Telelogic

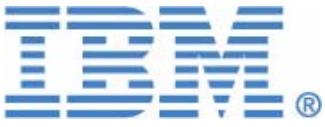
Rhapsody

Upgrade Guide

IBM

Rhapsody®

Upgrade Guide



Before using the information in this manual, be sure to read the “Notices” section of the Help or the PDF available from **Help > List of Books**.

This edition applies to Telelogic Rhapsody 7.4 and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Upgrade Considerations

This section describes behavior and functionality changes between versions of Rhapsody that you must consider when upgrading your installation.

Note

When you install a higher version of Rhapsody, you *must* use the properties that exist for that version.

Upgrading Rhapsody on Windows 98 Systems

If you are upgrading to a newer version of Rhapsody and want to keep the previous version as well, the usual method is to rename the existing installation directory and then install the new version. However, this might cause a problem as a result of a known bug in Windows 98—on the DOS level, directory names with eight characters are not renamed. For example, if you rename your default install directory from `Rhapsody` to `Rhapsody_old` and then install the newer version to `Rhapsody`, the new version will in fact be installed to the same directory where the old version resides.

Upgrading from Modeler to Developer Capability

If you have the Modeler version of Rhapsody and want to upgrade to the Developer version, send e-mail to Rhapsody Customer Support for a software license key that will allow you to open Modeler models in Developer. You can then save the model in Developer and the translation will be complete. This is a one-time process. The key will be valid only for a few days.

You must add the Modeler conversion key to your `license.dat` file, then invoke the product from a DOS window using the following command:

```
[c:\<rhapsody_dir>\]rhapsody
[-dev_ed|-modeler|-solo|-validator] -
lang=[Cpp|C|Java|Ada] [-convert]
```

This command allows Developer to open the Modeler project. To convert the Modeler project to Developer format, you must save the project. If you do not save the project, it will remain in the old format.

Upgrading Rhapsody Applications

In general, whenever you upgrade to a newer version of Rhapsody, it is recommended that you regenerate and rebuild your code. If you have your own operating system adapter, you should also rebuild your framework libraries in the new version of Rhapsody before building application code.

As a general rule, models created in any version of Rhapsody cannot be loaded into earlier versions of Rhapsody. However, a Rhapsody version can have several maintenance releases. Models can be loaded within any of the maintenance releases for a given Rhapsody version.

All precompiled samples accompanying version 2.2 and higher of Rhapsody were compiled with Microsoft Visual C++ (MSVC) 6.0. If you use MSVC 5.0, you must recompile the samples with MSVC 5.0. However, if the sample uses a `.dsp` file (as for a GUI interface), you cannot load a `.dsp` file created in MSVC 6.0 into 5.0. In this case, you must recreate the `.dsp` file in MSVC 5.0 before recompiling.

Upgrading Applications that Use Configuration Management

When upgrading a Rhapsody model that uses a configuration management tool, you must do one of the following:

Option 1

1. Outside of Rhapsody, check out all the units.
2. Open Rhapsody.
3. Save the model.
4. Check in all the units.

Option 2

1. Open the model.
2. Save the model under another name, to a writable directory.

This will avoid the possibility of attempting to write to units that are read-only, which could cause model corruption.

Required User Actions for All Releases

This subsection documents the changes you must perform for all Rhapsody upgrades.

COM API

Applicability: Rhapsody-compiled COM clients

Due to changes in the `rhapsody.tlb`, you should recompile every compiled COM client (as opposed to clients that use an interpreter or a VM environment such as Visual Basic) that takes advantage of the TLB information (not using the `IDispatch` interface).

Documentation

The Rhapsody product comes with user documentation. To access its help file, with Rhapsody open, choose **Help > Help Topics**. In addition, to access the List of Books to access links to the Rhapsody manuals, which are available as PDF files for easy printing, choose **Help > List Of Books**.

Upgrading to Version 7.4.0.1

Changes in Version 7.4.0.1

The changes in version 7.4.0.1 of Rhapsody are listed below.

Code Generation

C, C++ Previously, if you did not specify a value for the property ObjectsDirectory, object files would be generated in the same directory as the c/cpp files. Now, if you do not specify a value for ObjectsDirectory, the object files will be generated in the component directory.

C++ Previously, if a user manually specified an order for attributes, the generated code respected this order but only within visibility groups. Beginning in 7.4.0.1, the order of attributes in the generated code is as specified by the user, regardless of visibility.

MicroC Profile

- You will see differences in the code generated in the doExecute function for the default “active”.
- Changes have been made to the parameters used for the generated function `RiCTimedAction_init`.
- You will see minor differences in the code generated when using the segmented memory feature.

The following code generation changes were made to increase MISRA-compliance:

- The memory allocator’s macros were inlined into the generated code. Now, instead of using the macros, such as DYNAMICALLY_ALLOCATED, the code generator generates their definition inline.
- The flowport macro DIRECT_FLOW_DATA_SEND now receives a different set of parameters.

Frameworks

OXF - C++

- In file *linuxos.cpp*, minimum stack size was increased.

OXF - C

- Changes made to file *RiCEvent.h* in order to resolve mismatch between declaration and definition of `RiCTimeout_RiCSetMemoryAllocator` in *RiCEvent.h* and *RiCEvent.c*.

AutomotiveC

In version 7.4.0.1 of Rhapsody, many of the features of the AutomotiveC profile were moved to a new profile called MicroC, which is loaded by the AutomotiveC profile. Beginning with this version of Rhapsody, the AutomotiveC profile contains only features that are designed exclusively for the automotive industry.

As a result of this change, you must carry out a number of steps when you open, for the first time, projects that use the AutomotiveC profile.

The steps for upgrading such models are as follows:

1. Using File > Add to Model, add the MicroC profile [installation directory]*\Share\Profiles\MicroC\MicroC.sbs*.
2. If your project does not use the OSEK features from the AutomotiveC profile, you can use the *Change to* option in the browser's context menu to change the type of the project to MicroC and then delete the AutomotiveC profile from the project.
3. Save the model.

Upgrading to Version 7.4

Changes in Version 7.4

The changes in version 7.4 of Rhapsody are listed below.

Frameworks

OXF - C++

- In the class `OMThreadManager`, the forward declaration `class IOxfActive` has been moved to the header file.
- Version 7.4 of Rhapsody includes an option for using a C++ framework that does not use templates.
  To use this option, define `OM_NO_TEMPLATES_USAGE` in the file *Share\LangCpp\osconfig*<Adapter>*\omosconfig.h*, and rebuild the framework.
  To make this option possible, a number of classes have been modified, and a number of new classes have been added.

  Note: When using the template-less framework, the framework memory manager cannot be used. To avoid compilation errors, you should define the following macro:

```
#ifdef OM_NO_TEMPLATES_USAGE
#define OM_NO_FRAMEWORK_MEMORY_MANAGER
#endif // OM_NO_TEMPLATES_USAGE
```

Code Generation

C

Before version 7.4, if you created a global object with multiplicity greater than 1, Rhapsody would generate an *extern* forward declaration in the package header file, for example, for object a of type A, the following code would be generated: `extern struct A a[3];`
Because newer compilers do not accept this code, this forward declaration is no longer generated. If you want to retain the previous code generation behavior you can add the boolean property `CG::Package::GenerateExternDeclarationForObjectArray` and set the value of the property to True.
If you encounter compilation errors after roundtripping changes made to the code generated by Rhapsody, it is recommended that you restore the previous code generation behavior by setting `GenerateExternDeclarationForObjectArray` to True.

AutomotiveC Profile

C In the previous version of Rhapsody, mutators were erroneously generated for framework attributes, resulting in compilation problems. This problem has been fixed in version 7.4, so you may find differences in the generated code, compared to code generated with older versions of Rhapsody.

Upgrading to Version 7.3 MR-1

Changes in Version 7.3 MR-1

The changes in version 7.3 MR-1 of Rhapsody are listed below.

Frameworks

ExtendedC_OXF

- To allow timeout labels in sequence diagrams to display the state, an RhpString state parameter was added to the function `RiCTimeout_init`, when code is instrumented.

  The parameter was also added to the `RiCTimeout_create` function, which calls `RiCTimeout_init`.
- In the file *RiCTimer.c*, changes have been made to the function `goNext` so that when using instrumentation, when entering Idle state, the AOM will advance the time to the next pending Timeout or TimedAction, the earlier between the two.

Code Generation

C, C++ In the previous version of Rhapsody, forward declarations were generated even when the value of the property `CG::Configuration::GenerateForwardDeclarations` was set to False. This problem has been corrected in the current release, so you may find changes in your generated code relative to code generated with the previous version.

C, C++ Changes have been made to the property `MakeFileContent` for the various Microsoft environments in order to allow the Rhapsody framework to continue to catch asynchronous exceptions when newer versions of Visual Studio (2005 or later) are used to build the framework. As a result, you may find differences in the generated makefiles, relative to the previous version of Rhapsody.

C, C++ Changes have been made in the way annotations are used in code generated from flowcharts. As a result, you may find differences in your generated code relative to code generated with the previous version of Rhapsody.

AutomotiveC Profile

- C For periodic Actives, there is now a call to `RiCTaskEM_beginMyTask()` at the beginning of the operation `doExecute`.
- C In instrumented code for models with statecharts, there is now an additional entry in the statechart's virtual table for the relevant SerializeStates function.

Upgrading to Version 7.3

Changes in Version 7.3

The changes in version 7.3 of Rhapsody are listed below.

Code Generation

C — The default value of the property `C_CG::Configuration::DescriptionEndLine` was changed. As as result, an additional space now appears before the closing */ in comments generated for descriptions of model elements.

C — When using the AutomotiveC profile, if no Active has been defined, Rhapsody now provides a default Active. To enable this, the following changes have been made in terms of code generation:

- package file now contains:
  - attribute called `ric_timedAction`
  - operation called `Default_doExecute`
  - call to `RiCTimedAction_init` function
- file *ExtendedCOxf_cfg.h* contains additional `#defines`

C — In the MISRA98 profile, the value of the property `C_CG::Class::IsInOperation` has been changed to Inline. As a result, you will find that certain functions are now generated as macros.

C — Due to a bug, properties that affect C code generation for operations and attributes were not affecting code generation when the value was changed at the class level. Now that this bug has been corrected, you may find differences in your generated code due to the correct application of these properties to class members.

C++ — Comments generated for state titles now use the `//` notation rather than `/* */`.

C++ — A bug was fixed in the generation of code for classes with multiple inheritance. As a result, you may see differences in the order of the base classes, relative to the code generated with the previous version of Rhapsody.

C, C++ A number of code generation properties that apply to operations no longer affect autogenerated operations, for example, `C/CPP_CG::Operation::Inline`. As a result, you may find changes in the code generated for autogenerated operations.

C, C++ A number of bugs were corrected with regard to the enclosing of instrumentation code within the appropriate `#ifdef` directives. As a result, you may find that the instrumentation code generated with 7.3 includes `#ifdef` directives that were not generated previously.

C, C++ Autogenerated comments for groups of code elements, such as "framework operations", were previously not handled correctly during roundtripping. Roundtripping now handles these correctly, but if you regenerate code with 7.3 for a model that was previously roundtripped with an older version of Rhapsody, you may find that these comments are missing from the generated code.

C, C++ Due to the correction of a bug regarding the location of `#define` directives required for animation, you may find that these directives now appear in a different location in the generated code, relative to code generated with previous versions of Rhapsody.

C, C++ Due to the correction of a bug regarding the relative location of `#define` directives and forward declarations in generated code, you may find that the order of these items is different in the code generated with 7.3, relative to code generated with previous versions of Rhapsody.

C, C++ The `ignore` annotations generated by Rhapsody no longer contain a blank line.

C, C++ Beginning with version 7.3, Rhapsody allows you to specify a public global variable as static in the Features dialog in both RiC and RiC++, and generates the static variable declaration in the header file and not in the implementation file.
This change applies only to the *Advanced* code generation mode so it does not affect older Rhapsody in C++ models, which by default use the *Classic* code generation mode.

C, C++, Java A number of bugs were corrected with regard to the properties `CG::Class::CreateImplicitDependencies` and `CG::Type::GenerateDeclarationDependency`. As a result, you may find that certain `#includes` that were generated in your code previously no longer appear in the generated code.

C, C++, Java Prior to version 7.3 of Rhapsody, the transition-handling code generated by Rhapsody used a switch statement to represent the possible states. Now, this code uses an if/else structure. To allow older models to use the previous code generation behavior, a property called [lang]_`CG::Statechart::StatechartImplementation` was added to the *Pre73* backward compatibility profiles.
The default value of this property is `SwitchOnly`. It can also take the value `Default` which will result in use of an if/else statement.

Reverse Engineering

C, C++ Beginning with version 7.3, when Rhapsody's reverse engineering encounters static public global variable declarations in a header file, the declaration is imported into Rhapsody such that it will later be regenerated in the header file and not in the implementation file as was done previously.
This change applies only to the *Advanced* code generation mode so it does not affect older Rhapsody in C++ models, which by default use the *Classic* code generation mode.

C, C++ Beginning with version 7.3, when reverse engineering files, if a file references a header file but the path in the include directive is not clear enough for Rhapsody to find the file, Rhapsody will search the list of files to be reverse engineered to see if the list contains a header file with that name. If there is such a file, Rhapsody will use the full path that was provided for that header file, assuming that this is the header file that was being referenced in the original file. This behavior is controlled with the property `C/CPP_ReverseEngineering::ImplementationTrait::AutomaticIncludePath`.
In the *Pre73* backward compatibility profiles, the value of this property is set to False. If you would like to enable this feature for older models, you will have to change the value of this property to True.

