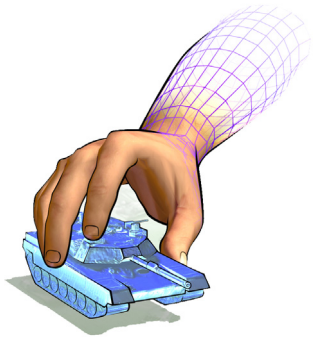

VR-Forces 3D Front End 2.0 Release Notes

This release note contains the following release-specific information for VR-Forces 3D Front End Release 2.0:

Document ID: VRF3D-2.0-2-090715

Systems Supported

VR-Forces 3D Front End 2.0 is compatible with VR-Forces 3.12. You cannot run it with previous versions of VR-Forces, nor will you be able to run it with any future VR-Forces maintenance releases (unless they are binary compatible) or feature releases (which will have their own compatible versions of the 3D Front End).

The 3D Front End is initially available for Windows XP and Vista. Please contact your MÄK salesperson about its availability on Linux. You can also sign up for the MÄK Product email list to receive announcements about MÄK releases. To sign up, go to http://www.mak.com/contact/makemail.php

A VR-Forces license includes the ability to run one front-end, either the 2D front-end or the 3D Front End. If you want to run both at the same time, you must purchase additional front-end licenses.

Unless otherwise stated in this document, all of the system and compatibility information in *VR-Forces Release Notes* applies to the VR-Forces 3D Front End.

3D Video Boards Supported

In general, VR-Forces 3D Front End 2.0 should support any board that claims to support OpenGL 2.0 and which has at least 256 MB of memory. The VR-Forces 3D Front End has been tested with NVidia GeForce graphics cards.

VR-Vantage Compatibility

The VR-Forces 3D Front End display is built using an internal MÄK build of VR-Vantage. All VR-Vantage libraries are included with the release, so it is not necessary, nor is it possible to link to a released version of VR-Vantage.

Installing VR-Forces 3D Front End 2.0

VR-Forces applications and plug-ins must be installed in the following order:

1. VR-Forces
2. VR-Forces 3D Front End
3. Optionally, B-HAVE Module for VR-Forces.

For a generic description of the VR-Forces 3D Front End installation process, please see the next section.

Installing the VR-Forces 3D Front End

The 3D Front End is installed separately from the base VR-Forces. It does not provide a complete VR-Forces installation. It installs files specific to the 3D Front End into the existing VR-Forces installation directory. These files include the executable file and required DLLs, configuration files, and 3D terrain databases.

! You must install the 3D Front End into an existing VR-Forces directory that contains the version of VR-Forces for which the 3D Front End was built.

To install the 3D Front End:

1. Install VR-Forces. Follow the procedures in Chapter 2, *Installing VR-Forces*, in *VR-Forces Users Guide*.
2. Run the installer for the 3D Front End. Use the appropriate procedure from Chapter 2, *Installing VR-Forces*, in *VR-Forces Users Guide* for your platform.

   In Windows, the installation wizard automatically chooses the default installation directory for the appropriate version of VR-Forces. If you installed VR-Forces to a different directory, change the installation directory in the 3D Front End installation wizard.

!

- On Windows, when the installation wizard notifies you that the installation directory already exists and asks if you want to install into this directory, click Yes.
- On Linux, you must install the 3D Front End from the same directory from which you installed VR-Forces. For example if you installed VR-Forces from */usr/local/mak*, you must install the 3D Front End from */usr/local/mak*.

New Features and Updates

VR-Forces 3D Front End 2.0 is in some respects completely different from VR-Forces 3D Front End 1.0. It is built on VR-Vantage, MÄK's new 3D visualization toolkit. (VR-Forces 3D Front End 1.0 was built using Presagis Vega Prime.) However many of the basic features are similar to those in VR-Forces 3D Front End 1.0. For a description of the many features of VR-Forces 3D Front End and a list of differences between the 3D Front End and 2D front-end, please see *VR-Forces 3D Front End Users Guide*.

