Rational Software Corporation

Conversion Guide

ObjecTime Developer to Rational Rose® RealTime

Version: 2002.05.20

Part Number: 800-025120-000

Windows/Unix

Rational®
the software development company

support@rational.com
http://www.rational.com

Part Number: 800-025120-000

Version Number: 2002.05.20

Contents

Preface

Contents

This manual describes how to convert an existing model from ObjecTime Developer (C++ or C) version 5.2 or 5.2.1 to the latest Rational Rose RealTime.

This chapter is organized as follows:

Audience

This guide is intended for all readers, including managers, project leaders, analysts, developers, and testers.

Other Resources

- Online Help is available for Rational Rose RealTime.

  Select an option from the **Help** menu.

  All manuals are available online, either in HTML or PDF format. To access the online manuals, click **Rose RealTime Online Documentation** from the **Start** menu.

- For more information on training opportunities, see the Rational University Web site: http://www.rational.com/university.

Contacting Rational Technical Publications

To send feedback about documentation for Rational products, please send e-mail to our Technical Documentation Department at techpubs@rational.com.

Contacting Rational Technical Support

If you have questions about installing, using, or maintaining this product, contact Rational Technical Support.

Your Location	Telephone	Fax	E-mail
North America	(800) 433-5444 (toll free) (408) 863-4000 Cupertino, CA	(781) 676-2460 Lexington, MA	support@rational.com
Europe, Middle East, Africa	+31 (0) 20-4546-200 Netherlands	+31 (0) 20-4546-202 Netherlands	support@europe.rational.com
Asia Pacific	+61-2-9419-0111 Australia	+61-2-9419-0123 Australia	support@apac.rational.com

Note: When you contact Rational Technical Support, please be prepared to supply the following information:

- Your name, telephone number, and company name
- Your computer's make and model
- Your computer's operating system and version number
- Product release number and serial number
- Your case ID number (if you are following up on a previously-reported problem)

ObjecTime Developer to Rose RealTime Conversion

1

Contents

This chapter is organized as follows:

Overview

This document describes how to convert an existing model from ObjecTime Developer (C++ or C) version 5.2 or 5.2.1 to the latest Rational Rose RealTime. You must have the following documents before proceeding with your model conversion:

- Rational Rose RealTime Toolset Guide
- C++ or C Language Add-in, Getting Started and Language Guides
- Team Development Guide

These documents contain information and the steps required to complete the model conversion. Models converted from ObjecTime Developer (OTD) into Rational Rose RealTime format cannot be converted back into OTD. Only one-way conversion is supported.

Important Features of Rational Rose RealTime

Upgrading your model from OTD allows you to access the following new features available in Rational Rose RealTime:

- Class modeling
- Interaction modeling (collaborations)
- Component and deployment modeling
- Building of libraries
- Increased type safety of message sends
- Increased support for integration of external libraries and 3rd party code
- UML support

This is a brief summary of the features available in Rational Rose RealTime, please refer to the "What's New" topic in the *Installation Guide - Rational Rose RealTime* for a complete list.

Steps for Converting a Model

To convert from OTD to Rational Rose RealTime:

1. *Exporting an ObjecTime Developer Model to Linear Form* on page 3
2. *Loading Linear form into Rational Rose RealTime* on page 10
3. *Temporarily Adding the Model to Source Control (optional)* on page 17
4. *Building and Running a Model in Rational Rose RealTime* on page 19
5. *Organizing for Team Development and Source Control* on page 20
6. *Configuring the Build Process* on page 22

Each step is explained in detail with easy to follow instructions. The conversion will be easier if you take your time and learn the basics of Rational Rose RealTime before starting.

See the Rational Rose RealTime on-line help for a list of the available learning material. We recommend that you complete the following:

- Run through the Quickstart and Card Game tutorials. This is an excellent way to learn Rose RealTime
- Load, compile, and run some of the example models included with Rational Rose RealTime. You can find these in `$ROSERT_HOME/Examples/Models`.

Note: $ROSERT_HOME is used throughout this document. It represents the directory in which Rose RealTime is installed.

You can also contact your sales representative if you are interested in attending a Rational Rose RealTime course.

Exporting an ObjecTime Developer Model to Linear Form

Rational Rose RealTime only imports linear form files from OTD 5.2 and 5.2.1. Other types of files, such as binary .update or .context files cannot be imported directly into Rational Rose RealTime.