Roundtripping

C, C++ Starting with version 7.3, Rhapsody's roundtripping feature can handle changes involving preprocessor directives such as `ifdef`. A new property called `RoundtripPreprocessorDirectives` (under `C_Roundtrip::General` and `CPP_Roundtrip::General`) has been added to allow this ability to be turned off/on. The default value of this property is True. In the *Pre73* backward compatibility profiles, the value of this property is set to False in order to provide the older roundtripping behavior.

Tracing

- The message *Popped method from empty callStack* no longer appears in the trace output.

Framework

C

OXF Library

- Since C++ libraries are required for use of tracing with C, the appropriate compilation commands for these libraries have been added to various make files.
- In a number of files, parentheses have been added where macro arguments are used in order to meet MISRA requirements.

- In a number of files, curly brackets have been added for all `if` structures in order to meet MISRA requirements.
- In the files *RiCCollection.h* and *RiCCollection.c*, the attribute `pos` has been renamed `position`.
- In the file *RiCOSVxWorks.c*, data struct initialization has been added to the function `RiCOSMessageQueue_isFull`.
- In the files *RiCOxf.h*, *RiCOxf.c*, and *RiCTask.c*, the function `RiCOXF_setTheDefaultActiveObject` has been renamed `RiCOXF_setDefaultActiveObject` in order to meet MISRA requirements.

AOM Library

- In the file *osconfig/MultiWin32/ricosconfig.h*, the flag `ROM_MISRA_COMPLIANT_ADAPTER` has been removed.
- In the files *AdaInterface.h* and *AdaInterface.c*, a number of new API functions have been added to support new features in RiA, such as timeouts and active classes.
- In the files *aomcalls.h* and *aomcalls.c*, a new function called `ARCCS_shouldNotifyOpReturn` has been added. This function is used in *aommacro.h*.

ExtendedC_OXF

- The framework now has a *Default Active* that will be used for elements that do not have a specific "active" defined to handle their execution.
- "Active" objects that use Asynchronous activation mode can now be used with the Mainloop Adaptor. *TimedAction* is used to schedule the execution of such objects.
- The framework now supports Animation and Trace, subject to the following limitations:
  - On host (PC) only
  - Only when using Mainloop Adaptor

C++

AOM Library

- In the files *aomcalls.h* and *aomcalls.cpp*, a new function called `shouldNotifyOpReturn` has been added. This function is used in *aommacro.h*.

TOM Library

- Several virtual destructors have been added to avoid GNU compiler warnings.

Other Changes

- Changes have been made to the way that Rhapsody tries to locate the rhapsody.ini file.

  Prior to 7.3, the order of locations checked was as follows:

  - the $USERPROFILE/Application Data/Rhapsody directory (Windows only)
  - the directory that contains rhapsody.exe
  - the current directory at the time Rhapsody was launched
  - the Windows directory

  Now, the process used by Rhapsody is:

  - look for a file named rhapsody.<major version>.ini (for example, rhapsody.7.3.ini) in the $USERPROFILE/Application Data/Rhapsody folder (Windows only).
  - look for rhapsody.ini in the $USERPROFILE/Application Data/Rhapsody directory (Windows only)
  - look for rhapsody.ini in the directory that contains rhapsody.exe
  - look for rhapsody.ini in the current directory at the time Rhapsody was launched
  - look for rhapsody.ini in the Windows directory

  This change allows users without administrator privileges to have a separate .ini file for each version of Rhapsody installed.

  Note: $USERPROFILE refers to the %USERPROFILE% environment variable on Windows systems, for example, ..*Documents and Settings\billsmith*.

- Beginning with 7.3, Rhapsody includes precompiled framework libraries only for the host environment. If you are using other environments, you will have to build the framework libraries. This can be done from within Rhapsody by selecting the *Code > Build Framework* option from the main menu.
- The Borland environment is no longer supported.

Upgrading to Version 7.2 MR-1

Changes in Version 7.2 MR-1

The changes in version 7.2 MR-1 of Rhapsody are listed below.

Code Generation

C — When using the customizable code generation feature, the annotations added by Rhapsody for state names now include a blank space before the */ that closes the comment.

Upgrading to Version 7.2

Changes in Version 7.2

The changes in version 7.2 of Rhapsody are listed below.

A number of items refer to code respect. In Rhapsody, code respect means that the order of elements in the original code is preserved during code generation. This means that you can freely change the order of class members and globals and Rhapsody “respects” those changes. For more information about code respect, see the *Rhapsody User Guide*.

Code Generation

C	Rhapsody now uses an improved code generator. When you open an existing RiC model and regenerate code, the new generator will be used. As a result, you may notice changes in the generated code.
C	The `me` pointer has been added as an additional argument for `OM_INSTRUMENT_EVENT`. The only thing you will notice because of this is diff-s.
C	There are new macros in Rhapsody for C in order to support animation of the return value in C. You are now able to use `OM_RETURN` in the same way as it is used in Rhapsody C++. The animation return value for a triggered operation is also supported. However, note that unlike Rhapsody in C++, using `OM_RETURN` will not update output arguments.
C	Code generated for functions with no parameters now appears as `functionName(void)`, not `functionName()`. To achieve this, the default value of the `EmptyArgumentListName` property has been changed to `void`.
C (IDF)	The object `theMainTask` now initializes and cleans up its attribute `itsRiCTask`.
C, C++	If the value of the property [lang]_`Roundtrip::General::RoundtripScheme` is set to `Respect`, then the new Rhapsody code generator will be used even if the value of the property [lang]_`CG::Configuration::CodeGeneratorTool` is set to `Classic` or `Customizable`. If you want to use the old code generator or use customized code generation, the value of `RoundtripScheme` must be changed to `Advanced` or `Basic`.

C, C++	There are changes in the code generated for flow ports due to the introduction of animation support for flow ports.
C, C++	The code generated for flow ports now includes different Rhapsody annotations.
C, C++	The behavior of the `DefaultSpecificationDirectory` and `DefaultImplementationDirectory` properties has been changed. Now, the names specified with these properties are added at the end of the code generation path (before h/hpp, c/cpp) rather than at the beginning.
C, C++	Now, if a model contains a component file, then the file will be generated even if only external elements are mapped to the file. In cases where you do not want the file to be generated at all, set the value of the property `Generate` to `False`.
C++	Rhapsody now uses an improved code generator. You may therefore notice changes in the generated code, compared to code generated with previous versions of Rhapsody.
C++	The description of the implementation dependency now prints only in the specification file. Previously, it printed in the implementation file as well.
C++	When working in Respect mode, in the code generated following reverse engineering, the keyword `inline` will be used only for functions that were explicitly declared inline in the original code.
C++	When working in Respect mode, there were cases where comments appeared erroneously for the implementation of model elements rather than the specification. These issues have now been corrected, so you may find instances where comments previously appeared next to element implementations but now appear next to the element specifications.
C++	When working in Respect mode, the generated code now has improved grouping of class members based on their visibility. This improvement may result in minor code differences when comparing code to code generated with the previous version of Rhapsody.
C++	If you open in Rhapsody 7.2 an existing C++ model where the `CPP_Roundtrip::General::RoundtripScheme` property was set to `Basic` at the project level, the value of the property will automatically be set to `Advanced`. If you would like to restore the previous setting, change the value of this property in the referenced copy of the backward compatibility profile in your model (`CGCompatibilityPre72Cpp`).

C, C++, Java

In previous versions of Rhapsody, if you specified a guard and/or action for a junction connector, there were cases where these would not be included in the code generated for the statechart. This problem has been corrected. As a result, you may see differences in the code generated for statecharts with junction connectors.

Java — The implementation of `cleanUpRelations()` has been changed for the following containers: BoundedOrdered, UnboundedOrdered, BoundedUnordered, UnboundedUnordered, and Fixed.

The change in the implementation is in the line: `iter.next();`

which was replaced with the line: `iter = $Relation.listIterator(0);`

`$Relation` would be name of the currently cleaned-up relation (for example, `itsClass_2`).

Java — The default values for the `Java(1.5)Containers::Qualified::Remove` and `Java(1.2)Containers::Qualified::Remove` properties have been changed to:

```
"$IterCreate;
while(iter.hasNext()) {
    Object key = iter.next();
    if ($cname.get(key).equals($item)) {
        iter.remove();
        break;
    }
}"
```

Previously the `iter.remove();` line was `$cname.remove(key);`

Java — With the introduction of the ability to send arguments to a Java application, the code that is now generated for the `main` method will include a parameter `args` in all calls to the framework's `Init` method.

All — The `OM_RETURN(triggerEvent.om_reply);` triggered operation now has a semicolon (`;`) at the end of it.

All — Now, if the `<configuration>.cg_info` file is missing, Rhapsody displays a dialog box that informs the user that Rhapsody does not know whether files have been manually modified since the last code generation and recommends that the user perform a Force Roundtrip operation. (Previously, Rhapsody ignored the possibility that files may have been manually modified since the last time code was generated.)

All — Forward declarations of packages, events, and classes were removed from generated code because they were redundant:

- class `OMCloseHandleEvent;`
- class `OMEndThreadEvent;`
- class `OMNullEvent;`
- class `OMReactiveTerminationEvent;`
- class `OMStartBehaviorEvent;`

Animation

- For C: For user-defined types that are based on primitive types, the serialization/unserialization functions for animation will now be the basic types which these types are defined on. Previously, this was treated as an unknown type.
- For C++: `//## ignore` has been replaced with `//## auto_generated` for port deletes.
- The position of the `DECLARE_OPERATION_CLASS` macro has changed. The change in position was made in order to support call invocation for operations that use types declared inside the class. Now the `DECLARE_OPERATION_CLASS` macro will appear after the declaration of the class, and before the declaration of the animated class.

Check Model

- For Web instrumentation, the `GetConnectedRuntimeLibraries` property (for example, `[lang]_CG::Microsoft::GetConnectedRuntimeLibraries`) specifies the list of libraries that need to be linked with Web-enabled projects. Now there is a check that tests for the existence of this property and its content to make sure it is not empty. The tested property is searched under the current environment metaclass that relates to the active configuration. If the property cannot be found or its value is empty, the following new check message displays. Note that this is only a warning; code generation will not fail because of it.

```
Missing runtime libraries required for Webify Toolkit.
Check the value of GetConnectedRuntimeLibraries property for your
current environment.
```

Note: This new check message replaced this old check message:

```
Missing runtime libraries required for Webify Toolkit!
Check the value of CG::Environment::
GetConnectedRuntimeLibraries!
```

- The `Dependency on unresolved element` check now only checks `<<Usage>>` and `<<Friend>>` stereotypes (because they are the only relevant stereotypes for code generation).
- The `Composite class without a statechart Composite` check has been removed. When originally introduced to Rhapsody, composite classes had to be derived from

reactive classes. Therefore, by definition, they had statecharts and this was enforced by a check. Composite classes no longer have this restriction, so this check has been removed.

Code Generation - Makefile

C — To enable support for Visual Studio 2005, changes have been made to the generated makefiles for MS environments (by changing the value of the property `MakeFileContent`).

C++ — For the QNXNeutrinoGCC environment, the generated makefile no longer contains the superfluous `libm.so.1` in the path for `LINK_FLAGS`.

All — New items were added to the Clean section of the makefile to remove CORBA derived sources.

Reverse Engineering

C — Rhapsody now, by default, uses the Ordering mode of reverse engineering. If you open an existing RiC model and reverse engineer source files, Rhapsody will also use the Ordering mode of reverse engineering. If you would like to temporarily use Rhapsody's previous reverse engineering behavior, you can set the value of the property `C_ReverseEngineering::ImplementationTrait::RespectCodeLayout` to `Mapping`.

C++ — During reverse engineering, Rhapsody will now create template instantiation classes where relevant. This behavior is controlled by the `CPP_ReverseEngineering::Promotions::EnableTypeToTemplateInstantiation` property, whose default value is set to `Checked` (meaning true).

Roundtripping

C — Rhapsody now uses the Respect mode of roundtripping by default. If you open an existing RiC model and roundtrip changes to code, Rhapsody will also use the Respect mode of roundtripping. If you would like to temporarily use Rhapsody's previous roundtripping behavior, you can set the value of the property `C_Roundtrip::General::RoundtripScheme` to `Advanced`.

C, C++ — Now, when working in Respect mode, the default value of the property that controls the roundtripping of deleted items, [lang]`_Roundtrip::Update::AcceptChanges`, is `Default`. This means that roundtripping will allow the deletion of all elements except classes, provided no parsing errors are encountered as a result.

C++ — In Respect mode, the Output window shows add/remove of files or files' fragments only when `[lang]_Roundtrip::General::ReportChanges` is set to `All` (default is `AddRemove`).

Java — The `[lang]_Roundtrip::Update::AcceptChanges` property now has a `Default` value, which is also set as the default. It takes affect when the `RoundtripScheme` property is set to `Advanced`, with the behavior the same as in C, C++, and Java.

C, C++, Java — It is now possible to roundtrip deletion of elements from the code. However, this is disabled for an element that has a prolog and/or an epilog.

C, C++, Java — In code generated for statecharts, there were cases where Rhapsody annotations would no longer appear when code was regenerated after roundtripping. This problem has been corrected.

All — The annotation for an event constructor has changed from `statechart_method` to `auto_generated`.

All — For samples with ports, port annotations like `//## classInstance x` was changed to `//## ignore`. Note that roundtrip will not work on ports.

