octal)	00

Table 2: IFF controller parameters (Sheet 2 of 2)

Parameter name	Values	Default
mode-2-on	True/False	False
mode-2-damage	True/False	False
mode-2-malfunction	True/False	False
mode-2-code	0000 - 7777 (octal)	2000
mode-3-on	True/False	False
mode-3-damage	True/False	False
mode-3-malfunction	True/False	False
mode-3-code	0000 - 7777 (octal)	2000
mode-4-on	True/False	False
mode-4-damage	True/False	False
mode-4-malfunction	True/False	False
mode-4-code	0000 - 4095	4095
alt-mode-4-valid	True/False	True
mode-5-on	True/False	False
mode-5-damage	True/False	False
mode-5-malfunction	True/False	False
mode-S-on	True/False	False
mode-S-damage	True/False	False
mode-S-malfunction	True/False	False
mode-S-tcas-i	True/False	False
emergency	True/False	False
flash	True/False	False
sti	True/False	False

DtIffControllerComponent

The *DtIFFControllerComponent* models a NATO IFF Transponder with settable modes and codes. This component will transmit IFF network updates if any of the IFF modes have been turned on. Once the component starts transmitting, it will continue sending updates even if all the modes are later turned off.

The *DtIffControllerDescriptor* specifies the default state values when the component starts up.

This component registers for *DtSetIffRequest* messages, so the IFF state can be changed on a per-entity basis by sending these messages the entity.

DtSetIffRequest

The *DtSetIffRequest* is a request for an entity with an IFF Controller Component to change its IFF settings to those specified in the message.

set-data-request-type: DtSetIffRequestType

parameters: See Table 3.

Table 3: DtSetIffRequest Parameters (Sheet 1 of 2)

Parameter	Description
system-on	True/False indicating overall state of system.
system-operational	True/False indicating overall state of IFF system.
mode-1-on	True/False indicating use of IFF Mode 1.
mode-1-damage	True/False indicating damage status of Mode 1.
mode-1-malfunction	True/False indicating malfunction status of Mode 1.
mode-1-code	00-77 octal code transmitted on Mode 1.
mode-2-on	True/False indicating use of IFF Mode 2.
mode-2-damage	True/False indicating damage status of Mode 2.
mode-2-malfunction	True/False indicating malfunction status of Mode 2.
mode-2-code	0000-7777 octal code transmitted on Mode 2.
mode-3-on	True/False indicating use of IFF Mode 3.
mode-3-damage	True/False indicating damage status of Mode 3.
mode-3-malfunction	True/False indicating malfunction status of Mode 3.
mode-3-code	0000-7777 octal code transmitted on Mode 3.
mode-4-on	True/False indicating use of IFF Mode 4.
mode-4-damage	True/False indicating damage status of Mode 4.
mode-4-malfunction	True/False indicating malfunction status of Mode 4.

Table 3: DtSetIffRequest Parameters (Sheet 2 of 2)

Parameter	Description
mode-4-code	pseudo-crypto value transmitted on Mode 4, or code 4095 indicating use of Alternate Mode 4.
alt-mode-4-value	status of Alternate Mode 4. (1 = valid, 2 = invalid, 3 = no response)
mode-5-on	True/False indicating use of IFF Mode 5C.
mode-5-damage	True/False indicating damage status of Mode 5C.
mode-5-malfunction	True/False indicating malfunction status of Mode 5C.
use-alt-mode-5	True/False indicating use of Alternate Mode 5.
mode-s-on	True/False indicating use of IFF Mode S.
mode-s-damage	True/False indicating damage status of Mode S.
mode-s-malfunction	True/False indicating malfunction status of Mode S.
mode-s-tcas-I	True: TCASI, False: TCASII
emergency	True/False indicating whether emergency code is transmitted.
flash	True/False indicating whether Squak/Flash/Ident is active.
sti	True/False indicating whether STI is active.

Notifying an Application when A *DtVrfObject* Changes

You can register callbacks for notification whenever some user-defined condition is met in a *DtVrfObject*. This mechanism can be used to check for changes in a *DtVrfObject*. It is done by registering a callback in conjunction with a *DtVrfObjectPredicate* object, on a *DtVrfObject*. The *DtVrfObjectPredicate* object contains a predicate member function that takes a *DtVrfObject* as an argument, and returns a Boolean value. The *DtVrfObject* periodically evaluates the *DtVrfObjectPredicate's* predicate member function, isTrue(). When isTrue() evaluates to true, it invokes the callback. You can define your own *DtVrfObjectPredicate* class to create a custom predicate function.

Predicate objects must be derived from *DtVrfObjectPredicate* (*vrfObjPred.h*).

Callback functions must have the following function signature:

```
typedef void (*DtVrfObjectPredicateCallbackFunction)(
   DtVrfObject* obj, DtVrfObjectPredicate* predicate, void* userData);
```

The combination of callback function and predicate object is registered with a *DtVrfObject* using its addVrfObjectPredicateCallback() function. It can be unregistered with removeVrfObjectPredicateCallback(). Remember to delete the *DtVrfObjectPredicate* once it is no longer needed.