Differences Between VR-Forces 3D Front End 1.0 and 2.0

Much of the basic functionality of the 3D Front End will be familiar to users of 3D Front End 1.0. However there are some important differences. Some of the most important new features are:

- All configuration is done through the GUI, not through MTL configuration files
- You can build your terrain dynamically at runtime by importing elevation data, raster images, a feature data. You can save the dynamically created terrains for quick loading in future sessions.
- The 3D Front End has a simplified, and yet highly flexible navigation model. All navigation is done from the keyboard using a layout similar to first person shooter games. However, you can remap the keys to suit your needs.
- View modes have been simplified. Keyboard navigation in the different view modes is more consistent and intuitive than the previous release.
- You can create multiple windows and channels through the GUI and configure them.
- Support for realistic head-up displays using GL Studio from DiSTI.
- Support for geocentric terrains from public data servers.
- The 3D Front End is based on VR-Vantage, which is built using Open Scene Graph, not Vega Prime.The 3D Front End includes all of the VR-Vantage and third party libraries required to extend or modify it. You do not need any additional licenses to modify the 3D Front End. (However you would need a VR-Vantage developers license if you wanted to use the supplied libraries to build a VR-Vantage-based application.)

3D Front End 1.0 Features not Supported by 3D Front End 2.0

The following features that were part of 3D Front End 1.0 are not included in 3D Front End 2.0. They may be included in future releases.

- StealthXR
- Aggregate display
- Hazard clouds
- Support for MetaFlight, VSB, and Blueberry 3D terrains
- Range volumes.

2D Front End Features not Supported in the 3D Front End

The following 2D front-end features are not supported in the 3D Front End, but were not listed in *VR-Forces 3D Front End Users Guide*:

- Emitters
- IFF
- Selection groups
- Setting units of measurement
- Object count toolbar
- Time multiplier toolbar
- Frame rate monitor
- Ability to click on terrain to retrieve soil type information.

Documentation

VR-Forces 3D Front End Users Guide has been completely rewritten to reflect the new 3D Front End design. The following information was not available prior to completion of VR-Forces 3D Front End documentation:

- Configuration of DDS textures. This informational supplements Chapter 11, Configuring 3D Windows and Channels, in *VR-Forces Configuration Guide*.
- VR-Forces 3D Front End API. This information supplements *VR-Forces Developers Guide* and *VR-Vantage Developers Guide*.
- Loading and saving multichannel configurations.
- Starting the VR-Vantage Distributed Code Generator.

Displaying DDS Textures Correctly

DDS is a bitmap format that is often used for textures. Sometimes these textures are coded so that they are upside down relative to the way VR-Forces 3D Front End expects them to be. In such cases, VR-Forces 3D Front End lets you flip the textures so that they display correctly. You can flip DDS textures as follows:

- On a per-model basis
- When you add a terrain patch
- When you display paged terrains.

Flipping DDS Textures for a Model

If a model is displayed upside down because its DDS textures are not what VR-Forces 3D Front End expects, you can specify that the model be flipped.

To flip DDS textures for a model:

1. Choose **Settings → Model Definitions**. The Application Settings dialog box opens to the Model Definitions page.
2. Optionally, filter the model definitions in the list.
3. Select the model for which you want to change the DDS parameter.
4. Set the flip DDS textures parameter to 1.

Flipping DDS Textures for a Terrain Patch

If a terrain patch has DDS textures that are upside down relative to VR-Forces 3D Front End's expectations, you can flip the textures when you load the terrain patch. This procedure does not affect paged terrains.

➤ To flip DDS textures for a terrain patch, when you add a terrain patch, select the FLIP DDS Textures check box in the Add Terrain Patch dialog box.

Flipping DDS Textures for Paged Terrains

If a paged terrain has DDS textures that are upside down relative to VR-Forces 3D Front End's expectations, you can flip the textures when the terrain gets paged in. This procedure does not affect terrain patches.

To flip DDS textures for paged terrains:

1. Choose **Settings → Render Settings**. The Application Settings dialog box opens to the Render Settings page (Figure 1).