All — Now, roundtripping is triggered only by the relevant menu item or by switching the focus from the file editor to the browser of other Rhapsody component. It is no longer triggered when switching from one file to another in the editor.

Modeling

- Blocks are no longer available or used in Rhapsody. When you open a model that was created in an older version of Rhapsody and it used blocks, Rhapsody 7.2 converts the blocks to be objects.
- Profiles for backward compatibility now appear in the **Settings** category of the Rhapsody browser.

GUI

- When you create a new project, the **Type** drop-down list in the **New Project** dialog now displays only profiles that represent specific domains such as SysML or DoDAF. The other profiles included with Rhapsody can be added to your project using the **File** > **Add to Model...** option. The technical criterion used by Rhapsody for filtering the project type list is the existence of a [profile name].*txt* file (which contains a description of the profile). Only those profiles with such a text file are included in the list.

Rhapsody API

- `IRPBlock` was removed.
- If you have to access an event in a script, the syntax to use now for constructing the full path is `PackageName::EventName` (double colon), rather than `PackageName.EventName` (single period). The new syntax is the same one used for accessing classes in a package. To access an operation within an event, you use `PackageName::EventName.OperationName` (single period), just as you do when accessing an operation of a class.
- `IRPInterfaceItem` is now derived from `IRPClassifier` rather than `IRPModelElement`. You should therefore check if this change will affect the behavior of your scripts. In addition, it is recommended that COM Rhapsody API clients be recompiled.
- The metaclass for reference activities has been changed from `State` to `ReferenceActivity`.
- In the callback API, additions have been made to the interface `IRPApplicationListener`. If you have implemented this interface in your code, make sure to update your code so that it fully implements the interface.

Java API

The Java API has been upgraded to Java 5.0. Any client applications also need to be upgraded.

SysML Profile

- `FlowAttribute` has been deprecated. Use `FlowProperty` instead.
- `ValueBinding` has been deprecated. Use `BindingConnector` instead.
- System Blocks are now referred to as Blocks.
- Links are now referred to as Connectors.
- When typing `FlowPorts`, use a `FlowSpecification` rather than an `Interface`.

Support for 64-bit Targets

If you want to build applications for 64-bit targets, you must first rebuild the Rhapsody framework libraries. If you are running Rhapsody on a 64-bit system, then if you rebuild the libraries using the menu option **Code > Build Framework**, the Rhapsody libraries will be rebuilt such that you will be able to build applications for 64-bit targets. However, if you are running Rhapsody on a 32-bit system, you will have to rebuild the Rhapsody framework libraries manually.

C++ Several files in the Rhapsody framework were changed to support 64-bit architecture.

Framework

C

- A new argument called `OM_INSTRUMENT_EVENT_NO_UNSERIALIZE` was added to RiC oxf macros `OM_INSTRUMENT_EVENT`. These macros are being used in animation mode to instrument events. This change affects the generated C file of packages that contains events. Old user code that contains this macro needs to be regenerated in order to run it.
- `RiCBoolean` has been changed from `int` to `unsigned char`.
- `RiCTRUE` and `RiCFALSE` have been defined as `((RiCBoolean)0/1)`.
- All assignments to `RiCBoolean` are with `RiCTRUE`, `RiCFALSE`, like `RiCBoolean a = RiCTRUE;`
- Previously, `RiCReactive_takeTrigger` returned a value that was not used. This violated MISRA rules. This problem has been corrected.
- The `RiCOSVxWorks.c` file has the following changes:
  - The `ifdef clause` for `#include <errno.h>` was removed.
  - For `RiCOSEventFlag_reset` function, if the call to `semTake` fails, it tests if `errno` is not `S_objLib_OBJ_UNAVAILABLE`, meaning that the queue is empty. In that case, there is no need to report an error, since the semaphore is locked by the current running thread.
- In order to provide support for MS Visual Studio 2005, changes were made to the files `msoxf.mak` and `Msdox.mak`.
- As part of the changes to code generated for flow ports, changes were made to the file `RiCPortMacros.h`.
- In the `RiCDefaultReactivePort.h` file, the include to `RiCReactive` has been replaced with an include to `IRiCReactive`.

- In the files `RiCOSNT.c` and `RiCOSVxWorks.c`, calls to the macro `OM_NOTIFY_ERROR` were added. This macro is called when framework functions encounter operating system-level errors. By default, the macro is empty, but you can provide content to achieve the error-handling behavior that you require. See also **Error Handling on VxWorks and Microsoft**.
- A method called `RequestTimeNotification` was added to the files `RiCTimer.c` and `RiCTimer.h`.

C++

- The `OUT_PORT` and `OPORT` macros have been changed to verify that the ports have been initialized. As a result, sending messaging through a port of another object via a direct link (an instance of a direct association) will lead to compilation errors. To resolve this, either call an operation on the associated class that will relay the message via the port, or use a link via ports instead of a direct link.

  Note that the `OPORT` macro is equivalent to the `OUT_PORT` macro.
- If a user creates a new adapter or upgrades his own existing adapter to Rhapsody 7.2, he must add the following lines to the `omosconfig.h` file for his adapter:

  ```
  typedef void * gen_ptr;
  typedef void * OMOSHandle;

  #define OM_NOTIFY_ERROR(call,func)
  ```

  For more information about `OM_NOTIFY_ERROR`, see **Error Handling on VxWorks and Microsoft**.
- `AnimMessageTranslator.cpp` has been removed from framework compilations.
- There are framework modifications to provide better support for ATG. These changes were done in the C++ oxf model: `OMEvent` and `OMHandleCloser` classes. Changes were made to the following files: `OMEvent.cpp` and OMHandleCloser.cpp.
- In QNX Neutrino 6.3.2A, animation was hanging in mutex because of its wrong (missing) initialization. This problem has been corrected. Changes were made to the qnxos.cpp file.
- The `vxos.cpp` file has the following changes:
  - It now has this include line: `#include <errno.h>`
  - For the `VxOSMessageQueue::get` function, if the call to `msgQReceive` fails, it tests if `errno` is not `S_objLib_OBJ_UNAVAILABLE`, meaning that the queue is empty. In that case, there is no need to call `OM_NOTIFY_ERROR`. For more information about `OM_NOTIFY_ERROR`, see **Error Handling on VxWorks and Microsoft**.
  - For the `VxOSEventFlag::reset()` function, if the call to `semTake` fails, it tests if `errno` is not `S_objLib_OBJ_UNAVAILABLE`, meaning that the queue is empty. In that case, there is no need to report an error, since the semaphore is locked by the current running thread.

- In the `IOxfEventSender.h` file, a virtual destructor was added for `IOxfEventSender`.
- Changes were made to the following files in order to provide better support for ATG: `omevent.cpp`, `omevent.h`, `omhandlecloser.cpp`.
- In the `linuxos.cpp` file, the following changes were made:
  - `DefaultStackSize` set to `PTHREAD_STACK_MIN` instead of `0`
  - Compilation warning fixed in the function `NotifySyscallFault`
- In order to correct problems due to timer heap overflows, changes were made to the following files: `CoreImplementation.sbs`, `TimeManagement.sbs`, `omreactive.cpp`, `omreactive.h`, `omtimermanager.cpp`, and `omtimermanager.h`.
- In order to provide support for MS Visual Studio 2005, changes were made to the file `msoxf.mak`.
- In order to provide support for VxWorks 6.5, changes were made to the file `vxoxf.mak`.
- In the `ntos.h` file, Win Mutexes were replaced with CriticalSection in `NTMutex` class implementation
- In the `qnxos.cpp` file, unused variable was deleted to prevent compilation warning.
- As part of the changes to code generated for flow ports, changes were made to the file `oxfportmacros.h`.
- In order to make the timer implementation for the INTEGRITY adapter more efficient, the function `Sleep` is now used instead of `usleep` in `IntegrityOSTimer` and `IntegrityOSFactory` (in the file `intos.cpp`).
- In the `ntos.cpp` and `vxos.cpp` files, calls to the macro `OM_NOTIFY_ERROR` were added. This macro is called when framework functions encounter operating system-level errors. By default, the macro is empty, but you can provide content to achieve the error-handling behavior that you require. See also **Error Handling on VxWorks and Microsoft**.

Error Handling on VxWorks and Microsoft

There were cases where VxWorks and Microsoft would return an error but the relevant Rhapsody framework function would return void, resulting in the loss of this error information.

Now, when these framework functions encounter such an error, they call a macro called `OM_NOTIFY_ERROR`. By default, this macro is empty, but you can provide content to achieve the error-handling behavior that you require.

Java

- The Java framework domain has changed from `com.ilogix` to `com.telelogic`. This change affects the framework packages name, the animated jar files, and the user-generated code.
- The path for various Java framework components has been change from `ilogix` to `telelogic` (for example, `<Rhapsody installation path>\Share\LangJava\src\com\telelogic\rhapsody`).
- Because there is no way to stop the `RiJTimer` thread, a new operation has been added called `RiJTimeoutManager::stopTimer`. It can be called as follows:

```
RiJTimeoutManager.instance().stopTimer()
```

ReporterPLUS

As mentioned in **Modeling**, blocks are no longer used and blocks in models created in earlier versions will be available as objects in Rhapsody 7.2. Therefore, you must update all existing ReporterPLUS templates that refer to blocks to refer to objects instead.

Upgrading to Version 7.1.1 MR-3

Changes in Version 7.1.1 MR-3

The changes in version 7.1.1 MR-3 of Rhapsody are listed below.

Code Generation

C, C++ **Inclusion of libraries in LIBS section of makefile**—For building executables, Rhapsody now includes libraries in the *LIBS* section of a makefile only if the relevant usage dependency is of type *specification* or *implementation*. If the usage type specified is *existence*, the library will not be included. As a result of this change, you may find changes in your makefiles, in terms of which libraries are listed in the *LIBS* section, after generating code with the new version of Rhapsody.

C, C++ **Link order of library dependencies in makefile**—In order to achieve maximum compiler compatibility, the link order of library dependencies in the makefile is now in accordance with the rule that a library that uses another library should appear before the library that it is dependent upon. This information is derived from the component dependencies defined in the Rhapsody model. As a result of this change, you may find changes in your makefiles, in terms of the order of libraries in the *LIBS* section, after generating code with the new version of Rhapsody.

Upgrading to Version 7.1.1 MR-2

Changes in Version 7.1.1 MR-2

The changes in version 7.1.1 MR-2 of Rhapsody are listed below.

Code Generation

C — When using customized code generation, the value of the property `CG::Operation::Generate` now affects code generation as it should. Specifically, when the value of the property is set to *Specification*, the implementation of the operation is no longer generated in the .c file.

Upgrading to Version 7.1.1 MR-1

Changes in Version 7.1.1 MR-1

The changes in version 7.1.1 MR-1 of Rhapsody are listed below.

Code Generation

C — Previously, when using customized code generation, the property `C_CG::Attribute::Inline` did not affect code generation, i.e., when set to a value of *in_header*, the expected #define was not generated. This problem has been corrected, and now the property affects generated code as it does in standard code generation.

C, C++ — The code for connecting ports now uses the accessor methods named using the format *get_[portName]* (for example, `get_myPort()`) instead of the accessors with names that follow the value of the property `CG:Relation:Get`. This is done to avoid possible compilation errors. The code implementing the port itself remains unchanged as both types of accessors were also available previously.

Other Changes

Changes to the SysML Profile

Note

The changes described in this subsection are not yet supported in Gateway and ReporterPLUS.

Note

The changes made to the SysML profile in this release include the addition of a number of "new terms." Keep in mind that if you use these new terms in a model and then open the model in version 7.1.1 of Rhapsody, these terms will appear as Unresolved. To avoid such problems, you can replace the 7.1.1 version of the SysML profile with the new profile by copying the directory ..*Share\Profiles\SysML*.

The following changes were made to the SysML profile:

Support for Value Types

- The following new terms were added to the profile:
  - ValueType
  - Unit
  - Dimension
- StandardValueTypes package added for Complex and Real
- BaseSIUnits and DerivedSIUnits added, as specified in Annex C of the SysML Specifications.
- Header file SIDefinition.h provided for the implementation of the standard SysML ValueTypes. The header file was added to the property `CPP_CG:Framework:HeaderFile` in the SysML stereotype (applied to SysML projects) so that the file would be included automatically.
- The following changes were made to Rhapsody:
  - Tags can by typed by terms. This was done to support the unit and dimension tag values of ValueType (typed by Unit and Dimension) as well as the dimension tag of Unit.
  - Changes were made to the GUI controls to set the type of tags typed by Terms so that the term instance can be selected via a mini-browser.

Note

The tag's value is still a string; renaming the term instance has no effect.

Flow Specification and Flow Properties Support

- The following new terms were added to the profile:
  - FlowSpecification
  - FlowProperty
- FlowSpecification tool was added for Block Definition Diagrams (including new icon)
- The following changes were made to Rhapsody:
  - Flow Port dialog changed to use Flow Properties instead of Flow Attributes
  - Flow Properties are created by default via the dialog UI, instead of Flow Attributes.
  - When setting Flow Port types, selecting <New> creates a Flow Specification, rather than an Interface.

Object Flow and Control Flow Support

- The following new terms were added to the SysML profile:
  - ObjectFlow
  - ControlFlow
- The following limitations apply to object flow and control flow support:
  - The browser still lists control flows and object flows as activity flows
  - There are no tools in activity diagrams for drawing control and object flows; you have to draw an activity flow and set the stereotype to control flow or object flow.
  - Target and Source are not yet enforced.