You can use *DtVrfObjectPredicate's* to check for changes to an object's state. For example, suppose that you want to know when an object's altitude changes. In your predicate object, you would need to maintain the last value for altitude (the altitude value that the object had the last time the predicate function, isTrue() was invoked). In your callback function, you would update the stored value, to be used the next time the predicate function gets invoked.

```
#include"vrfObjPred.h"

class MyAltitudeChanged : public DtVrfObjectPredicate
{
   MyAltitudeChanged ();

   // This function tests to see if the given vrfObject's altitude is
   // different from our stored altitude.
   virtual DtBoolean isTrue(DtVrfObject* vrfObject);

   // Set/get our altitude
   virtual void setAltitude(DtReal altitude);
   virtual DtReal altitude() const;
   . . .
   protected:

   DtReal myAltitude;
};

DtBoolean MyAltitudeChanged::isTrue(DtVrfObject* vrfObject)
{
   DtVector localPosition = vrfObject->vrfState()->vrfKinematicState()
      ->localPosition();

   // The z component of local position is altitude – compare it to our
   // stored value.
   if(localPosition [2] != myAltitdue)
   {
      return DtTrue;
   }

   return DtFalse;
}
```

In your altitude changed callback, you need to update the predicate's altitude, so that the next time the predicate function, isTrue() gets invoked, the object's current altitude will be compared against its previous value.

```
void altitudeChangedCallback(
   DtVrfObject* vrfObject, DtVrfObjectPredicate* predicate, void*
   userData)
{
   DtWarn("Altitude changed for object %s\n",
      vrfObject->objectIdentifier().string());

   // Update the predicate's altitude with the object's current altitude

   DtVector localPositon = vrfObject->vrfState()->vrfKinematicState()->
      localPosition();

   predicate->setAltitude(localPositon [2]);
}
```

In your application, you would register your callback and predicate as follows:

```
// Create a predicate that checks for changes in altitude
MyAltitudeChanged checkAltitudeChanged = new MyAltitudeChanged();

// Register the callback with the predicate
vrfObject->addVrfObjectPredicateCallback(altitudeChangedCallback,
   checkAltitudeChanged, NULL);
```

Setting the Rate for Checking Changes to *DtVrfObjects*

For performance reasons, you may want to be able to control the rate at which *DtVrfObjectPredicates* get evaluated by a *DtVrfObject*. If for example, the test being performed in the predicate's isTrue() function is particularly computing-intensive, then you might not want to have it evaluated each tick. You can set the rate at which a predicate is evaluated by calling setTestInterval(). This will cause the *DtVrfObject* to only call the predicate's isTrue() at that time interval. By default, the interval is zero, and the predicate gets evaluated each tick.

For example, to register the previously defined altitude changed predicate and callback, to be evaluated once every 5 seconds:

```
// Create a predicate that checks for changes in altitude
MyAltitudeChanged* checkAltitudeChanged = new MyAltitudeChanged();

// Check for changes to altitude every 5 seconds
checkAltitudeChanged ->setTestInterval(5.0);

// Register the callback with the predicate
vrfObject->addVrfObjectPredicateCallback(altitudeChangedCallback,
   checkAltitudeChanged, NULL);
```

Getting Notified when a *DtVrfObject's* Vertices Change

You can register a callback to be invoked whenever an object's vertices change (the ordered list of vertices that make up an *DtVrfObject's* geometry, see section 3.2.2 "Object Geometry" *MÄK VR-Forces Developer's Guide*). The callbacks are invoked whenever a vertex is added, removed, or modified. Vertices changed callbacks must have the following function signature:

```
typedef void (*DtVrfObjectCallbackFunction)(DtVrfObject* obj,
   void* userData);
```

The callbacks are registered with a *DtVrfobject* using its addVerticesChangedCallback() member function. They can be unregistered with the removeVerticesChangedCallback() member function.

Documentation Updates

The list of parameters for *DtIfCreateVrfObject* on page C-8 of MÄK VR-Forces Developer's Guide is incorrect. Table 4 lists the correct parameters.

Table 4: *DtIfCreateVrfObject* parameters

Parameter	Description
vrf process address	A DtSimulationAddress identifying the particular VR-Forces back-end to create the object.
object name	A DtString used to uniquely identify the object. This string must be unique for all objects in the exercise.
object type	The *DtObjectType* of the new entity. *DtObjectType* is an extension of the DIS/RPR enumerated *DtEntityType* (from *objTypeEnums.h*).
force type	The *DtForceType* value for the new entity (from *disEnums.h*).
object label	A DtString associated with the new object. This string provides an additional means to identify an object without the strict uniqueness requirement of object name.
position	A *DtVector* that is the starting location of the object in local (database) coordinates. Set the position for those objects that have a single vertex (for example, waypoints, entities).
heading	A DtReal value that is the initial heading of the object in radians. A value of 0.0 is north and positive values are clockwise.
vertices	A DtList of the *DtVectors* that make up the object's geometry (i.e. the vertices that make up the points along a route). Note that position is just an alias for the first vertex. Do not call setPosition() if you have an object with multiple vertices (call addVertex() to set up the object's geometry instead).
projection	A DtBoolean flag indicating whether or not the Z elements in the object's vertex list have meaning. For example, phase lines do not have meaningful Z elements.
closed figure	A DtBoolean flag indicating whether the object's vertices make up a closed figure (i.e. does the last vertex connect with the first?). A control area is an example of an object that is a closed figure.
appearance	A *DtAppearance* indicating the initial appearance of the object (e.g. damage state, frozen status, concealed).
create subobjects	A DtBoolean flag indicating whether or not you wish to have subobject created. The parameter file may contain a list of subobjects. When building from the bottom up (for example, using Aggregate As or explicitly calling addSubordinate) then the subobjects in the parameter list should be ignored.