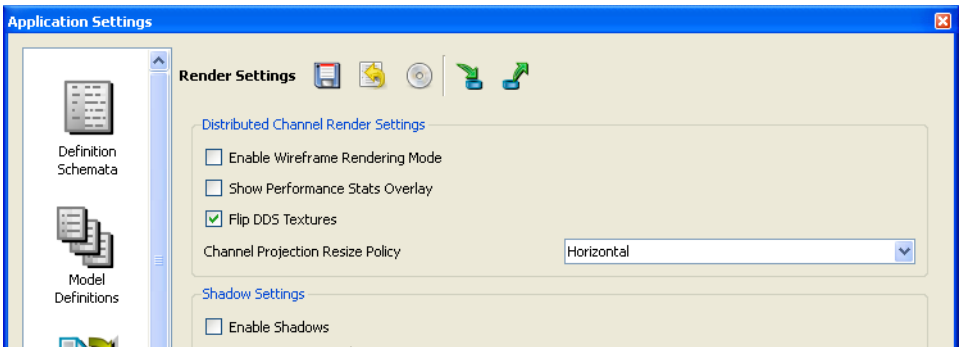

Figure 1. Render Settings page

2. Select the Flip DDS Textures check box.
3. Click Close.

The VR-Forces 3D Front End API

This section explains how to change or extend the VR-Forces 3D Front End-specific components of the 3D Front End.

The 3D Front End is built using the VR-Vantage Toolkit. All visualization functionality is from VR-Vantage. VR-Forces 3D Front End's CGF functionality is added using plug-ins. Unlike the 2D front-end and the VR-Forces 3D Front End back-end, you cannot rebuild the 3D Front End executable. If you want to change or extend the 3D Front End, you must write plug-ins that add the new or changed functionality.

The 3D Front End installation includes the header files and class documentation for VR-Vantage and the *VR-Vantage Developers Guide*. You can review this documentation to get a basic understanding of the VR-Vantage Toolkit and its plug-in architecture.

2D front-end plug-ins cannot be loaded into the 3D Front End. While you might be able to share some common code, a 3D Front End plug-in should be considered a separate development effort from a 2D front-end plug-in.

Plug-in Strategy

When you create a plug-in for the 3D Front End, a good strategy is to take a relevant example, modify the project settings, and add files that are needed to support your plug-in. All plug-ins must implement the initialization routine initDeModule() to enable the 3D Front End to recognize the plug-in as a valid 3D Front End plug-in. This will usually be composed of two files:

- The source file *.cxx* contains the initDeModule() method:

```
void initDeModule(makVrv::DtDe* de)
{
  //Make sure that the vrvCoreQt module is initialized.
  makVrv::vrvCoreQt::init(*de);
}
```

- The header file (included by the *.cxx* file) contains the appropriate exports:

```
//Get proper local export symbol
#include <vrf3DPluginQt/vrf3DPlugin.h>

//Setup proper plugin export symbol
#define DT_DE_PLUGIN_EXPORT_MACRO DT_DLL_3DPLUGINEXPORT

//Export plug-ins function void initDeModule(DtDe*);
#include <vrvCore/exportPlugin.h>
```

Adding a Task or a Set Data Request

The 3D Front End API allows developers to add new kinds of tasks and set data requests to the application. You can create custom dialog boxes for configuring them and add them into the appropriate menus.

The addTask and addSet examples contain a 3D Front End plug-in for adding a task or set data request to the 3D Front End (in a similar manner to a 2D front-end plug-in). These examples have four main classes:

- Menu item associated with the new action
- Creator that creates an instance of the dialog box class
- Dialog box class
- Logic class that fills in and sends the task, set data request, or simulation command to the VR-Forces simulation back-end.

For more information about the design patterns used to create dialog boxes, please see Section 8.6 “The Humble Dialog Pattern”, in *VR-Vantage Developers Guide*. For information about the task and set data request classes, see Chapter 10 “Tasks and Set Data Requests” in *VR-Forces Developers Guide*.

Adding Tasks or Set Data Requests to the Menu

The class *DtVrfTaskSetMenuItem* is a base class for creating new task or set menu action. Your new task or set menu actions should subclass from *DtVrfTaskSetMenuItem*. Please see *DtRetreatTaskAction.cxx* or *DtSetSpeedAndHeadingAction.cxx* for examples of new task and set data request actions.