Link Renamed to Connector

- The following new terms were added to the SysML profile:
  - Connector
- Now, Block Definition diagrams and internal block diagrams have a connector tool instead of a link tool.

Upgrading to Version 7.1.1

Changes in Version 7.1.1

The changes in version 7.1.1 of Rhapsody are listed below.

Code Generation

C — For code generated with customized code generation:

- Spacing is now added before `#endif`.

  The changes responsible for this change in behavior can be found in the script *file_guard_end* in the file *src/Files_Generation/Rhapsody_File.tgs*.
- Descriptions provided for operation arguments in Rhapsody now appear as comments in the generated code.

  This change in behavior is due to the following changes:
  - New operation, *args_description*, added in file */src/Logical_Model_Elements_Generation/Rhapsody_Operation.java*
  - Call to *args_description* was added in *specification* script in file *src/Logical_Model_Elements_Generation/Rhapsody_Operation.tgs*.
  - New script, *argument_description*, added in file *src/Logical_Model_Elements_Generation/Rhapsody_Variable.tgs*.
  - New guard operation, *guard_argument_description*, added in file */src/Logical_Model_Elements_Generation/Rhapsody_Variable.java*

C++ — When generating code in Respect mode, you can expect to see certain differences compared with your code that was previously generated in Rhapsody, for example, the order of elements in code.

C++ — For the Cygwin and Linux environments, the default value of the property `CPP_CG::<environment>::UseTemplateTypename` has been changed from *False* to *True*. This means that generated code will use the `typename` keyword where necessary to prevent possible compilation errors.

Java — The property `JAVA_CG::Class::ComplexityForInlining` has been removed, and the property `CG::Class::ComplexityForInlining` no longer has any effect on code generation in Java.

Reverse Engineering / Roundtripping

C++ — The default value of the property `CPP_Roundtrip::General::CreateFileAsUnit` has been changed from *Default* to *AsModel*. This means that a component file created during reverse engineering will be saved as a unit only if the element that is being reverse engineered is saved as a unit. (When the value of the property is *Default*, the component file is saved as a unit only if the model-level property `General::Model::ComponentFileIsSavedUnit` is set to *True*.)

Framework

C++

IOxfEventSender

A new interface, `IOxfEventSender`, has been added, containing event sending operations. If you set the new property `CPP_CG::Class::ReactiveInterfaceScheme` to *Thin*, reactive interfaces will inherit from `IOxfEventSender` rather than `OMReactive`. The default value of the property is *Full*.

Note that `IOxfEventSender` includes only operations related to event sending, while `OMReactive` includes also attributes and operations related to statechart behavior.

When `ReactiveInterfaceScheme` is set to *Thin*, Rhapsody checks another new property, `CPP_CG::Framework::EventSender`, for the name of the base class to use for reactive interfaces. This property can be used if you want to provide your own interface for event-sending behavior. The default value for this property is *IOxfEventSender*.

IOxfReactive

`IOxfReactive` now inherits event sending operations from `IOxfEventSender`.

OMTimeout

The content of the method `OMTimeout::cancel()` has been modified to prevent problematic application behavior that had been reported.

Error Handling on VxWorks

There were cases where VxWorks would return an error but the relevant Rhapsody framework function would return void, resulting in the loss of this error information.

Now, when these framework functions encounter such an error, they call a macro called `OM_NOTIFY_ERROR`. By default, this macro is empty, but you can provide content to achieve the error-handling behavior that you require.

Upgrading to Version 7.1

Changes in Version 7.1

The changes in version 7.1 of Rhapsody are listed below.

Code Generation

C	The formatting of code generated when using customizable code generation has been enhanced in terms of use of whitespace.
C	When using customizable code generation, if the model contains flowports, then the file *FlowportInterfaces.h* will now contain forward declarations for each flowport interface, and the file *FlowportInterfaces.cpp* will now contain include statements for each of the flowport interfaces.
C	Due to a change in the way reactive instances are destroyed, you will notice changes in the code that handles the destruction of such instances.
C	The code generated when using the MISRA98 profile has been modified to be more MISRA-compliant, for example, int has been replaced with RhpInteger, and *break*-s used for leaving for loops have been replaced by an additional condition check.
C	In the C framework, a number of fields have been added to *RiCReactive_Vtbl*. With some compilers, this may lead to MISRA violation warnings.
C	For the INTEGRITY and Multi environments, in file paths in makefiles, backslashes have been replaced by slashes.
C	When using customizable code generation, *classInstance* annotations for blocks have been replaced by *block* annotations.
C, C++	For the INTEGRITY and Multi environments, the default value for the property *QuoteOMROOT* has been changed to True in order to facilitate the use of spaces in paths.
C, C++	Prior to version 7.1, an external VB-based makefile generator was used for the INTEGRITY and Multi environments. In 7.1, the internal Rhapsody generator is used.

C, C++ In version 7.1, if a package file contains associations, Rhapsody generates a constructor that initializes the pointers that are generated to implement the associations.

C, C++ The common annotation previously used for typedefs has been replaced by specific annotations that relate to the name of the type. These annotations were added to support roundtripping of modeled types.

C, C++ When destroying a reactive instance inside another instance, Rhapsody now uses a function called `RiCReactive_DelayedDestroy`, which was added to the framework, instead of `_Destroy`.

C, C++ After reverse engineering, generated code now contains an include to `oxf/oxf.h`, if a backward compatibility profile is used.

C, C++ Beginning with Rhapsody 7.0 MR-1, on Linux, variables of type OMBoolean were no longer mapped to *true* or *false* in the trace log. Instead, *0* or *1* were displayed. Now, the original behavior has been restored - *true* and *false* are used once again.

C, C++ In version 7.1, when the property `CG::Type::GenerateDeclarationDependency` is set to `False`, #include is not generated for the type definition for an operation return type.

C++ The prototype for the function `getItsWebAdapter()` is now generated in a different location, in the same file.

C, C++, Java In certain cases, for example, where multiple parts are used, the generated code now checks the return value of the method `startBehavior()`, something which was not done previously.

Java New annotations have been added before import statements in Java code. These have been added to support roundtripping of imports in RiJ.

Ada History connectors now behave like deep history connectors.

Reverse Engineering / Roundtripping

C For C models from versions prior to 5.5, the property `C_Roundtrip:General:RoundtripScheme` will be given the value *Advanced* when opened for the first time in version 7.1.

C, C++ In 7.1, Rhapsody uses by default the "smart" reverse engineering option (the *SmartPackageAndComponent* option in the Dependencies drop-down list on the Mapping tab). In this mode, each include statement results in only one dependency in the model. You may therefore find that certain dependencies in your model have "disappeared" after reverse engineering with version 7.1.

Framework

Ada

A new behavioral framework is now available. This framework relies on Ada 95 constructs as opposed to the previous version, which was limited to Ada 83 constructs.

C

Prior to 7.1, before a reactive instance was destroyed, the events addressed to it were removed from the queue. Now, a delayed destroy mechanism is used, whereby the reactive instance raises a flag and sends a termination event to the queue. The flag prevents events from being carried out as they are processed in the queue. When the end of the queue is reached, the termination event is handled and the instance is destroyed. A number of changes have been made to the C framework to implement this new delayed destroy mechanism:

RiCReactive

- `RiCReactive_Vtbl`—two pointers added: `cleanupMethod`, `freeMethod`.
- New function added—`RiCReactive_DelayedDestroy`
- New static function added—`handleEventUnderDestruction`
- New static boolean variable—`globalSupportDirectDeletion` (for backward compatibility), along with `RiCReactive_setGlobalSupportDirectDeletion` and `RiCReactive_shouldSupportDirectDeletion` for setting and testing the value of the variable.

RiCTask

- `RiCTask_execute`—modified such that it calls `RiCReactive_shouldSupportDirectDeletion`, and if false, calls the new function `RiCReactive_DelayedDestroy`. Otherwise, uses its previous behavior.
- `RiCTask_cancelEvents`—now only deletes events from queue if call to `RiCReactive_shouldSupportDirectDeletion` returns true.

C++

For Solaris, timers have been switched to work with *nanosleep*, instead of the signals mechanism.

Properties

C, C++ The possible values for the properties `C_Roundtrip:General:RoundtripScheme` and `CPP_Roundtrip:General:RoundtripScheme` have been changed.
For C, the new values are *Basic* and *Advanced*.
For C++, the new values are *Basic*, *Advanced*, and *Respect*.
For models from versions prior to 7.1, if this property has been overridden, you will see

the old values. In such cases, if you want to choose *Respect*, you have to first "un-override" the property.

C, C++ In the property files, the metaclasses *VxWorks6.2diab*, *VxWorks6.2diab_RTP*, *VxWorks6.2gnu_RTP*, *VxWorks6.2gnu* have been renamed to *VxWorks62diab*, *VxWorks62diab_RTP*, *VxWorks62gnu_RTP*, *VxWorks62gnu*, respectively.

C, C++ The property `[lang]_Roundtrip::Type::Ignore` has been removed.

C++ The property `CPP_ReverseEngineering::ImplementationTrait::CreateDependencies` can now take the value `SmartPackageAndComponent`. This ensures that only once dependency is creates in a model for each include statement.

Java The property `CG::Component::InitializationScheme` can now take the value `ByComponent` to support links across packages in Java.

Java The default value for the property `Java_RoundTrip::General::RoundTripScheme` is now `Advanced`, rather than `Basic`.

Other Changes

- The name of the metaclass representing object nodes in activity diagrams has been changed from *State* to *ObjectNode*.
- pSOS is no longer available as a target environment.
- OsePPCDiab is no longer available as a target environment.
- The uninstallation process no longer contains a *Repair* option.

Automatic Upgrade Performed by Rhapsody

When pre-7.1 models are loaded, Rhapsody loads the profile CGCompatibilityPre71. This profile includes all settings that are required to ensure backward compatibility with older models. Separate profiles are provided for Ada, C, C++, and Java.

Changes that May Necessitate User Action

Code Generation

C++ For Solaris, timers have been switched to work with nanosleep =, instead of the signals mechanism.
To use the signals mechanism instead, define the flag OM_USE_SIGALRM_BASED_TIMER in the file omosconfig.h.

Ada — History connectors now behave like deep history connectors.
To use the previous connector behavior, set the value of the property Ada_CG.Statechart.HistoryConnectorDepth to Shallow. (For pre-7.1 models, the value of the property defaults to Shallow.)

Reverse Engineering / Roundtripping

C — For C models from versions prior to 5.5, the property `C_Roundtrip:General:RoundtripScheme` will be given the value `Full` when opened for the first time in version 7.1.
To use Basic roundtripping, set the value of this property back to Basic.

Framework

Ada — A new behavioral framework that relies on Ada 95 constructs is available.
To select the version of the framework to use for a given class and/or package combination, set the values of the properties Ada_CG::Class::UseAda83Framework and Ada_CG::Package::UseAda83Framework accordingly. (For pre-7.1 models, the Ada 83 framework is activated by default.)

C — For destroying reactive instances, Rhapsody now uses a new delayed mechanism for new models, and the old direct deletion mechanism for older models.
Each of these mechanisms handles the deletion of the reactive instance in a safe manner.
However, if you try to use the old _Destroy function in new models (where the property `UseDirectReactiveDeletion` is set to False), you will encounter problems.

Other Changes

The name of the metaclass representing object nodes in activity diagrams has been changed from *State* to *ObjectNode*.
If you have used the name of this metaclass, you will have to replace the reference with the new name of the metaclass.

Backward Compatibility Settings

Note

Backward compatibility profiles are used to set property values in order to maintain previous behavior in cases where Rhapsody's default behavior has been changed. Keep in mind that the property values in these compatibility profiles always take precedence over project-level property value overrides that you may have made in your existing models. If you have such project-level overrides for properties included in the compatibility profile, you will have to un-override the property values from the compatibility profile after opening the model for the first time in the new version of Rhapsody.

The CGCompatibilityPre71 profiles contain the following properties to ensure backward compatibility with pre-7.1 models. For each property, the relevant languages are indicated.

Code Generation

Ada `Ada_CG::Class::UseAda83Framework`
In 7.1, a new behavioral framework that relies on Ada 95 constructs was introduced, and this framework is used by default.
In the compatibility profile, the property `Ada_CG::Class::UseAda83Framework` is set to True so that the older framework is used for pre-7.1 models.

Ada `Ada_CG::Package::UseAda83Framework`
In 7.1, a new behavioral framework that relies on Ada 95 constructs was introduced, and this framework is used by default.
In the compatibility profile, the property `Ada_CG::Package::UseAda83Framework` is set to True so that the older framework is used for pre-7.1 models.

Ada `Ada_CG::Statechart::HistoryConnectorDepth`
In 7.1, history connectors now behave like deep history connectors.
In the compatibility profile, the property `Ada_CG::Statechart::HistoryConnectorDepth` is set to `Shallow` (rather than `Deep`) to restore the old behavior for pre-7.1 models.

C `C_CG::Class::InterfaceGenerationSupport`
Version 7.1 introduced the ability to realize interfaces in C.
In the compatibility profile, the property

`C_CG::Class::InterfaceGenerationSupport` is set to False to restore the previous behavior for pre-7.1 models.

C — `C_CG::Framework::UseDirectReactiveDeletion`
In 7.1, a new mechanism was introduced for destroying reactive instances.
In the compatibility profile, the property `UseDirectReactiveDeletion` is set to True to restore the previous behavior for pre-7.1 models.