Before you Convert a Model

Before you convert an ObjecTime Developer model to Rational Rose RealTime, you must consider the following:

- *What version of OTD are you using?* on page 4
- *What is your patch level?* on page 4
- *Is the Model Under Source Control?* on page 5
- *Have you configured or customized the C or C++ Services Library?* on page 6
- *Does the Model Compile and Run Correctly?* on page 6

Exporting a Model

- See *Exporting a Model from ObjecTime Developer* on page 6

After Exporting a Model

After exporting a model to Rational Rose RealTime, you may want to consider the following:

- *Loading Linear form into Rational Rose RealTime* on page 10
- *OTD Model Considerations* on page 9

What version of OTD are you using?

If you are using a pre-5.2 version of OTD, you will first need to open your model in OTD 5.2 or 5.2.1 and save it. For detailed information on the differences between OTD 5.2/5.2.1, and earlier versions of OTD, please see the Model Upgrade/Conversion section and the Changes in Developer 5.2.1/5.2 section of the OTD 5.2/5.2.1 *Getting Started Guide*.

What is your patch level?

Before you can export your model to a Rational Rose RealTime compatible format, your OTD session file (the .otd file) must have the correct patch level. We recommend that all generic patches be applied to your 5.2/5.2.1 session, then apply the **RRT04.RRTExport.patch** to your OTD 5.2/5.2.1 session to enable exporting a model for conversion.

Note: The export patch is meant for exporting a model only. It should not be used for continued development work in OTD. You should apply the export patch without saving your session, then once the model is exported, abandon your session and restart.

If you want to reuse OTD requirements, see *Converting OTD Requirements* on page 8.

The patches can be found on the Rational Customer Support web site at **www.rational.com/support.** You will require your account and password to access the customer-only portion of the site.

To install a patch follow these instructions:

1. From the main **OTD** menu, click **Browsers > Open Workspace Browser** if a workspace browser is not currently open.
2. Determine the directory path of location of the patch files, for example: %OBJECTIME_HOME%\specials on NT or $OBJECTIME_HOME/specials on Unix.
3. From the main OTD menu, select **Browsers > Open Directory Browser**.
4. Type the name of the directory path.

   A directory browser displays with each patch file identified with a + next to it.

Figure 1 Example patch file shown in a directory browser

5 Drag each patch file from the **Directory Browser** window to your **Workspace Browser** window.

To confirm that the patch is installed, do the following:

6 From the main OTD menu, click **Help > About ObjecTime**.

The Patch Level should display RRT04 and/or rf52 or rf521, and may show other pre-existing patches as well.

Is the Model Under Source Control?

For a major part of the conversion, you will be working with a model not under source control. Rational Rose RealTime can only load a linear form file that must contain the entire OTD model.

Therefore, to start the conversion of an OTD model stored in source control, you will have to export a non-version controlled **baseline** of your model as a linear form file.

After the model is loaded, built, run, and tested in Rational Rose RealTime, you can then proceed to place the new Rational Rose RealTime model under source control. **It is highly recommended that you do not convert a model that is in the middle of a development cycle**. Instead, wait until you have a stable baseline that you can convert, convert to Rational Rose RealTime, learn Rational Rose RealTime, then continue development of the converted model.

As a result of many underlying file storage differences between OTD and Rational Rose RealTime the OTD files that were in source control will differ from those added to source control for the Rose RealTime model. Thus, file history will be lost.

Note: Some tools, like ClearCase, allow you to create hyperlinks between files. This enables you to retain traceability between source controlled files for the OTD model and those in the new converted Rational Rose RealTime model.

For details regarding source control, see *Organizing for Team Development and Source Control* on page 20.

Customized Library Interface Scripts

If you have customized your library interface scripts, consider why the modifications were made and review the current scripts to determine if additional customization is required. The library scripts changes are minimal, such that merging changes should not be too difficult.

Have you configured or customized the C or C++ Services Library?

Any customizations or configurations to the Services Library will have to be migrated to the current release of the Rose RealTime Services Library.

Migrating Changes

After changes have been identified, migrate them into the Rose RealTime Services Library. The architecture and file structure of the Services Library is the same as 5.2.1.

For minor problems migrating customizations or configurations, contact Rational Customer Support. For all other problems migrating your custom changes contact your sales representative to arrange for consulting services to assist in the migration.

Does the Model Compile and Run Correctly?

Perform a final test of the OTD model by compiling and running the model. Ideally, you would run full set of regression tests to confirm that everything works before migrating it to Rational Rose RealTime.

To minimize potential problems, the compiler and linker referenced in your **OTD Configuration** should be the same one you use in Rational Rose RealTime. It will reduce the number of unknowns when compiling the model in Rational Rose RealTime.

Note: Rational Rose RealTime does not support Simulation mode. Ensure that your model compiles and runs in the TargetRTS mode.