The menu action determines which dialog box is displayed when the menu action is selected. This is done as follows:

1. The constructor assigns a unique ID to the dialog box creator to be displayed when the menu item is activated.

```
makVrf3D::makVrf3DQt::DtVrfTaskSetDialogManager::instance(de).
  addDialogCreator(myId, new DtTaskRetreatDialogCreator);
```

2. The createAction() method connects the unique menu item ID to the menu action.

```
registerMenuItem(makVrv::DtMenuItem::tr("Retreat..."), parent, myId,
  "TaskRetreat");
```

Once the action is defined, install it in the menu system by adding the menu action to the appropriate task or set menu. See the plug-in initialization modules in examples\addTask\addTaskPlugin.cxx and examples\addSet\addSetPlugin.cxx, for examples of how to install an action into the appropriate menu system.

```
makVrf3D::makVrf3DQt::init(*de);
makVrf3D::makVrf3DQt::DtTaskMenu& taskMenu =
  makVrf3D::makVrf3DQt::DtTaskMenu::instance(*de);

taskMenu.addMenuItemBefore(makVrv::DtMenuPath::mainMenu("DtTaskMenu").
  item("DtTaskRetreatAction"), makVrv::DtMenuPath::mainMenu(
  "DtTaskMenu").item("DtVrfTaskRotaryWingLandAction"),
  new vrfAddTaskExample::DtTaskRetreatAction(*de));
```

Finally, create the dialog box and logic classes. The dialog box class's main responsibility is to display the data. The logic class's main responsibility is to provide the dialog box with data to display and to execute the 'work' of the dialog box. In the case of task and set data request dialog boxes, the 'work' is to send the filled-out task or set data request message to a VR-Forces back-end.

Task and Set Data Request Dialog Classes

The base class from which all task or set data request dialog boxes must be derived is *DtBaseTaskSetDialogInterface*. However, in most cases it will make sense to subclass from one of the following derived classes that offer some additional functionality:

- *DtBaseComboTaskSetDialog*
- *DtBaseFilteredListViewTaskSetDialog*

DtBaseComboTaskSetDialog contains a basic combo box. Derive from this class when you want to provide a way for a user to pick from a range of options, such as choosing to "Retreat" or "Hold Ground" in the addTask example. To use this class, implement a logic class that subclasses from *DtBaseComboTaskSetLogic* and override the initializeItems() method to add your combo items, for example:

```
((DtBaseComboTaskSetDialog*)myInterface)->addItem(QObject::tr(" "));
((DtBaseComboTaskSetDialog*)myInterface)->addItem(QObject::tr("Yes"));
((DtBaseComboTaskSetDialog*)myInterface)->addItem(QObject::tr("No"));
```

DtBaseFilteredListViewTaskSetDialog contains a filtered list view. Derive from this class when you want to provide a way for the user to choose from a list of available entities or objects in the simulation. To use this class, implement a logic class that subclasses from *DtBaseFilteredListViewTaskSetLogic*. Override the initializeSelectionFilters() method to add your filters. In the following example code (taken from the logic class used with the "Move To" dialog box), we configure a list containing the waypoints available in the simulation:

```
void DtTaskMoveToLogic::initializeSelectionFilters()
{
  DtVrfSelectionObjectFilter waypoints(QObject::tr("Waypoints"),
  DtEntityType(DtPointObjectKind, -1, -1,
  DtPointObjectCategoryControlPoint, -1, -1, -1));
  addFilter(&waypoints);
  …
```

Task and Set Data Request Logic Classes

Logic classes work with the task and set data request dialog classes. Derived logic classes must implement the following methods:

- virtual void sendRequest() creates the task or set:

```
DtRetreatTask task;

task.init();
task.setRetreatKind(((DtTaskRetreatDialog*)myInterface)->
  retreatKind().toLatin1().constData());
```

```
sendTaskRequest(task);

if (myInterface)
{
  myInterface->close();
}
```

- virtual void setupStatement(const DtSimStatement* stmt) sets up the statement for editing:

```
if (stmt->type() == DtTaskStatementType)
{
  const DtTaskStmt* taskStmt = dynamic_cast<const DtTaskStmt*>(stmt);

  if (taskStmt && taskStmt->task())
  {
    DtRetreatTask* col = dynamic_cast<DtRetreatTask*>(taskStmt->task());

    if (col)
    {
      ((DtTaskRetreatDialog*)myInterface)->setRetreatKind(
  col->retreatKind().c_str());
    }
  }
}
```