C, C++ — `[lang]_CG::INTEGRITY5::InvokeMakeGenerator`, `[lang]_CG::Integrity5ESTL::InvokeMakeGenerator`, `[lang]_CG::Multi4Win32::InvokeMakeGenerator`
Prior to 7.1, an external VB-based makefile generator was used for these environments.
In 7.1, the internal Rhapsody generator is used.
In the compatibility profile, these properties have been set to use the external makefile generator that was used previously, for pre-7.1 models. For INTEGRITY5, the value is `$OMROOT/etc/Integrity5MakefileGenerator.bat`. For `Integrity5ESTL`, the value is `$OMROOT/etc/Integrity5MakefileGenerator.bat`. For Multi4Win32, the value is `$OMROOT/etc/MultiMakefileGenerator.exe`.

Java — `JAVA_CG::Dependency::GenerateOriginComment`
In 7.1, Rhapsody now generates annotations for dependencies, which explain for each import statement why it was generated.
In the compatibility profile, the property `GenerateOriginComment` is set to False to restore the previous code generation behavior for pre-7.1 models.

C, C++, Java — `[lang]_CG::flowPort::InvokeRelay`
Version 7.0 of Rhapsody included a change to flowport behavior. Previously, updated data was always sent to the flowport, regardless of whether or not the data had changed. As of 7.0, by default, the data is sent only if the attribute value has changed.
In the compatibility profile, the property `InvokeRelay` is set to `Always` (rather than `UponAttributeChange`) to restore the previous flowport behavior for pre-7.0 models.

Reverse Engineering / Roundtripping

C++ — `CPP_ReverseEngineering::ImplementationTrait::UsePackageForExternals`
In 7.1, by default, external elements are imported into a dedicated package.
In the compatibility profile, this property is set to False to restore the previous behavior for importing external elements.

C++ — `CPP_ReverseEngineering::ImplementationTrait::CreateDependencies`
In version 7.1, the reverse engineering process was refined so that only one model dependency would be created for each dependency that appears in the code. This is

represented by the value `SmartPackageAndComponent` for the property `CreateDependencies`.
In the compatibility profile, this value of this property is set to `PackageAndComponent` to restore the previous reverse engineering behavior for dependencies for pre-7.1 models.

C, C++ `[lang]_ReverseEngineering::ImplementationTrait::ImportPreprocessorDirectives`
In 7.1, the reverse engineering feature can handle all types of C/C++ preprocessor directives, such as #ifdef.
In the compatibility profile, the property `ImportPreprocessorDirectives` is set to False to restore the previous reverse engineering behavior for pre-7.1 models.

C, C++ `[lang]_Roundtrip::Type::Ignore`
In 7.1, this property has been removed.
In the compatibility profile, this property is included with a default value of True to restore the previous roundtripping behavior for pre-7.1 models.

C, C++, Java `[lang]_ReverseEngineering::ImplementationTrait::CreateFolderByPath`
Version 7.1 introduced an improved folder hierarchy creation approach for reverse engineering.
In the compatibility profile, the property `CreateFolderByPath` is set to False to use the previous folder hierarchy approach for pre-7.1 models.

Java `JAVA_Roundtrip::General::RoundtripScheme`
In 7.1, the default value of the property `RoundtripScheme` has been changed to `Advanced`.
In the compatibility profile, the value of this property is set to `Basic` to restore the previous roundtripping behavior for pre-7.1 models.

Upgrading to Version 7.0 MR-3

Changes in Version 7.0 MR-3

The changes in version 7.0 MR-3 of Rhapsody are listed below.

Code Generation

C, C++ Previously, for arguments, if the user set the property `CreateImplicitDependencies` to False, implicit dependencies were still generated in the code. Now, this is no longer the case. If the property is set to False, implicit dependencies will not be generated.

Framework

C++ The operation `consumeTime` has been restored to the class `OMTimerManager`.

Upgrading to Version 7.0 MR-2

Changes in Version 7.0 MR-2

The changes in version 7.0 MR-2 of Rhapsody are listed below.

Code Generation

C	Handling of the property `ReusableStatechartSwitches` was changed. For the implications of this change, see **Changes that May Necessitate User Action**.
C	When code is generated using the customizable code generation mechanism, auto-generated code now appears below user code in Cleanup operations.
C, C++	In code generated for Simulink integration, include statements now enclose file paths in quotation marks (““) rather than angle brackets (<>). This was done to solve problems with file paths that included spaces.
C++	Previously, Rhapsody would add empty namespace declarations in implementation files when the user instantiated a template and had DefineNameSpace=true. Now, these unnecessary namespace declarations are no longer generated.
C++	If a class contains an association end that is a container, and the multiplicity is not an absolute number, initialization code is now added to the constructor.
Java	The method `startBehavior` is no longer generated for Java interfaces.

Framework

C++ The following changes were made in the file `vxos.cpp` for VxWorks RTP support:

- In RTP mode, the operation `VxOSMessageQueue::getMessageList` is empty.
- In the operation `VxOSMessageQueue::isFull`, references to fields that don’t exist in RTP mode have been enclosed in an `#ifdef`.
- A number of include statements have been excluded for RTP mode.

C++ In the file `linuxos.cpp`, `#include <unistd.h>` is always included. (Previously, had been under _OMINSTRUMENT.)

C++ In the file `state.cpp`, the inconsistency between generated and supplied source files was eliminated.

C++ In the file `vxoxf.mak`, RTP support was added.

Other Changes

- aom library:
  - Files `aomdisp.h, aomdisp.cpp`

  A new argument, `void* destOrSource`, was added to the operation `AOMSchedDispatcher::sendForeignMessage`.
  - File `aomclass.h`

  Two new virtual functions, `notifyGotControl` and `notifyLostControl`, were added.
- tom library:
  - Files `tomdisp.h, tomdisp.cpp`

  A new argument, `void* destOrSource`, was added to the operation `TOMDispatcher::sendForeignMessage`.

Changes that May Necessitate User Action

Code Generation

C The handling of the property `ReusableStatechartSwitches` was changed such that if you manually add the `-D` switch, you could end up with the switch appearing twice. The new version of the MultiMakefileGenerator script for the Integrity environment was modified to overcome this problem—if you add the `-D` switch to the property value, it will not be added a second time.
If you have customized the MultiMakefileGenerator script, you will have to modify your version of the script to prevent this problem from occurring or integrate your customizations into the updated MultiMakefileGenerator script.

Upgrading to Version 7.0 MR-1

Changes in Version 7.0 MR-1

The changes in version 7.0 MR-1 of Rhapsody are listed below.

Framework - Linux

C The default value of thread priority has been changed to 0.

Previous value:

```
const RiC_ThreadPriorityType RiCOSDefaultThreadPriority = PRIO_NORMAL
```

where

```
PRIO_NORMAL = 30
```

New value:

```
const RiC_ThreadPriorityType RiCOSDefaultThreadPriority = 0
```

Properties

A property named `CG::General::ReportToOutputWindow` was added. This property can take the values `Basic` or `Detailed` (default value). When set to `Basic`, only a subset of the output messages are shown in the Output window during code generation. This can improve code generation performance, especially on Linux.

When set to `Basic`, you can still see all code generation messages by adding *ReportToLogFile=TRUE* in the *[CodeGen]* section of the rhapsody.ini file. This will result in all messages being written the file *Generation.log* in the project directory.

Upgrading to Version 7.0

Changes in Version 7.0

The changes in version 7.0 of Rhapsody are listed below.

Code Generation

C	For a qualified relation, the function getKey now casts the result before returning it.
C	For the Integrity environment, the MultiMakefileGenerator script was modified to support the event across address space feature.
C, C++	Extraneous semicolon after closing brace "}" has been removed.
C, C++	When generating template-based code descriptions, if there is no value for a keyword, an empty string is now returned rather than the keyword itself, as previously.
C, C++	Where file paths appear in annotations in the code, they are now presented in a more intuitive manner.
C, C++	Now, for events, default constructors are generated only if the event doesn't have parameters or if code is being generated for animation or tracing.
C, C++	Since Rhapsody now allows you to have two operations with the same name and arguments if one is defined as *const*, the annotation for const operations now contains the string `const` after the operation name and arguments.
C, C++	Now, code is generated for dependencies on component files.
C, C++	In the clean-up code for containers, there is now a null-pointer check to verify that the container exists.
C++	Clean-up code generated for relations no longer includes unused variables.
C++	Now, files are generated for all component files defined, even if no elements have been mapped to them.

C++ Previously, if an external class was included in the scope of code generation, the code was generated as if a dependency existed between the package and the external class (forward declarations and #include). Now, this code is no longer generated.

C++ As part of the Simulink integration feature, a variable called SimulinkLibName was added to the makefile.

C++ Changes have been made in terms of the location of a number of types of code fragments within a file (comments, package annotations, forward declarations, preprocessor directives). Also, certain comments appear now only once in a file rather than being repeated for each element mapped to the file.

C++ If all of the primitive operations of a class/interface are abstract, then the destructor of the class will be abstract too, by default, provided that the property `CPP_CG::Class::Destructor` is set to `Auto`, which is the default value. To change this behavior, change the value of this property.

C++ Destructors of interfaces are now virtual by default.

C++ The formatting of code generated for ports has been enhanced in terms of use of whitespace.

C++ Two empty lines were added to the code generated for COM and CORBA to align the code generation with that of ordinary .cpp files.

C++ The keyword *static* is now generated for variables with initial values that are set to be both Constant and Static. (To avoid generation of the *static* keyword, unselect the Static option in the Features dialog box.)

C++ When events are derived from events with arguments, initialization of the base event's arguments is now done in the constructor of the base event rather than in the constructor of the derived class.

Java The code generation mechanism now uses containers that take advantage of the new capabilities included in JDK 1.5.

Java The default value of the property `JAVA_CG::JDK::PathDelimiter` has been changed from a backslash (\) to a forward slash (/).

Ada If you have modified the Ada code generator rules, please read *Upgrading from the UML Meta-model to the Rhapsody Meta-model* in the RiA documentation.

Reverse Engineering

C, C++ Now, structs that contain nested structs/classes are imported as classes, rather than types.

C, C++	Now, by default, reverse engineering goes through all "include" files and collects any macros defined in them.
C++	Now, by default, reverse engineering creates separate specification and implementation component files, rather than a logical file, as previously.
C++	Now, by default, reverse engineering creates component files in the model.
C++	Now, by default, Rhapsody creates dependencies for elements in component files, rather than only for elements under packages.
C++	Now, by default, reverse engineering imports global variables as private if they are declared as `static` in implementation files.
C++	Now, by default, reverse engineering maps global variables, functions, and types to component files, reflecting their original file locations.

Framework

C	Significant changes were made to the OXF, including code cleanup and additional documentation.
C	In the OXF, RiCSystemTimer is no longer a global variable. Instead, it is accessed via RiCTimerManager_getSystemTimer.
C	In the file `RiCQueue.c`, the function `increaseTail_` was modified to solve problem of memory overflow when using fixed-size queue.
C	In the file `RiCTimer.c`, the function `RiCTimerManager_unschedTm` was modified to solve problem of timeouts not being removed from the heap.
C	In the file `RiCTask.c`, the function `RiCTask_execute` was modified.
C	For support of ports in C, the following classes and interfaces were added to the C framework: `RiCDefaultReactivePort`, `RiCDefaultReactiveOutbound`, `RiCDefaultReactiveInbound`, `IRiCDefaultReactive`.
C	For the vx makefiles (vxbuild.mak, vxaom.mak, vxomcom.mak, vxOxf.mak, vxWebComponents.mak), when you use the CFG parameter, the value you provide should not include the *vx* prefix. For example, the value should be *oxf* and not *vxoxf*.
C	To enable the Events across Address Spaces feature, the following changes were made to the framework: ◆ Added file `RiCAddressSpace`—contains static buffer called `AddressSpaceName`.

- In the file `RiCReactive.h`, a new attribute called `registeredId` was added to the struct `RiCReactive`.
- File `RiCTask`

  Added the functions `RiCTask_destroyEvent` and `NotifyAnimQueueEvent`.

  Added following functions under the preprocessor flag RIC_DISTRIBUTED_SYSTEM: `RiCTask_initDistributed`, `RiCTask_InitDistributed`, `RiCTask_createDistributed`, `RiCTask_CreateDistributed`.

  Two additional parameters—`toDistributeQueue` and `queuePublishedName`—were added to the init function in `RiCTask`.
- File `RiCOSWrap.h`

  Added following functions under the preprocessor flag RIC_DISTRIBUTED_SYSTEM: `RiCOSMessageQueue_initDistributed`, `RiCOSMessageQueue_createDistributed`, `RiCOSMessageQueue_getMessageQueueId`, `RiCOSMessageQueue_isMessageQueueIdString`, `RiCOSMessageQueue_getRegisteredId`
- File `RiCONST.h`

  Under the preprocessor flag RIC_DISTRIBUTED_SYSTEM, the attributes for `RiCIntMessageQueue` are: `m_hQueueWnd`, `m_pCopyData`, `m_ToDistributeQueue`, `m_QueuePublishedName`, `m_RegisteredId`.
- File `RiCONST.c`

  Alternative implementations of the RiCOSMessageQueue functions were added under the preprocessor flag RIC_DISTRIBUTED_SYSTEM.
- File `RiCEvent.h`

  Added the function `RiDSendRemoteEvent` under the preprocessor flag RIC_DISTRIBUTED_SYSTEM

  Added the macro `RiCGENREMOTE` under the preprocessor flag RIC_DISTRIBUTED_SYSTEM
- File `RiCOXF.c`

  Alternative implementation of the `init` function was added under the preprocessor flag RIC_DISTRIBUTED_SYSTEM.
- IntegrityBuild.bat—If you use this file to rebuild the INTEGRITY libraries, you must include the command-line parameter *distributed* if you want the libraries to include support for the Events across Address Spaces feature.