Exporting a Model from ObjecTime Developer

To export a model to the Rational Rose RealTime compatible linear form format:

1. Select the model in the Workspace browser.
2. Open a **Model** browser on the model.
3. From the main model menu, select **Update > Export**.

4 Select **Rose RealTime Linear Form** as the File Format.

If the Rose RealTime Linear Form option is not available it is because you have not applied the correct patch lineup to ObjecTime Developer. See section *What is your patch level?* on page 4 for help on applying the correct patch lineup.

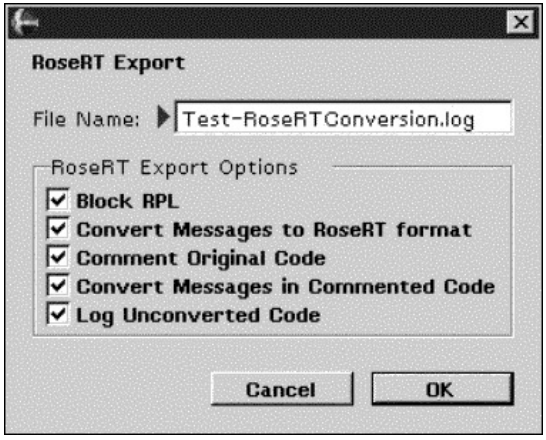
5 Accept the default file name, or rename it. The file name must have the **.lf** file extension.

6 Select **OK**.

The **RoseRT Export** dialog displays.

This dialog contains a list of useful export functions. Select the desired functions and click **OK**. A description of each function is listed below.

Figure 2 Exporting



Block RPL

When this option is selected all RPL content will not be exported into the linear form. For example, RPL actor classes and RPL methods on Data classes will be blocked. Since RPL does not exist in RoseRT, it does not make sense to export RPL content into the linear form for the purposes of migrating. If you RPL content is required after the migration, you must convert it to C or C++ prior to migrating. To help identify the blocked code, a report is updated in the specified Log file.

Convert Messages to RoseRT Format

When this option is selected port messages in the model will be converted to RoseRT format. For example, **port.send(sig)** is converted to **port.sig().send()**. This removes the need to manually convert messages. This will only be seen in the linear form after an export operation; it does not change the code in the OTD model.

See **Appendix A** for a list of port message conversions.

Comment Original Code

This option is only available when the **Convert Messages To RoseRT Format** option is selected. When selected, if a port message is converted, the toolset leaves the original OTD code in the linear form as a comment.

Convert Messages in Commented Code

This option is only available when the **Convert Messages To RoseRT Format** option is selected. When selected, port messages inside comments are also converted to RoseRT format.

Log Unconverted Code

When this option is selected, a report is maintained in the specified log file. It contains information about code that the toolset identifies as incorrect for RoseRT, and is unable to automatically convert. You will have to manually correct the code after the model is imported into RoseRT. The log file does not contain this code. The log contains a list of items that the toolset cannot automatically convert, as well as the number of times each code segment appears in the model.

See **Appendix B** for a list of code segments that cannot be converted automatically.

The linear form file and the conversion log file will be written to the directory containing your **.otd** session file.

Converting OTD Requirements

Requirements captured in OTD models can be converted through a requirements-specific patch for 5.2 and 5.2.1. An HTML file will be generated that will contain the actual requirements from the OTD models. Links to these requirements will be converted when the actual model is imported into Rational Rose RealTime. The HTML requirements file is stored outside of the Rational Rose RealTime toolset. Place the file in your configuration management library for storage purposes.

To migrate existing requirements from OTD to Rational Rose RealTime:

1 Apply the **rf52.reqFwdToRoseRT.patch** or **rf521.reqFwdToRoseRT.patch** to OTD.

Note: The export requirements patch should not be used for continued development work in OTD. You should apply the export patch without saving your session, then after the model is exported, abandon your session and restart.

2 Open a browser on your requirements update.

3 Select **Output** from the Requirements menu.

4 Select the **Export to File** radio button.

5 Name the output file the same name as your requirements update, with .html as the file extension. For example, if your requirements update is called MyRequirements, then the output file must be named MyRequirements.html.

6 Click **OK**.

OTD produces an HTML file in the same directory that contains your .otd session file. This file will contain an HTML-formatted version of all fields of all individual requirements within that requirements update. All requirements are tagged with their name, which will enable linking to them from Rational Rose RealTime.

OTD Model Considerations

When converting a model, you must consider the following:

SimulationRTS and EmulationRTS

If you only run your model using the **SimulationRTS** or **EmulationRTS**, create a TargetRTS configuration in OTD for your intended target, and ensure that your model correctly compiles and runs in that environment. Rational Rose RealTime only supports the TargetRTS.