Bug Fixes

This release fixes the following problems:

- The back-end and Radar Warning Receiver examples were left out of the VR-Forces 2.1 release.
- VR-Forces was not able to display PvFltMetafileRecord databases correctly.
- Late joining front-ends did not show existing back-ends in the Back-ends list.

Known Problems

VR-Forces Release 2.1.1-ngc has the following known problems:

- The Go To Next and Go To Previous options on the Map Navigation menu do not work with aggregates. If you have a mixture of aggregates and individual entities, using these menu options works fine if there is a series of contiguously listed individual entities. Once the selection moves to an aggregate (11:1:0 ...) the entity list window loses a selection and the view in the Simulation View window doesn't change.
- Aggregates composed of fixed-wing entities do not carry out tasks as an aggregate.
- The Windows and IRIX versions of VR-Forces do not work together.
- If you are using VR-Forces on a multi-processor computer, set the affinity to single-processor. VR-Forces does not currently take advantage of multiple processors.
- If you are using a DTED database and "Projected False" is specified in his PvDted-MetafileRecord, entities do not execute movement tasks.
- The *world.gdb* file is based on a DTED database. If you use this database, some entities may not execute tasks. Some entities will not display properly.
- Batch scenarios do not automatically record to MÄK Logger files.
- When you aggregate a set of aggregates into a higher-echelon aggregate using the "Aggregate As" menu item, make sure that the aggregates being combined have been collapsed to their highest level. Failure to do so will aggregate the entities at the level they are currently displayed. For example, suppose that you want to aggregate two teams of two tanks each into a platoon. If at the time of the "Aggregate As" operation, one of the teams is expanded, displaying its constituent tanks, while the other team is collapsed, displaying a single team icon, the resulting platoon will be composed of two individual tanks and a team, instead of two teams.
- Phase line labels are displayed in the middle of the phase line rather than the beginning. To determine which side of a phase line is "left" for the purpose of the EntLeftOfLine condition, you will need to remember where you started each phase line.
- The missile icon may flash briefly origin of the database when a missile is fired.
- Surface and lifeform entities are not configured to fire weapons or be fired upon. You can customize the object parameters database entries to enable this.

- Repeated attempts to toggle the intervisibility fan may result in multiple intervisibility fans displayed on top of one another. It will not turn them off.
- The waypoint icon is currently being displayed centered on its location. According to MIL-STD-2525B it should be positioned with the lower tip of the positioned at the waypoint's location.
- Stealth control using the Stealth Control dialog box from VR-Forces does not work well. (The Stealth Attach and Teleport Stealth commands work.)
- Joysticks do not work correctly with DirectInput.
- If a front-end or back-end tries to load a FLT file more than once, it will crash.
- If you run multiple front-ends, the "extra" front-ends do not show the run or pause buttons depressed when the simulation state changes.
- The Berkeley Demo recording provided with the MÄK Logger does not work with this release of VR-Forces.
- The pseudo-aggregate formation controller ignores entities in a formation that are independently tasked when determining if the promotion order needs to be updated. Currently it has no way to determine if remote entities are independently tasked. One possible solution might be to listen to task and task complete messages for all the subordinates in order to maintain a local picture of what is going on.
- The list of parameters for DtIfCreateVrfObject on page C-8 of MÄK VR-Forces Developer's Guide is incorrect. The correct list of parameters is in the Documentation section of this release note.
- Front-ends that did not initiate a scenario cannot use canvas selections to complete dialog boxes, for example, selecting a waypoint for a move to waypoint task.
- You cannot add a right-click menu item with the plug-in interface.
- Larger scenarios, such as Berkeley scenario provided with VR-Forces, may exceed the capacity of the buffers on the network socket when running the HLA version of VR-Forces. To increase the buffer size, download the MÄK RTI, version 1.3.6a1, from:

  `ftp//ftp.mak.com/out/makRti1.3.6a1-ngc-win32.zip`

  Extract the zip file into your *vrforces-2.1.1-ngc/binRPR* directory. Then add the following line to your *rid.mtl* file (located in your MÄK RTI directory):

  ```
  (setqb RTI_socketReceiveBufferSize 65536)
  ```

This problem has been fixed for DIS within the VR-Forces code.

- The online help may not work well with more recent versions of Netscape. If the contents does not display, try clicking the contents button after the introductory topic finishes loading.