C INTEGRITY: To enable the Events across Address Spaces feature for the INTEGRITY operating system, the following changes were made:

- File `RiCOSIntegrity.h`

Under the preprocessor flag RIC_DISTRIBUTED_SYSTEM, the attributes for `RiCIntMessageQueue` are: `m_MessageQueue`, `m_MessageQueueBuffer`, `m_pMessageQueueBuffer`, `m_ToDistributeQueue`, `m_QueuePublishedName`, `m_RegisteredId`.

- File `RiCOSIntegrity.c`

  Alternative implementations of the RiCOSMessageQueue functions were added under the preprocessor flag RIC_DISTRIBUTED_SYSTEM.

C++ Following functions defined as inline to improve performance during event processing:

- `OMEvent::isTypeOf`
- `OMEvent::getId`
- `OMEvent::getlId`
- `OMEvent::setId`
- `OMReactive::shouldTerminate`
- `OXF::getRhp5CompatibleAPI`
- `OXF::setRhp5CompatibleAPI`

C++ Operation `OMReactive::send` now uses static variable `OMOSEventGenerationParams`.

C++ Call to function `getCurrentEvent` was replaced by direct usage of `currentEvent`.

C++ Checks were moved from `OMReactive::processEvent` to `OMReactive::handleTrigger`.

C++ Copy constructor and `operator=` function added to class `OMProtected`.

C++ Function `CancelTimeouts` was added to class `OMReactive`.

C++ Change in implementation of function `OMReactive::scheduleTimeout`. Now, if timeout heap if full, tries to empty cancelled timeouts.

C++ Function isHeapFull added to class OMTimerManager.

Rhapsody API

- The interface `IRPStereotype` is now derived from `IRPClassifier` (instead of `IRPModelElement`).

Other Changes

- aom library:
  - File `aomNotifyUtils` was added.
  - In class `aomItem`, operations `notifyGotControl` and `notifyLostControl` were changed to virtual.

  - To increase efficiency of the NOTIFY_OPERATION animation macro, a utility service was added.

- omcom library:
  - Changes were made in the way that elements are serialized/unserialized in animation/tracing for Linux and Cygwin.
- The name of the Harmony profile has been changed from *HarmonyProfile* to *Harmony.* Also, the location of the profile has been changed from */Share/Profile/* to */Share/Profile/Harmony/*.

Automatic Upgrade Performed by Rhapsody

When pre-7.0 models are loaded, Rhapsody loads the profile CGCompatibilityPre70. This profile includes all settings that are required to ensure backward compatibility with older models. Separate profiles are provided for C, C++, and Java.

Changes that May Necessitate User Action

Code Generation

C — For the Integrity environment, the MultiMakefileGenerator script was modified to support the event across address space feature, by generating an integration file and linking the necessary libraries when the property C_CG::Configuration::Distribution is set to True.
If you have customized MultiMakefileGenerator, and you are planning to change the value of C_CG::Configuration::Distribution to True in order to allow usage of the multiple address space feature, you will need to integrate your customizations into the updated MultiMakefileGenerator script.

C++ — Variable named SimulinkLibName was added to the makefile. This variable gets a value if a Simulink block exists, otherwise it remains null.
If you have customized the makefile, and you wish to use it with a Simulink block, you will need to update your makefile template.

C++ — Before 7.0, if an external class was included in the scope of code generation, the code was generated as if a dependency existed between the package and the external class (forward declarations and #include). This code is no longer generated.
If you require this, you should add the dependency manually.

C++ — Destructors of interfaces are now virtual by default.
If you would like to restore the previous code generation behavior, you can customize the predefined types package as follows:

Note: *Be sure to create a backup copy of the original Predefined Types packages before attempting this.*

1) Add the Predefined types package (for example, <Rhapsody install>\Share\ Properties\PredefinedTypesC++.sbs) to your model by value (as a unit) which replaces the read-only reference to the predefined package.

2) Make the necessary edits to this unit that you have added and then save it.

3) Replace the unit in the <Rhapsody install>\Share\Properties directory with the unit that you have created (which is saved in the <model>_rpy directory).

Framework

C — In the OXF, RiCSystemTimer is no longer a global variable. Instead, it is accessed via RiCTimerManager_getSystemTimer.
If you have used RiCSystemTimer in your code, you will have to modify your code.

C — For support of ports in C, the following classes and interfaces were added to the C framework: `RiCDefaultReactivePort`, `RicDefaultReactiveOutbound`, `RiCDefaultReactiveInbound`, `IRiCDefaultReactive`.
If you have existing models that specified ports that relay events (rapid ports and ports that only have event receptions in their contract), and implemented the ports by specifying attributes and operations on the class, you should disable the automatic code generation for the ports (set property C_CG::Port::Generate to False for the port) or revise the implementation.

C++ — Call to function `getCurrentEvent` was replaced by direct usage of `currentEvent`.
If you overrode this function, your modifications will not have an effect.

C++ — Change in implementation of function `OMReactive::scheduleTimeout`. Now, if timeout heap if full, tries to remove cancelled timeouts.
If your code included steps to deal with this problem, these steps are no longer necessary.

Other Changes

The name of the Harmony profile has been changed from *HarmonyProfile* to *Harmony.* Also, the location of the profile has been changed from */Share/Profile/* to */Share/Profile/Harmony/*.
If you load a pre-7.0 model that used the Harmony profile, Rhapsody indicates that it cannot find the profile. To restore the Harmony profile to the model:

1. When Rhapsody displays the *Search for file* dialog, asking for the location of the profile, click *Ignore*.

2. Right-click your Harmony project in the browser to display the context menu, and select *Change To* > *Project*. (At this point, it is no longer considered a Harmony project.)

3. In the browser, under Profiles, right-click *HarmonyProfile* (which is now unreferenced), and from the context menu, select *Delete from Model*.

4. Add the new Harmony profile as a reference: Select *File* > *Add to Model* from the main menu, find the file *Harmony.sbs* under */Share/Profile/Harmony/*, select the *As Reference* option, and click *Open*.

5. Change the project back to a Harmony project by right-clicking the project in the browser and selecting *Change To* > *Harmony* from the context menu.

Backward Compatibility Settings

The CGCompatibilityPre70 profiles contain the following properties to ensure backward compatibility with pre-7.0 models. For each property, the relevant languages are indicated.

Code Generation

C `RiCContainers::Qualified::GetKey`
In 7.0, the function getKey now casts the result before returning it.
In the compatibility profile, this property is set to the value $(CType)_getKey(&($me$cname), (gen_ptr)$keyName). This restores the old behavior.

C, C++ `CG::Dependency::ForwardDeclarationPlacement`
In 7.0, code is generated for dependencies on component files.
In the compatibility profile, this property is set to the value BeforeElements, causing the code generation mechanism to refrain from generating code for such dependencies.

C, C++ `CG::Event::ForceDefaultConstructor`
In 7.0, for events, default constructors are generated only if the event doesn't have parameters or if code is being generated for animation or tracing.
In the compatibility profile, this property is set to True, causing the code generation mechanism to always generate a default constructor.

Java `JAVA _ CG::Configuration::ContainerSet`
In 7.0, the default value of this property is Java(1.5)Containers, and this instructs the Rhapsody code generation mechanism to take advantage of the new capabilities included in JDK 1.5.
In the compatibility profile, this property is set to its old value of Java(1.2)Containers.

Reverse Engineering

C, C++ `[lang]_ReverseEngineering::ImplementationTrait::CollectMode`
In 7.0, the default value of this property is Once, causing reverse engineering to go through all "include" files and collects any macros defined in them.
In the compatibility profile, this property is set to the value None so that macros will not be collected.

C++ `CPP_ReverseEngineering::ImplementationTrait::ComponentFileType`
In 7.0, the default value of this property is SpecificationOrImplementation, causing Rhapsody to create separate specification and implementation component files in reverse engineering.
In the compatibility profile, this property is set to its old value of Logical.

C++ `CPP_ReverseEngineering::ImplementationTrait::CreateDependencies`
In 7.0, the default value of this property is PackageAndComponent, causing Rhapsody to create dependencies for both elements in component files and elements under packages.
In the compatibility profile, this property is set to its old value of PackageOnly.

C++ `CPP_ReverseEngineering::ImplementationTrait::CreateFilesIn`
In 7.0, the default value of this property is Component, causing reverse engineering to create component files in the model.
In the compatibility profile, this property is set to its old value of None.

C++ `CPP_ReverseEngineering::ImplementationTrait::ImportGlobalAsPrivate`
In 7.0, the default value of this property is StaticInImplementation, causing reverse engineering to import global variables as private if they are declared as `static` in implementation files.
In the compatibility profile, this property is set to its old value of InImplementation.

C++ `CPP_ReverseEngineering::ImplementationTrait::MapGlobalsToComponentFiles`
In 7.0, the default value of this property is True, causing reverse engineering to map global variables, functions, and types to component files, reflecting their original file locations.
In the compatibility profile, this property is set to the value TypesOnExternal so that only types will be mapped and only if the user selected the reverse engineering option *Import as External*.

Upgrading to Version 6.2 MR-1

There are no upgrade issues for Rhapsody 6.2 MR-1.

Upgrading to Version 6.2

Changes that Require User Action

RiC++ OXF

OM_DECLARE_FRAMEWORK_MEMORY_ALLOCATION_OPERATORS

Applicable to users with custom adapters, that use compilers that do not support replacement delete operators.

Replacement new and delete operators were added to the definition of `OM_DECLARE_FRAMEWORK_MEMORY_ALLOCATION_OPERATORS`.

The replacement delete definition is guarded by ifndef `OM_NO_COMPILER_SUPPORT_FOR_REPLACEMENT_DELETE`

If you get a compiler error related to the replacement operator delete definition (for example, when using diab 4.3f), you should add #define `OM_NO_COMPILER_SUPPORT_FOR_REPLACEMENT_DELETE` to your adapter omosconfig.h file.

Adapters

VxWorks support

Applicable to RiC/RiC++ VxWorks 6.0 users

The VxWorks 6.0 environments (VxWorks6.0diab, VxWorks6.0gnu) were replaced by VxWorks 6.2 (VxWorks6.2diab, VxWorks6.2gnu).

The change was made in the RiC/RiC++ properties and includes the name change and a change in the InvokeMake properties.

You should move to the new environment even if you are actually using VxWorks 6.0.

Automatic Upgrade Performed by Rhapsody

RiC++ OXF

- `UseNullBlockContainter` (ifdef guard) was replaced by `OMUseNullBlockContainer`; full backward compatibility is provided in rawtypes.h
- `OM_NEED_THORW_IN_NEW_OPERATOR` spelling was fixed to `OM_NEED_THROW_IN_NEW_OPERATOR`; full backward compatibility is provided in rawtypes.h.
- `OM_DECLARE_COMPOSITE_OFFSET` definition moved from aom/aommacro.h to rawtypes.h in order to remove the dependency of non-instrumented code on the AOM subsystem. The declaration is required for non-instrumented code in order to allow the mix of instrumented and non-instrumented libraries.

Additional Information

Code Generation

- Meaningless package files that were generated in instrumented configurations (Animation or Tracing) are no longer generated.
- C++, Java: Getters of static attributes are generated as static operations instead of regular (non-static) operations.
- Activity diagrams for operations (C++): Redundant friend declaration was removed.
- *Block added after instrumented code in relation helpers (C):*
  In instrumented code for relation helpers, the action after the instrumentation code is now wrapped in braces to prevent errors on declaration of local variables.

  For example:

```
void Bus__addItsSensorSuperClass(Bus* const me, int key, struct
SensorSuperClass * p_SensorSuperClass) {
    if(p_SensorSuperClass != NULL)
    {
        NOTIFY_RELATION_ITEM_ADDED(me, Bus, SensorSuperClass,
"itsSensorSuperClass", p_SensorSuperClass, FALSE, FALSE);
    }
    else
    {
        NOTIFY_RELATION_CLEARED(me, Bus, "itsSensorSuperClass");
    }
```

```
    {
        int pos;
        for(pos = 0; pos < 360; ++pos) {
            if (!me->itsSensorSuperClass[pos]) {
                me->itsSensorSuperClass[pos] = p_SensorSuperClass;
                break;
            }
        };
    }
}
```

- *Missing initialization now generated (C, C++, Java):*

  When operation ordering was used, there were some cases where automatically generated initialization code was missing from the default constructor.

  This issue was fixed and the automatic initialization is now fully generated.

  For example:

  When class_0 has an association to class_1, and the operation order was modified for class_0, the following code will be generated.

```
void class_0_Init(class_0* const me) {
    me->itsClass_1 = NULL;
    initRelations(me);
}
```

  Before the issue was fixed, the following code would have been generated:

```
void class_0_Init(class_0* const me) {
     initRelations(me);
}
```

RiC IDF

`RiCOXFInit()`: The initialization calls were modified so that the `initRelations()`packages (e.g. `Core_initRelations()`) are called directly instead of through <package>`_OMInitializer_Init()`.

RiC++ OXF

- Linux: Initialization of the `pthread_mutexattr_t` local variable was added to the `LinuxMutex` constructor.
- Nucleus: NULL was replaced by dummy local variables in RTOS calls.
- Handling of canceled timeouts in the timer's heap:

  Starting from Rhapsody 6.0, the OXF provided a timeout cancellation scheme that is based on the reactive instances themselves. This scheme requires less interaction with the timer manager and is therefore more effective.