Note: Building **SimulationRTS** or **EmulationRTS** models in Rational Rose RealTime is not supported. Trying to do so will result in build errors reported to the build Output window.

Batch Mode

OTD batch mode is not supported in Rational Rose RealTime. You will need to manually convert the functionality that exists in any OTD batch scripts that you currently have. The Rational Rose RealTime Extensibility Interface (RRTEI) includes a powerful scripting capability that can be used to perform batch operations.

To learn how to convert build processes to Rational Rose RealTime see *Configuring the Build Process* on page 22.

RPL

RPL is not supported in Rational Rose RealTime. If you use RPL as the programming language for any of your actors or data classes, you will need to convert these to C++. For help on converting RPL actors to C++ or C, please see the Converting RPL actor classes to C++ section in the OTD C++ Guide.

You will know if there are any RPL actors in your model when you compile in OTD for the TargetRTS and you receive the following compile-time error message:

Error: Attempt to compile an RPL class

If you do import a model containing RPL actors into Rational Rose RealTime, the capsule will be created, will have a Language property of RPL, and all code will be retained as the same ASCII text that it originally was. If you try to compile this in Rational Rose RealTime, you will get a variety of arbitrary error messages back from the C++/C compiler.

TestScope

Models that have packages which were created using TestScope can be exported and loaded into Rational Rose RealTime. The test framework packages will be converted. Rational Rose RealTime has equivalent TestScope functionality called Rational Quality Architect that can run and generate test frameworks.

Loading Linear form into Rational Rose RealTime

Rational Rose RealTime only imports linear form files (.lf) from OTD 5.2/5.2.1. Other types of files, such as binary .update or .context files cannot be imported directly into Rational Rose RealTime.

Before Importing

Loading a Model in Rational Rose RealTime:

After you Import

Have you properly installed Rational Rose RealTime?

Confirm that you correctly installed Rational Rose RealTime. Try running Rational Rose RealTime to familiarize yourself with the new interface. If you haven't already, this would also be a good time to review the *Installation Guide - Rational Rose RealTime* for a quick review of Rational Rose RealTime.

Are your Services Libraries available?

Refer to the I*nstallation Guide - Rational Rose RealTime* to see if the Services Libraries you were using with OTD are shipped with Rational Rose RealTime.

If your Services Libraries are not included with Rational Rose RealTime, contact your sales representative.

Is your Source Control tool supported?

Refer to the *Installation Guide - Rational Rose RealTime* to see if the Source Control tool you were using with OTD is supported with Rational Rose RealTime.

If your Source Control tool is not included with Rational Rose RealTime, contact your sales representative.

Do you use the External Layer Service (ELS)?

The External Layer Service (ELS) is not supported in Rational Rose RealTime. If you have been using the ELS to communicate between your OTD model and non-OTD components such as legacy C/C++ applications or Java applets/applications, you will need to create your own socket interface or use Connexis.

For information on creating your own socket interface, please refer to the Socket Interface Example shipped with Rose RealTime.

For information on Connexis, please see http://www.rational.com/products/

Can you build sample models in Rose RealTime?

It is very important that you confirm that your environments are properly configured before building your imported OTD model.

We strongly recommend that you load, run, and test one or more test models in your intended target environments (host and target boards) before importing your OTD model. This is to confirm that your compiler, linker, and target boards are correctly configured with Rational Rose RealTime.

At a minimum, you must do the following before importing your model:

- Do the Quickstart and Card Game tutorials. This is an excellent way of learning Rose RealTime.
- Load, compile, and run some of the example models included with Rational Rose RealTime. You can find these in $ROSERT_HOME/Examples/Models.

Loading the Linear form into Rational Rose RealTime

To open the .lf version of your model into Rational Rose RealTime:

1. Set your default language in Rational Rose RealTime by open the **Options** dialog and clicking **Tools > Options**, then click the **Language/Environment** tab.
2. If your OTD model is in C++, then set **Default Language** to **C++** and **Default Environment** to **C++ TargetRTS**. If your OTD model is in C, then set the **Default Language** to **C** and the **Default Environment** to **C TargetRTS**.
3. From the main menu, click **File > Open**.
4. In the directory browser, change the file type to **Linear Form (*.lf)**, and navigate to the directory that contains your .lf file.