- If you are going to add more graphical elements to the dialog box (such as a combo box), override the initializeItems(const std::vector<makVrv::DtSimEntry*>& data) method:

```
((DtTaskRetreatDialog*)myInterface)->addItem(QObject::tr(" "));
((DtTaskRetreatDialog*)myInterface)->addItem(QObject::tr(
  "Retreat Immediately"), "retreat-immediately");
((DtTaskRetreatDialog*)myInterface)->addItem(QObject::tr("Hold Ground"),
  "hold-ground");
```

Adding a Condition for Plans

The 3D Front End API allows you to add new kinds of conditional statements to the plan dialog boxes. The addConditional example shows how to create a plug-in example that adds a new conditional statement (‘is the entity facing north?’) to the 3D Front End.

For information about conditional statements classes used in the VR-Forces plan language, see Chapter 11 “Plans”, in *VR-Forces Developers Guide*.

The procedure for adding a conditional statement to the plan dialog boxes is as follows:

1. Add a new menu item to the conditional menu.
2. Implement the dialog box and logic classes to create and configure the conditional statement.

In this example, the conditional statement is added to the menu in *addConditionPlugin.cxx*. The *DtConditionalMenu* class represents the menu containing all of possible conditional statements you can configure in as part of a plan. It has public member functions that allow a new condition to be added before or after an existing item, or at the end of the menu. The following code snippet adds the new "DtNorthConditionalAction" condition, before the existing "RANDOM" condition in the menu:

```
makVrf3D::makVrf3DQt::DtConditionalMenu& conditionalMenu =
  makVrf3D::makVrf3DQt::DtConditionalMenu::instance(*de);

conditionalMenu.addConditionBefore("DtNorthConditionalAction", "RANDOM",
  new vrfAddConditionExample::DtNorthConditionalCreator);
```

All conditional statement dialog boxes must be derived from *DtBaseEntityConditionalExpressionDialog*. Allcondition statement logic classes must be derived from *DtBaseEntityConditionalExpressionLogic*. The logic must implement the createConditional(), conditionalExpression(), and setupConditional() methods to be considered complete. See the code for the *DtNorthConditionalLogic* class in *DtNorthConditional.cxx* for an example of how to implement these methods

Adding an Interface Content Message

The 3D Front End API allows developers to add new kinds of interface content (that is, VR-Forces-specific message content) to the application. Interface content messages are used to send a wide variety of VR-Forces-specific content between VR-Forces applications, such as scenario management information. See Chapter 8 "Messages", in *VR-Forces Developers Guide* for information about the interface content classes used in VR-Forces.

The interfaceContent example shows how to add a custom interface content message to the 3D Front End, and how to send and receive that message. While the example shows how to access this content through a menu item, the use of a menu item in this case is incidental. Interface messages can be sent or received in a variety of contexts in the front-end.

To add a new type of interface content:

1. Define the message (defined in *myInterfaceContext.h* and *myInterfaceContext.cxx*)
2. Add it to the factory (*interfaceContentPlugin.cxx*)

```
makVrf3D::DtVrfSimMessager::instance(*de).
  addInterfaceContentCreatorFcn(MyInterfaceContentType,
  MyInterfaceContent::creator);
```

3. Set up a callback to receive the message (*interfaceContentPlugin.cxx*)

```
makVrf3D::DtVrfSimMessageHandler::instance(*de).
  registerSimMessageCallbackFor(MyInterfaceContentType,
  &vrfInterfaceContentExample::DtInterfaceContentMenuItem::
  processInterfaceMessage, 0);
```

Adding or Removing a Menu Command

The 3D Front End API lets you add or remove menu items from the GUI. There are a couple of different approaches to adding and removing menu items:

- C the specific menu classes that contain those items (for example, *DtTaskMenu*, *DtSetMenu*). Choose this approach when the action is used in multiple places, such as the as the Task, Set, Create, Conditional, and Simulation Command menus. In this case, you must register menu commands with those classes so that they can be used at the appropriate place on demand.
- C the menu configuration to add or remove a new menu item (as is done in the interface content example). Menu configurations (*DtMenuConfiguration*) specify the layout and content of menus (see Section 3.1.1 “Menus” in *VR-Vantage Developer’s Guide* for more information). Choose this approach when you know there is just one specific place in the menu system that needs to be modified. Menu items that are only available on the main menu bar (such as Simulation Control, File action, and so on) can be added to the menu system in this way during plug-in initialization.