  However, canceled timeouts are left inside the timer manager timeout OMHeap until their due time and only then are they destroyed. As a result, the canceled timeouts use a portion of the timeout heap and may increase the probability of an overflow.

  To address this issue, we have made it possible for the user to recognize this situation and clean up the canceled timeouts from the list.

  For this purpose, the following changes were made:

  – OMHeap<Node>
  The add(Node*) operation return type was changed from void to bool. The operation returns false if the add failed due to a full heap.
  In addition, the message in the event of an error was changed from "Timer heap overflow" to "Heap overflow".

  – OMTimerManager
  A new public operation: `RP_FRAMEWORK_DLL bool cleanupCanceledTimeouts()`was added.
  The operation removes canceled timeouts from the heap. It returns true if canceled timeouts were removed.

  The return type for the set(IOxfTimeout*) operation was changed from void to bool. The operation returns false if the addition of the timeout to the timeout heap failed.

  – bool operator ==(const IOxfTimeout&, const IOxfTimeout&):
  In normal mode (not managed timeout canceling), returns true if both timeouts are canceled before checking the due time.

  – OMReactive
  A new protected virtual operation, `void handleTimeoutSetFailure(IOxfTimeout*)` was added.
  This operation is called when the setting of a timeout failed. This happens when the timer manager cannot add the timeout to the waiting timeout heap.

  The user should override this in derived classes to handle the error. For example, the user may call

`OMTimerManager::instance()->cleanupCanceledTimeouts()` and then retry setting the timeout.

`scheduleTimeout(OxfTimeUnit, const char* = 0)`: calls `handleTimeoutSetFailure()` if the call to `OMTimerManager::instance()->set()` returns false.

(void) cast was added before calling `OMTimerManager::set()` in `OMDelay` constructor and the `OMThread::schedTm()` pre-6.0 compatibility operation to avoid LINT warnings.

– `OMList<class Concept>::Item::operator =(const Item&)`: Added missing return statement.

Upgrading to Version 6.1 MR-2

Changes that Require User Action

COM API

In the interface `IRPFileFragment`, the type of the property `fragmentElement` has been changed to `IRPModelElement`.

Code Generation

Initialization of C++ Instances Realizing CORBA Interfaces

Applicable to C++ CORBA users

Rhapsody 6.1 MR-1 introduced the ability to initialize instances of common C++ composite classes that contain parts that realize CORBA interfaces.

This meant that the expansion of the `$instance` keyword in the `CORBA::<ORB>::InitialInstance/DestroyInitialInstance` properties included the composite instances as well as instances that realize CORBA interfaces directly.

Since the initialization of the parts of the composite was not automatically generated, requiring the user to handle the completion of the initialization, this behavior was removed in Rhapsody 6.1 MR-2.

If you used this feature, you can enable it by adding the boolean property `CORBA.Class.CppCompositeInitialization` and setting its value to `true`.

Additional Information

Framework

OXF

Addition of Missing Initialization

Applicable to C++.

In `OMEventQueue` and `OMTMMessageQueue`, initialization of the queue to 0 was added in the non-default constructor.

Upgrading to Version 6.1 MR-1

General Recommendations

Code Generation

Working with Rhapsody 6.1 and Rhapsody 6.1 MR-1 Simultaneously

Applicable to C, C++, and Java

Rhapsody lets you exchange models between Rhapsody 6.1 and Rhapsody 6.1 M1.

If you plan to work with both versions, you should be aware that Rhapsody 6.1 MR-1 provides several improvements related to the 6.1 release.

To avoid different results when generating code with the different versions, we recommend that you add the profile CGCompatibilityPre61M1<lang> (from <Rhapsody>/Share/Profiles) to the model.

This profile disables the MR-1 specific improvements, thereby ensuring compatible code generation results.

The CORBA Package

Applicable to RiC++ CORBA developers that are using the CORBA reference package provided with Rhapsody.

The CORBA package has been updated in Rhapsody 6.1 MR-1.

To ensure that you are using the updated version of the package, we recommend that you carry out the following steps:

1. Add the CORBA package to the model, by reference, from <Rhapsody>/Share/Properties (replace the existing package).
2. Select Unit > Edit Unit from the context menu to open the unit dialog, and change the unit path to "$OMROOT\Properties". This step will ensure that you continue using the updated version when upgrading to future releases of Rhapsody.

Changes that Require User Action

Code Generation

General

Implicit Dependencies:

Applicable to any model with:

- Two types with the same name, or a type and a class with the same name
- Enabled implicit dependencies (#include) generation (`CG::Class::CreateImplicitDependencies` = `TRUE`, `CG::Type::GenerateDeclarationDependency` = `TRUE`)

Rhapsody's automatic dependency generation mechanism was enhanced in order to avoid potential erroneous dependency generation when a potential ambiguity is found.

This means that when using a verbatim declaration for an argument or an attribute type, Rhapsody will not create an include/import statement if:

- There is more than one potential candidate type/class that matches the name.

As a result of this change, automatically-generated include statements may be removed from your source code.

Although these include statements are potentially wrong, there may be cases where these statements were correct and were used to compile the code successfully. In such rare cases, you will need to add `<<Usage>>` dependencies to the appropriate element(s) in order to generate the include statements.

Automatically generated dependencies

Applicable to C, C++ and Java

The automatic generation of dependencies (include/import statements) was improved.

These improvements reduce the number of generated dependencies to better suit the model.

In rare cases, where you took advantage of redundant #include or #import statements, you may witness compilation problems. In these cases, you'll need to model the appropriate <<Usage>> dependencies.

Redundant semicolons:

Applicable to C, C++ and Java.

Code generation was improved to avoid the addition of redundant semicolons after user code, such as transition actions and language type definitions. These redundant semicolons would occur when the code ended with a preprocessor directive (#endif), comment, or some user block termination (for (...) {}).

In some rare cases, if the redundant semicolon fixed an error in the user code, this change may lead to compilation errors.

COM-ATL Support (C++)

Property `ATL::Macro::ATLConnectionPointImpl`:

The value of this property (which was previously ignored) now affects the generated code for ATL classes.

If you have modified the value of this property in your model, this change will affect the generated code.

It is recommended that you review and verify the change.

CORBA Compatibility with Rhapsody 6.0

Applicable to RiC++ CORBA developers who are upgrading from Rhapsody 6.1, and use C++ attributes whose type is a CORBA element (type or class).

Improvements were made to the backward compatibility of Rhapsody code generation with Rhapsody 6.0.

As a result, there are a few changes in the way Rhapsody generates code for C++ attributes whose type is a CORBA element:

- The Constant and Reference checkboxes now affect the code as they do with regular C++ attributes. This means that if you have attributes whose type is a CORBA element and these options were selected, you will need to unselect them in order to get the same code as you got with Rhapsody 6.1.
- The property `CORBA::C++Mapping_<CORBAStereotype>::in` no longer affects the code generation for attributes. This means that if you need to specify the mapping of a CORBA language type to a C++ attribute, you must use the property `CORBA::Type::CPP_in` instead. (It is recommended that you specify the mapping at the C++ attribute and not on the CORBA type.)

Framework

RiC++

OXF

- OMReactive state getter and setter:

  `getState()` was renamed `getReactiveInternalState()`.

  `setState()` was renamed `setReactiveInternalState()`.

  If you have used the state attribute getter/setter (introduced in Rhapsody 6.0) in your application code, you should update your code.

  Note: The getter and setter signatures used in the statechart serialization auto-generated code were also updated.

- Animation:

  – Support for long double types:

  Applicable to custom adapters (probably using diab compiler, version 4.X or earlier).

  Explicit support for long double types was added under `#ifndef OM_NO_SPECIAL_SERIALIZE_LONG_DOUBLE`

  This support is required by some compilers but should be disabled under diab 4.X

  If you experience compilation errors related to ambiguous operator calls, you should add the #define in the adapter omosconfig.h

RiC

OXF and IDF

- Usage of the GEN macro without semicolon:

  The GEN macros (RiCGEN, CGEN, CGEN_BY_X, etc.) were modified so that the user can check whether the event was successfully sent or not.

  In order to achieve this, the macros were modified to a single statement and the wrapping block was removed.

  As a result, if one of these macros is used in your code without a terminating semicolon, you will get compilation errors and will need to add the semicolon(s).

Automatic Upgrade Performed by Rhapsody

Code Generation

Backward Compatibility Profiles

When loading pre-6.1 MR-1 models, Rhapsody automatically adds a compatibility profile to the model.

This profile provides property settings that maintain the pre-6.1 MR-1 behavior.

Statechart Serialization

Applicable to C++

The signatures used to access the OMReactive state attribute were updated in accordance with the framework changes.

Redundant Assignment of Event IDs Removed

Applicable to C

The assignment of the event ID inside the generated event constructor was removed since this is done in the initialization of the framework base class as well.

Declaring Empty Throw

Applicable to C++

There were changes made to the way Rhapsody interprets the property `CPP_CG::Operation::ThrowExceptions`.

Now, if the property contains whitespace, Rhapsody generates an empty `throw()` as part of the operation declaration, instead of ignoring the whitespace.

Changes Disabled for Backward Compatibility

Code Generation

General

Initialization of StaticArray Composite Relations

Applicable to C, C++ and Java

Improvements were made to the initialization code generated for parts with bounded multiplicity implemented as StaticArray (for example, C* itsC[5]).

These improvements avoid the redundant search for free locations in the array inside the composite create operation (e.g., newItsC()). This is done by passing the index to the create operation from the external loop in `initRelations()`.

Since this is a change to the create operation signature and behavior (e.g., newItsC() replaced by newItsC(int i)), the change is disabled when loading pre-6.1 MR-1 models. This is accomplished by setting the value of the property `CG::Relation::CreateComponentUsingIndex` to `False` in the automatically-loaded profile CGCompatibilityPre61M1<lang>.

Composite Qualified Relations

Applicable to C, C++ and Java

Code generation was modified to ignore the qualifier of a composite (black-diamond) relation.

While this change is not optimal, it prevents compilation errors. A warning is issued to ensure that the user is aware of this code generation behavior.

This behavior is disabled when loading pre-6.1 MR-1 models in order to support users that worked around the compilation errors by taking over part of the generated code.

This is accomplished by setting the value of the property `CG::Relation::IgnoreQualifierOnBlackDiamond` property to `False` in the automatically-loaded profile CGCompatibilityPre61M1<lang>.

Static Attribute Initialization Style

Applicable to C++

Initialization of static attributes is now affected by the property `CPP_CG::Attribute::InitializationStyle`.

When the property value is set to `ByInitializer`, parentheses are used instead of assignment (e.g., OMString A::className("A") instead of OMString A::className = "A").

Since the default property value is `ByInitializer` and static attributes were always initialized by assignment, the behavior is disabled by setting the property `CPP_CG::Attribute::EnableInitializationStyleForStaticAttributes` to `False` in the automatically-loaded profile CGCompatibilityPre61M1Cpp.

Generation of Package Initialization and Cleanup Operations

Applicable to C, C++ and Java

The generation of the package initialization and cleanup operations (e.g., the <package>_OMInitializer constructor and destructor) was improved to reduce the number of cases where empty operations are generated. In addition, you can prevent the generation of these operations altogether by setting the value of the properties `CG::Package::GeneratePackageInitialization/GeneratePackageCleanup` to `Never`.

To avoid compatibility issues, the pre-6.1 M1 behavior is maintained via the CGCompatibilityPre61M1<lang> profiles by setting these property values to Always. Note that if either of the properties is set to Always, Rhapsody will maintain the compatibility mode for both operations.

CORBA

Constant Attributes

Rhapsody 6.1 MR-1 supports generation of read-only attributes for attributes of CORBA interfaces whose constant modifier is set.

This behavior is disabled when loading pre-6.1 MR-1 models by setting the value of the property `CORBA::Attribute::ConstantAsReadOnly` to `False` in the automatically-loaded profile CGCompatibilityPre61M1Cpp.

Mapping of Events and Triggered Operations

Applicable to RiC++ CORBA developers who are upgrading from Rhapsody 6.0 or earlier.

Beginning with Rhapsody 6.1, event and triggered operation arguments whose type is a CORBA element are mapped to code using the property `CORBA::C++Mapping_<CORBAStereotype>::TriggerArgument` instead of the property `CORBA::C++Mapping_<CORBAStereotype>::in`

For backward compatibility, this is disabled by the automatically-loaded profile CGCompatibilityPre61Cpp.

Additional Information

Code Generation

Template Instantiation Usage

Applicable to C++

When template instantiation is used by another class (relation to a template instantiation, etc.), Rhapsody will generate the #include to the template instantiation in the specification file, instead of forward declaration, to ensure that the template declaration is available for the compiler.

This is done to avoid a situation where the template instance forward declaration results in a compilation error since the template declaration was unavailable.

Web Instrumentation

Applicable to C++

Missing 'static' qualifier was added to the `notifyWebRelationModified()` web instrumentation function.

Namespace Cleanup

Applicable to C++ and Java

Redundant namespace/package usage in generated code attributes and relations, accessors/mutators was removed. For example, when classes C and D are in the same namespace NS, `OMIterator<NS::C*> D::getCs() const` is replaced by `OMIterator<C*> D::getCs() const`.

Generation of Empty Packages

Applicable for COM and CORBA developers

Empty C++ package generation is prevented when the property `CG::Package::GeneratePackageCode` is set to `Smart`.