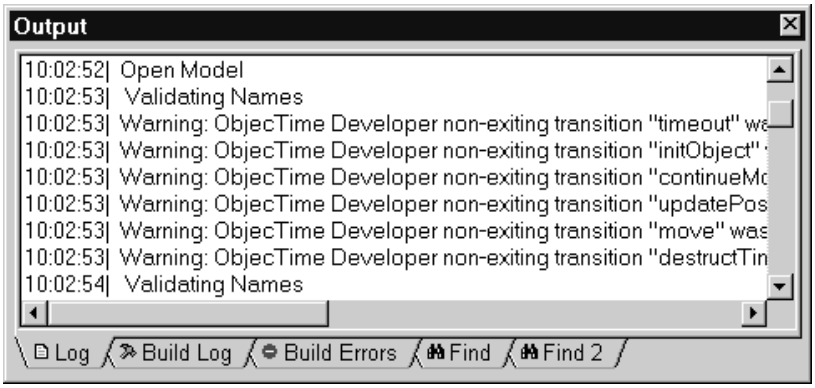
5 Select the file to load.

6 Click **Open**.

The model is converted to Rational Rose RealTime. During the conversion process, warnings appear in the Output window. The warnings help you identify potential problems, and may also include important notes about the conversion of model elements.

Note: Review all warnings because some warnings may require action.

Figure 3 Example log output



Reviewing the Log

After the conversion completes, review the output log and browse the model before attempting to build your model. Warnings inform you of conversion details that you should review, such as the toolset making a change that may impact the behavior of the model.

The following is a list of warnings and messages that may occur:

- *Non-Exiting Self Transitions* on page 14
- *'X' Upgraded* on page 14
- *Classes in Multiple Packages* on page 14
- *Updating 'X' Model Properties* on page 14

Non-Exiting Self Transitions

If you open a .lf file that has OTD non-exiting self transitions, you will get the following type of warning message which you can view in the log:

```
16:46:30|  Warning: ObjecTime Developer non-exiting transition
"MyTrans" was made internal to state "Ready" in class "Mine"; semantic
changes could be introduced.
```

In OTD, non-exiting transitions and internal self-transitions have the identical semantics, however at run-time if both transitions have the same trigger, the internal self-transition would always be triggered instead of the external non-exiting. Since both kinds of transitions have the same semantics, in Rose RealTime, non-exiting transition have been removed and replaced by internal to the state self transitions. If the transitions are now located at the same state hierarchy level, and if two self-transitions (internal to a state) have the same trigger, at design time you cannot determine which transition will be triggered. You should review the transitions that have been made internal to a state, and ensure that there are no semantic changes to your model.

Note: Having two transitions originating from the same state with the same triggers should be avoided since the transition triggering order is not known at design time.

'X' Upgraded

This is an information message only, and is expected when loading a previous version model into the toolset. The message indicates that a certain model element has been upgraded with the new C++ or C language model properties.

Classes in Multiple Packages

In OTD a model entity can belong to more than one package, whereas in Rational Rose RealTime, it cannot. When Rational Rose RealTime imports such a model, it will drop all but one of the package references from a capsule and log a message for each.

You should decide which is the primary package for each element which used to belong to multiple packages. Used the log to review each and move if necessary.

Updating 'X' Model Properties

This is a typical Log message. It indicates that add-in property sets are being loaded. You should see C++ and possibly C property sets being updated.

Understanding the Conversion Mappings

The Rose RealTime user interface and modeling language differs from the one you are used to in OTD. When browsing your model, you will notice that objects have moved and that there are new modeling elements, for example Capsules, Components, Collaborations, Sequence Diagrams, Classes, Processors, Class Utilities, and so on.

Table 1 lists the conversion mappings which you can use to understand where the main objects from the OTD model are in the converted Rational Rose RealTime model. The list is not exhaustive, compliment by also reviewing the converted model to understand how the model is organized.

Table 1 Conversion mappings

ObjecTime Developer	**Rose RealTime**
Configurations	Each configuration becomes a component. Only the currently active language, compiler, and target will be converted. All of the component build settings are stored in properties. If your OTD model contains multiple valid languages, compilers, and targets, you will have to create separate configurations in OTD before converting the model.
Logical threads	In C++ logical threads are added to each component. In C the logical threads are assigned to the top level capsule.
Physical threads in update	Physical thread definitions are added to each component.
Overrides File	Added to the component <*language*> Target RTS property called “CompilationMakeInsert”.
SequenceOf	Becomes a class subclassed from SequenceOf. The class contains new operations that implement the SequenceOf behavior which in OTD was generated by the code generator. The **default size** becomes a parameter of the constructor.
Sequence	Becomes a class subclassed from Sequence. All fields become attributes and methods become operations.
Constants	A class utility is created for each constant. The class utility is a place holder and does not actually get generated as a class. The class utility contains an attribute of the same name which is defined as a constant.