To remove a menu command, you must connect to a signal that is sent by the application object before the menu is assembled. This ensures that the entire system’s menu structure has been preset and is accessible. You cannot remove a menu command before it has been added to the menu system. To register for this signal, make the following connection in your plug-in initialization:

```
makVrf3D::DtVrf3DApplication* igApp =
dynamic_cast<makVrf3D::DtVrf3DApplication*>(makVrv::DtVrvApplication::
  findFromIg(*de));

if(igApp)
{
  igApp->signal_preAssemble.connect(boost::bind(
  &DtMenuPreAssemble, _1, _2));
}
```

In the function that gets invoked upon receipt of this signal, modify the menu configuration as desired. Items in a menu configuration are referred to as ‘menu paths’, and they may represent either a submenu or an individual menu entry. The *DtMenuConfiguration* class has public member functions for adding and removing ‘menu paths’. In the following example, we add a separator, followed by a new “DtVrfVrlinkObjectInformationItem” entry to the “DtEntity” menu:

```
static void DtMenuPreAssemble(makVrv::DtDe& de,
  makVrv::DtMenuConfiguration& configuration)
{
  makVrv::DtQtMenuAssembler& qtAssembler =
    makVrv::DtQtMenuAssembler::instance(de);

  configuration.addMenuPath(makVrv::DtMenuPath::mainMenu(
    "DtEntityMenu").separator("100"));
  configuration.addMenuPath(makVrv::DtMenuPath::mainMenu(
    "DtEntityMenu").item("DtVrfVrlinkObjectInformationItem"));
}
```

Adding or Removing Menus

The 3D Front End API lets you add or remove menus.

To add a menu, register the menu creator function with *DtQtMenuAssembler* (the class responsible for directing the construction of the menu) and create the Qt menu in the menu class. For example, to add the menu "MyMenu", in your pre-assemble callback you would do the following:

1. Get access to the menu assembler to register your menu:

```
makVrv::DtQtMenuAssembler& qtAssembler =
  makVrv::DtQtMenuAssembler::instance(de);
```

2. Register your new menu:

```
qtAssembler.registerMenu(new DtMyMenu);
```

   Your menu class might look like:

```
class MyMenu : public makVrv::DtMenu
{
public:

  //! \brief CTOR
  MyMenu ();

  //! \brief DTOR
  virtual ~ MyMenu ();

  //! Build a menu.
  virtual QMenu* createMenu(makVrv::DtDe&, QWidget* parent);

};

MyMenu:: MyMenu (): makVrv::DtMenu("MyMenu ")
{
}

MyMenu::~ MyMenu ()
{
}

QMenu* MyMenu::createMenu( makVrv::DtDe&, QWidget* parent )
{
  QMenu* menu = new QMenu(QMenu::tr("&My Menu"),parent);
  return menu;
}
```

3. Add menu commands:

```
configuration.addMenuPath(makVrv::DtMenuPath::mainMenu(
  "MyMenu").item("MyMenuItem"));
```

4. Add your menu command to the assembler:

```
qtAssembler.registerMenu(new MyMenuItem(de));

class MyMenuItem : public makVrv::DtMenuItem
{
```

```
public:
  //CTOR
  MyMenuItem (makVrv::DtDe& de);

  //DTOR
  virtual ~ MyMenuItem ();

  virtual QAction* createAction( makVrv::DtDe&, QWidget* parent);

  //! OVERRIDE THIS METHOD TO IMPLEMENT ACTION
  virtual void on_triggered();

protected:
  makVrv::DtDe& myDe;
};

MyMenuItem::MyMenuItem(makVrv::DtDe& de) :
  makVrv::DtMenuItem("MyMenuItem"), myDe(de)

  MyMenuItem::~MyMenuItem()
{
}

QAction* MyMenuItem::createAction( makVrv::DtDe&, QWidget* parent )
{
  QAction* act = new QAction(parent);
  act ->setText(makVrv::DtMenuItem::tr("My Menu Item"));

  return act;
}
```

Adding or Removing a Toolbar

The 3D Front End API lets you add and remove toolbars from the 3D Front End. The unitStatus example shows how to add a toolbar. (Please refer to the online class documentation and example source for more information.)