ObjecTime Developer	Rose RealTime
External Data Classes	Each external data class is converted to a class. The **GenerateClass** property is unchecked which means that a class is not actually generated, instead the definition will only be a placeholder within the model for something externally defined. The compiler will still expect to see the definition at compile time in a header file that you include and at link time in a library or object file.
Last compiled actor	Becomes the top level capsule of each component.
Compilation environments	A component is created for each compilation environment configuration.
Debugging tools settings	Not converted - now available on processors and is independent of the build settings.
Daemons and probes	These are ignored during the conversion process, and are therefore not converted. You should make a note of them in your OTD model and recreate in your Rational Rose RealTime model once it is built and running.

Package Inheritance

Rose RealTime supports the same package inheritance as ObjecTime Developer. OTD and Rational Rose RealTime have the same basic capabilities for package inheritance and package containment, however the UI capabilities are different. The following describes how the Linear Form Import handles those.

- Logical View:

  ObjecTime Developer's package inheritance hierarchy remains the same when imported into Rose RealTime.

- Component View:

  The above also applies to Compilation Unit Packages. In addition, for each Compilation Unit Package that has at least one configuration attached to it, a component of type "C/C++ Library" will be created. The name of that component will be the name of the package concatenated to the name of the top level configuration (of type "C/C++ Executable"). This is necessary as a Compilation Unit Package could have multiple configurations attached to different top level configurations. A final step is required to create the missing dependencies between the resulting components. These are created by means of an algorithm that analyses the class usage of each components.

Adjusting Graphical Layout

Examine each diagram in your model. In a few cases you may need to adjust the size of capsule roles, states and other graphic entities. Some of the general options available for an Rational Rose RealTime model will have an impact on the graphical layout, for example font, font size, whether or not capsules display the capsule class name as well as the capsule role name, and whether or not the protocol name is shown on ports.

Use **Tools > Options > Diagram > UML options** to toggle the display of classifier names on roles, and the display of protocol names on ports.

Verifying OTD Requirements

If your OTD model contained any references to requirements that were located in a requirements update, verify that you can link to them from your Rational Rose RealTime model. If you followed the steps to output your requirements update from OTD, there should be an HTML file containing these. Any capsule, protocol or class that contained a reference to a requirement in OTD, should have a URL link to the HTML file. You can verify this either in the Rational Rose RealTime browser, or by looking at the Files tab for each such capsule, protocol and class. If you double-click on this link, you should see the corresponding requirement loaded into your default web browser.

There should be a **$REQUIREMENTS** symbol in each such link.

Temporarily Adding the Model to Source Control (optional)

For large models it may take some time before your model is actually built, run, and tested. For this reason your project leader may decide to temporarily place the model under source control until the Rose RealTime conversion is completed.

Configuring Your Source Control Tool

Read the section in the *Guide to Team Development - Rational Rose RealTime* which details step by step how to setup a supported source control system to work with Rose RealTime. Because this will be temporary, you probably only have to setup for single stream development.

You should test that your source control tool is properly configured by following the steps in the *Guide to Team Development - Rational Rose RealTime* to add a sample model to source control. This will confirm that the tools are configured properly.

Splitting a Model into Smaller Units

Before adding a model to source control, split the model into smaller controlled units. For large models, we recommend that you control units down to the class level. This will greatly improve the time required to save and generate the model.

To control the top level packages into separate units:

1. Select **Model** in the browser, right-click (this will show the context menu) and select **File>Control Child Units**. Click **Yes** for the "control child units recursively" question.
2. Save the model.

   You are now prompted for the filenames of the controlled units.
3. Select **Yes to All**.

   Note: A model or controlled unit must always be saved before it can be submitted to source control.

Next, you will submit the files to source control.

4. Add the model to source control using **Tools > Source Control > Submit all Changes**.

Once you can build and run the converted model we highly recommend that you uncontrol units and save the model in another file. You can keep the converted model in source control, but use only as a history of the conversion itself. Any model architecture changes and organization should be made while the model is in one file. This is explained more in the *Guide to Team Development - Rational Rose RealTime* and in Organizing for Team Development and Source Control

If you have any problems at this point refer to the for *Guide to Team Development - Rational Rose RealTime* troubleshooting information.

Building and Running a Model in Rational Rose RealTime

Once the model has been loaded into Rational Rose RealTime and you are familiar with the location of all the parts of your model, it is time to build your model.

Configuring Your Environment

Ensure that your compilation and target environments are setup correctly. You could build and run test models or some of the example models to verify the steps. In addition, all supporting libraries, include files, and so on that were available to your OTD model must be available and visible to the components in the Rose RealTime model. In some cases you may have to change paths, or environment variables to ensure that the components will find the supporting files.

The dependencies list for attributes in ObjecTime Developer are not converted. Dependencies must be recreated using the **Build > Add Class Dependencies...** command. This runs a script that checks the model elements for dependencies and adds them. It does not, however, find references that exist only in detailed code.

Building a Model

On account of several substantial enhancements to the language add-ins (these provide support for language support with Rose RealTime) you will have to make some changes to your model in order to build.

Note: For large models it is recommended that before building you should control units to the class level. This will greatly improve save and generation times. The *Guide to Team Development - Rational Rose RealTime* explains how to control units.

See the *Installation Guide - Rational Rose RealTime* for information on how to update your model to use with the current release of the language add-in.

For C++ models:

Follow the steps detailed in the *Installation Guide - Rational Rose RealTime* to get your converted model built and running.

For C models:

Follow the steps detailed in the *Installation Guide - Rational Rose RealTime* to get your converted model built and running.

It is highly recommended that you run the same tests that you did in OTD prior to the conversion. This will ensure that the conversion to OTD is complete and correct.

Organizing for Team Development and Source Control

When the model builds and runs in Rose RealTime and the project leader determines that the model is ready for main-stream development to begin or continue, then you should start concerning yourself with organizing the model for team development and adding the model to source control.

To complete this step of the conversion we have provided the The *Guide to Team Development - Rational Rose RealTime* which provides detailed discussion regarding organization of models and all the detailed steps required to configure and use Rational Rose RealTime with the supported source control systems.

New Projects

For newer projects, your architecture may not be fixed in stone yet. In this case you should think about how best to organize your model for team development. For example, in Rose RealTime you can split models into subsystems and have different teams work on different models.

Existing Projects

For existing projects where the architecture is stable, you may want to add the model, as is, to source control.

If the Model was Temporarily Added to Source Control

When the model builds and runs, the conversion is successful and the project leader determines that the model is ready for main-stream development then you have two options:

Changing the Granularity and Submitting Elements

If you want to keep developing the model in the current source control area and with the current model organization you can simply adjust the granularity of the controlled units and submit to source control. See the *Guide to Team Development - Rational Rose RealTime* for more details on selecting the correct granularity level.

To control the model to the finest granularity (class level), then submit to source control:

1. For each package that was initially controlled in *Temporarily Adding the Model to Source Control (optional)* on page 17, open the specification dialog and on the **Unit Information** tab and set **Control new child units**.
2. Click **OK** to accept the change.
3. In the **Model View** tab in the browser, multi-select all **Logical View** and **Component packages**.
4. From the context menu, click **File > Control Child Units**. Apply recursively.
5. Submit all new units to source control using **Tools > Source Control > Submit All Changes to Source Control**.

Uncontrol the Model and Save into One .rtmdl File

You may not want to immediately submit the temporary model to source control. For example, if you want to place the project-ready model into another source control database or directory location. At this point, you can save the result of your conversion into a single .rtmdl file.

If you perform the following steps, you will not lose what was already in source control; however, it will allow you to save the model into one .rtmdl file that you can then move or work on before preparing for mainstream development.

1. Open the converted model that was temporarily added to source control.
2. Select the model element in the browser.
3. Right-click and select **File > Uncontrol Child Units**.
4. Answer **Yes** when prompted.
5. Select **File > Same Model As...** and chose a new name for the model.

   Note: Note that you want to keep the .rtmdl file that represents the model that was placed in source control.

You will now have a second .rtmdl file which contains the converted model. If you want to access the model that you originally stored into source control, open the workspace for the original model.

Preserving Source Control History

If ClearCase is being used and an audit trail to ObjecTime Developer files is required, you may want to create hyperlinks from the new Rose RealTime files back to the old OTD files in source control. Hyperlinks should be created when the model is in a stable state where creation of the links is relevant. See your ClearCase documentation for the syntax for creating hyperlinks.

Configuring the Build Process

In OTD, you can use batch mode to automate builds. In Rose RealTime, batch mode does not exist, however similar functionality can be implemented.

Note: If you require help converting your build scripts, contact your sales representative.

The *Guide to Team Development - Rational Rose RealTime* details how to configure automated builds with Rose RealTime.

Changes to Code that Uses Default Arguments

ObjecTime Developer models which used the RTTimespec constructor with only one parameter, as in the following code:

```
timer.informIn(RTTimespec(2));
```

will result in a compile error after conversion of the model to Rational Rose RealTime. The compile error will appear something like:

```
..\rtg\Driver.cpp(67) : error C2440: 'type cast' : cannot convert from
'const int' to 'struct RTTimespec'

No constructor could take the source type, or constructor overload
resolution was ambiguous.
```

The reason is that in ObjecTime Developer, the RTTimespec constructor included default arguments, that is, RTTimespec (long=0, long=0). The default constructor values are not supported with RTTimespec in Rational Rose RealTime. Any code that made use of the default arguments needs to be changed to supply both constructor arguments. For example:

```
OTD = RTTimespec (2);
```

must be changed to:

```
RRT = RTTimespec (2, 0);
```

Modifying the Comment Block Size

Clarification: The size of the action blocks is not changed; the sized used in OTD is preserved. If the fonts used to create labels are different (as in this case), the line of text might not fix the box. You can now scale the width of the action box to the desired amount. This amount depends on the font used in the OTD model; RoseRT; and user's perception.