To remove a toolbar, implement the pre-menu assemble callback and get a reference to the user interface assembler:

```
makVrv::DtUserInterfaceAssembler& uia =
  makVrv::DtUserInterfaceAssembler::instance(*de);
```

Destroy the builder with the particular toolbar. The toolbar is usually the name of the class:

```
uia.destroyBuilder("DtSceneGraphTreeWidget",
uia.findMenuBuilder("DtSceneGraphTreeWidget"));
```

Loading and Saving Multichannel Configurations

To save a multichannel configuration:

1. Start the VR-Forces 3D Front End and one or more VR-Vantage Display Engines.
2. In the Display Engine Configuration Editor, connect the display engines.

3. Configure the display engines with the windows and channels that you want them to have.
4. For each display engine in the Display Engine Configuration Editor:
   a. Select the display engine.
   b. Click the Save Display Engine Configuration icon.
   c. Type and name for the configuration.
   d. Click Save.

To load a multichannel configuration:

1. Start the VR-Forces 3D Front End and one or more VR-Vantage Display Engines.
2. In the Display Engine Configuration Editor, connect the display engines.
3. For each display engine in the Display Engine Configuration Editor:
   a. Select the display engine.
   b. Click the Load Display Engine Configuration icon.
   c. Select the configuration you want to load.
   d. Click Load.

Starting the Distributed Code Generator

VR-Vantage Developers Guide explains how to start the Distributed Code Generator from the command line. On Windows, you can also start it from the Start menu, as follows:

➤ Choose **Programs** → MÄK **Technologies** → **VR-Vantage x.x** → **Tools** → **VR-Vantage Distributed Code Generator**.

Known Problems

VR-Forces 3D Front End Release 2.0 has the following known problems:

- VR-Forces 3D Front End does not include translation files.
- The 3D Front End does not prompt you to save changes to model definitions when you exit.
- When you create a DI-Guy character, a soldier is always displayed with the cursor until you click to place the entity. At that point the correct DI-Guy model gets displayed.
- Clicking to select an entity can be difficult when there are control objects between the observer and the entity. This can happen when you are zoomed in close to the entity. You can overcome this and select the entity by clicking inside the entity's location circle.
- Terrain dragging does not always work as expected when you are zoomed in very close to the terrain (it may be difficult to reach distant points on the terrain). The workaround is to zoom out and then perform terrain dragging.
- Height above terrain lines associated with rotary-wing entities tend to wobble slightly in the display.
- When you load a geocentric terrain, clouds are displayed in the center of the earth.
- Sometimes if you use **Alt**+**Tab** to switch applications, VR-Forces 3D Front End does not receive the **Alt**-Up message. The result is that keyboard input may not work. To get the keyboard to work properly, press **Alt** several times.
- SpeedTrees do not align correctly on geocentric terrains.
- If a graphics card does not support shaders, you may receive the following error message when you load a terrain that uses them:

  ```
  Shader [Branches] validation failed
  ```

  To work around this problem, set the SpeedTree Performance Profile to Disabled, as follows:

  a. Choose Settings → SpeedTree Settings. The Application Settings dialog box opens to the SpeedTree Settings page.

  b. In the Performance Profile drop-down list, select Disabled.

- There may be some discrepancies between terminology and title descriptions in *VR-Vantage Users Guide* and the final GUI.
- On external display engines, particle effects, such as smoke and trailing effects, sometimes restart spontaneously.

Using the Alt Key Under Linux with the Gnome Window Manager

By default, the Gnome window manager on Linux uses the Alt key as a modifier that lets you move a window by clicking and dragging anywhere on the window (as opposed to only being able to use the title bar). This default behavior prevents you from using the Alt key to set an entity's altitude in the 3D Front End.

To set the altitude of entities in advanced creation mode either use the mouse wheel or change the behavior of the Alt key.

To change the Gnome setting:

1. Choose **Applications** → **Preferences** → **Window**.
2. Change the "Movement Key" setting to "Super (or Windows Logo)". It is not recommended to change this setting to "Control" because that will also interfere with advanced creation mode.