To scaled the width, at the end of RoseRT.ini file, add the following two lines:

```
[OTDImport]
ActionBlockScale=130
```

RoseRT will use the value specified after `ActionBlockScale` in the following way:

```
<RoseRT's width of action block> = <OTD's width of action block> *
ActionBlockScale / 100
```

The width of action blocks will be increased 1.3 times.

Continue to modify this number to obtain the desired scaling.

Appendix A: Port Message Conversions

Port Message Conversions

The following table specifies the list of port messages conversion for Rational Rose RealTime.

	Old syntax:	New syntax:
send	port.send(sig)	port.sig().send()
	port.send(sig, data)	port.sig(data).send()
	port.send(sig, data, prio)	port.sig(data).send(prio)
	port[index]->send(sig)	port.sig().sendAt(index)
	port[index]->send(sig, data)	port.sig(data).sendAt(index)
	port[index]->send(sig, data, prio)	port.sig(data).sendAt(index, prio)
invoke	port.invoke(buf, sig)	port.sig().invoke(buf)
	port.invoke(buf, sig, data)	port.sig(data).invoke(buf)
	port[index]->invoke(buf, sig)	port.sig().invokeAt(index, buf)
	port[index]->invoke(buf, sig, data)	port.sig(data).invokeAt(index, buf)
purge	port.purge()	port.purge()
	port.purge(sig)	port.sig().purge()
	port[index]->purge()	port.purgeAt(index)
	port[index]->purge(sig)	port.sig().purgeAt(index)
recall	port.recall()	port.recall()
	port.recall(0)	port.recall()
	port.recall(0, 0)	port.recall()
	port.recall(0, 1)	port.recallFront()

	port.recall(sig)	**port.sig().recall()**
	port.recall(sig, front)	**port.sig().recall(front)**
	port[index]->recall()	**port.recallAt(index)**
	port[index]->recall(0)	**port.recallAt(index)**
	port[index]->recall(0, front)	**port.recallAt(index, front)**
	port[index]->recall(sig)	**port.sig().recallAt(index)**
	port[index]->recall(sig, front)	**port.sig().recallAt(index, front)**
	Note: The script understands that the sig EmptySignal == 0)	
recallAll	**port.recallAll()**	**port.recallAll()**
	port.recallAll(0)	**port.recallAll()**
	port.recallAll(0, 0)	**port.recallAll()**
	port.recallAll(0, 1)	**port.recallAllFront()**
	port.recallAll(sig)	**port.sig().recallAll()**
	port.recallAll(sig, front)	**port.sig().recallAll(front)**
	port[index]->recallAll()	**port.recallAllAt(index)**
	port[index]->recallAll(0)	**port.recallAllAt(index)**
	port[index]->recallAll(0, front)	**port.recallAllAt(index, front)**
	port[index]->recallAll(sig)	**port.sig().recallAllAt(index)**
	port[index]->recallAll(sig, front)	**port.sig().recallAllAt(index, front)**
	Note: The script understands that the sig EmptySignal == 0	
msg operations	**msg->reply(sig)**	**rtport->sig().reply()**
	msg->reply(sig, data)	**rtport->sig(data).reply()**

	msg->sap()->getIndex()	**msg->sapIndex0()**
	msg->sap()->index()	**msg->sapIndex()**
other	**timer.informIn(...).isValid()**	**timer.informIn(...)**

Appendix B: Code Segments that are Not Converted

Code Segments

The following list specifies entities that are recognized and flagged, but can not be converted to Rational Rose RealTime:

RTDataWrapper
RTDataWrapper_char
RTDataWrapper_double
RTDataWrapper_float
RTDataWrapper_int
RTDataWrapper_long
RTDataWrapper_short (These classes are no longer needed.)
RTEndPort (This concept does not exist in Rational Rose RealTime.)
RTEndPortRef (Replaced by RTProtocol)
RTCommSAP
RTAsyncCommSAP
msg->signal
msg->getSignal
RTSignal::name
RTSignal::lookup

The pattern Using *RTDATA (problem with a Null Data Class in a Protocol) is also recognized and flagged, but can not be converted to Rational Rose RealTime.

Index

G

I

L

M

N

O

P

R

S

T