Activity Diagrams for Operations

Applicable to C++

Redundant C++ references (i.e. "&") to the operation arguments were removed (e.g., for operation f(int i), the functor argument is int i instead of int& i).

Framework

Adapters

POSIX Thread Creation Parameters

Applicable to RiC++ Linux adapter and RiC Linux and POSIX (RiCOSPosix) adapters.

`PTHREAD_CREATE_JOINABLE` was replaced by `PTHREAD_CREATE_DETACHED` in the p_thread creation parameters.

This was done because the adapters' implementation does not follow the join semantics on thread destruction.

POSIX Mutex Creation

Applicable to RiC++ Linux adapter

Setting of the mutex kind is done by calling the `pthread_mutexattr_settype()` function, instead of attempting to set the `pthread_mutexattr_t.__mutexkind` member directly.

INTEGRITY

- Build

  Applicable to C and C++

  A new optional switch (`-trg`) was added to IntegrityBuild.bat.

  This switch let you specify the target processor (e.g., PPC, ARM), for example:

  ```
  IntegrityBuild.bat c:\ghs\int505 rpx-cllf  c:\ghs\ppc407  -trg
  ppc_integrity.tgt
  ```

- Memory leaks in animation

  Applicable to C and C++

  Memory leaks due to sending of messages to Rhapsody were resolved.

- Memory leaks on task creation

  Applicable to C and C++

  The missing cleanup of the task name was added to `RiCOSTask_init()` (RiC) and the `IntegrityOSThread` constructor (RiC++).

Nucleus

Applicable to C++

`#define OM_NO_SPECIAL_SERIALIZE_LONG_DOUBLE` added to omosconfig.h

This was done to disable the explicit animation support in long double since diab 4.4 does not differentiate between double and long double.

pSOS

Applicable to C++

`#define OM_NO_SPECIAL_SERIALIZE_LONG_DOUBLE` added to omosconfig.h

This was done to disable the explicit animation support in long double since diab 4.2 does not differentiate between double and long double.

WinCE

Applicable to C++

Creation of nameless NTSemaphores is now supported.

Solaris

- Timer implementation:

  Applicable to C++

  `usleep()` was replaced with `nanosleep()` since usleep is not thread-safe.

  The registration on the SIGALRM moved from the do-while loop to `VoidSigAlrmHandler()` and additional registration was added before the do-while loop.

RiC++

OXF

- template <class Concept> class OMNullValue:

  New public static operation `void initNullBlock()` was added. This operation ensures that the initialization of the OMContainersNullBlock is defined under #ifdef UseNullBlockContainer. `initNullBlock()` is called from `get()` to ensure the expected behavior.

- Statically allocated Active-Reactive object behavior termination:

`OMReactive::setShouldDelete()` was modified to ensure that the thread part of the class is aware that the class should not be deleted.

This is done by calling `OMThread::setDeletionAllowed(false)`.

Note that the change assumes that OMThread is the base class of active-reactive classes. If you are replacing the CoreImplementation, you will need to modify this code.

- OMTimerManager destruction:

  The deletion of the timer manager tick-timer (OMOSTimer) was wrapped by lock() - unlock() to prevent potential race between the destruction of the timer manager and the tick callback.

- OMOSFactory:

  The name of the `createOMOSThread()` entry function function-pointer was changed from "entry" to "entryFunction" to prevent collision with an environment macro.

- OMStartBehaviorEvent:

  Redundant friend declaration to the pre-6.1 OMFriendStartBehaviorEvent class was removed.

RiC

IDF

- RTOS-specific conditional calls were removed from initialization.c, macros.h, ric.h and ricosnt.c
- Redundant semicolons were removed as a result of the code generation improvements.

RiJ

- Redundant #import statements of java.lang.Object and java.lang.String were removed.

MULTI Makefile Generator

INTEGRITY Target Selection

Applicable to C and C++

A PrimaryTarget property was added to control the INTEGRITY target (PPC, ARM, etc.)

The switch `-G` was added to the debug settings when using generating GPJ projects.

Resolution of dependencies between components was fixed.

Enumeration of changes:

- New key was added to the Keys table for -G (key 22).
- Constant key was added for the PrimaryTarget value (key 17).
- `getDependentComponents()`: the provided component (aComponent) is used as the context of the call to `getConfigByDependency()` instead of activeComponent.
- InitKeys():

  The initialization of the PrimaryTarget key was modified based on the property.

  Initialization of key 22 was added.
- `AddExeCompileProperties()`, `AddLibCompileProperties()`: key 22 is added in debug mode if the value is not empty.

Properties

Modified Properties

- The -check flag was removed from the BLDAdditionalOptions, BLDMainExecutableOptions and BLDMainLibraryOptions properties of the following RiC and RiC++ environments: Multi4Win32, INTEGRITY5, INTEGRITY5ESTL.
- CORBA::TAO::IDLCompileCommand was modified to support generation of directory per package.
- RiCContainers: Type cast in the IterCreate properties now uses a new property called CastRT that holds the cast operator. This change allows code generation to omit the cast when it is not required.

Renamed Properties

Spelling error in CORBA::Package::DeclareInterfracesInModule was corrected. The property is now called DeclareInterfacesInModule.

Upgrading to Version 6.1

Changes that Require User Action

Code Generation

Attribute Multiplicity field

Applicable to: CORBA Models

The Attribute Multiplicity field that was ignored for CORBA in pre-6.1 models now affects the code.

If your model contains CORBA attribute where the Multiplicity field value is other than 1, you will experience changes in the code.

Framework

VxWorks Adapters

Applicable to: C/C++ VxWorks users

A defect in the RTOS message queue adapter that automatically set the priority of an ISR message to `MSG_PRI_URGENT` when calling `msgQSend()` was fixed and by default the priority is now `MSG_PRI_NORMAL`.

To maintain the pre-6.1 behavior, compile the framework with the `OM_VX_URGENT_PRIORITY_FOR_ISR` flag.

RiC++ Framework File Changes

Library	File	Status	Reason
AOM	AnimServices,cpp/h	Added	Animation decoupling from The OXF CoreImplementation
AOM	OMFriendStartBehaviorEvent.cpp/h	Removed	Animation decoupling from The OXF CoreImplementation
AOM	OMFriendTimeout.cpp/h	Removed	Animation decoupling from The OXF CoreImplementation
AOM	OMTime.cpp/h	Removed	Animation decoupling from The OXF CoreImplementation
AOM	OXFInstrumentation.cpp/h	Removed	Animation decoupling from The OXF CoreImplementation
OXF	OXFCogeGen50.h renamed to OXFCodeGen50.h	Renamed	Fix spelling
OXF	IOxfAnimReactive.h	Added	Animation decoupling from The OXF CoreImplementation
OXF	IOxfAnimHelper.h	Added	Animation decoupling from The OXF CoreImplementation
OXF	IOxfAnimThreadManager.h	Added	Animation decoupling from The OXF CoreImplementation
OXF	IOxfAnimTimerManager.h	Added	Animation decoupling from The OXF CoreImplementation
OXF	OMAnimReactive.cpp/h	Added	Animation decoupling from The OXF CoreImplementation
OXF	OMAnimHelper.cpp/h	Added	Animation decoupling from The OXF CoreImplementation

IDF (Interrupt Driven Framework)

Applicable to: RiC IDF users

The RiC IDF was integrated into Rhapsody and is part of the Rhapsody distribution.

This version of Rhapsody only contains an adapter for the Microsoft environment. Also, the IDF is slightly different from the previous IDF versions.

The most significant difference is the adapter definition scheme—the adapter is now defined as a separate model.

To upgrade an adapter or application, carry out the steps described in **Preliminary Steps**, and then carry out the steps described in **Upgrading Adapters** or **Upgrading Applications**.

Preliminary Steps

1. Open your IDF_<target>_compiler.prp file, and make the following changes:

   a. Add the empty property MakeFileName.

   b. Change the property CppCompileSwitches as follows:

      Replace $(CIDF_ROOT)/oxf with $(OMROOT)/LangC/idf

      Replace $(CIDF_ROOT)/<adapters> with LangC/idf/Adapters/<target name>

   c. Change the property MakeFileContent as follows:

      For the variable OXF_LIBS, replace
      OXF_LIBS=$(CIDF_ROOT)\<target name>\oxf\idf$(LIB_EXT)
      with
      OXF_LIBS=$(OMROOT)\LangC\lib\<library prefix>idf$(LIB_EXT)

Upgrading Adapters

If you have an IDF adapter and you wish to upgrade to the new version of the IDF, you should follow the following procedure:

1. Open the existing adapter model.
2. Add, by reference, the profile IDFProfile.sbs to the model. If it was already referenced, remove it and add it again.
3. In the Configuration dialog, select the IDF stereotype.
4. Change the directory so that the code is generated in $(OMROOT)/LangC/idf
5. Create the OSAL package and drag the following functions, types, and variables to the package:

   - `RiCInitTimer`—sets up a periodic interrupt that calls the RiCTick operation every `RiC_MS_PER_TICK`.
   - `RiCExitCriticalRegion`—enables interrupts.
   - `RicEnterCriticalRegion`—masks interrupts.
   - `RiCGetSystemTick`—returns system tick size.
   - `RiCSleep`—operation is called when there are no events to handle and sleep can be used until the next timeout or when an interrupt occurs.
   - `RIC_MEMORY_ALLOCATION`—sets up the buffers used for the memory allocation.
   - `RIC_MAX_EVENTS`—maximum number of simultaneous events.

- `RIC_MAX_TIMEOUTS`—maximum number of simultaneous timeouts.
- `RIC_MS_PER_TICK`—periodic timeout in milliseconds.
- `RiCTick`—this is the operation that will be called from the periodic interrupt ISR routine
- `RiCTickThread`—the root of the timer thread that really just sets the bTick flag (this is only really needed for running on VxWorks or Windows).
- `tRiCCriticalSection`—OS-specific type, which is used during critical section processing.

6. Remove all elements from the Component scope.
7. Define the following two component files:
   a. RICOS.h, containing OSAL specification. Its path should point to the Adapters/<target> directory.
   b. <target>OS.c, containing the OSAL implementation. It should be generated in the idf directory, so its name should be unique.
8. Go to the Configuration properties page, and make the following changes:
   a. For the property <target environment>.CppCompileSwitches, replace $(OMROOT)/LangC/idf with the current directory path (.)
   b. In <target environment>.MakeFileContent:

      Include idfFiles.list

      Add <target>OS$(OBJ_EXT) to the list of objects.

      Change the output library location, for example, /out:../lib/msidf.lib
   c. Set MakeFilename to <target prefix>idf. This affects the name of the generated makefile.

Upgrading Applications

1. Open the model.
2. If it contains a reference to another IDFProfile, delete the reference, and add, by reference, the new IDF profile from the directory $(OMROOT)/Share/Profiles.
3. Set the Configuration stereotype to IDF.
4. Generate and make.

Callback Function Pointers in the RiC++ OSAL

Applicable to RiC++ users with custom adapters

The OSAL function pointers declaration was aligned with ANSI.

As a result, some compilers require you to fix the declaration in the adapters as well.

The changed operations:

```
OMOSFactory:

virtual OMOSThread* createOMOSThread(void (*entry)(void*),
void * param, const char * const threadName = 0,
const long stackSize = OMOSThread::DefaultStackSize)=0;

virtual OMOSTimer* createOMOSTickTimer(OxfTimeUnit time, void
(*callback)(void*), void * param)=0;

virtual OMOSTimer* createOMOSIdleTimer(void (*callback)(void*), void *
param)=0;

OMOSConnectionPort:

virtual void SetDispatcher(void (*dispfunc)(OMSData*))=0;
```

Properties on Stereotypes

Applicable to users with stereotypes on components or configurations.

Properties specified on Component and Configuration stereotypes are now visible outside the component hierarchy for the active component and configuration.

This means that an overridden value on the active component or configuration stereotypes will affect classes, etc.

In particular, such overridden properties will affect code generation and may create unwanted modifications in the code.

If you have such stereotypes, review the overridden properties and verify that this is what you expect.

Automatic Upgrade Performed by Rhapsody

Code Generation

General

Applicable to: C, C++ and Java

Parentheses were added around expressions that have more than one operator to ascertain the computation order.

The change focus was RiC in order to comply with additional MISRA-C rules and some of the changes affect C++ and Java as well.

RiC

A `CGCompatibilityPre61C` profile is loaded for pre-6.1 models

The profile sets the new `Cast` and `IterReturnType` properties of the `RiCContainers` subject in order to provide backward compatibility in the signature of the *get* helpers.

Removing the profile will prevent generation of redundant cast in the get operations and is required to generate MISRA-C 98 compatible getters.

RiC++

General

A `CGCompatibilityPre61Cpp` profile is loaded for pre-6.1 models

- The profile sets the new `IterReturnType` properties of the `OMContainers` subjects in order to provide backward compatibility in the signature of the *get* helpers.
- The profile sets the `CORBA::Configuration::Pre61C++TypeNamesResolution` property to true and prevents the new properties resolution scheme described in **Property Resolution**

Animation and Tracing Configurations

Additional `#include <aom/aom.h>` statements are generated to each header file in instrumented mode in order to support the decoupling of the animation and the OXF.

CORBA – Type Modeling Support

Rhapsody 6.1 supports modeling of structure, enumeration and typedef types in the CORBA domain.

Union type modeling is not supported by code generation.

As part of the support, the ability to reference CORBA types and interfaces was improved, providing a mechanism similar to the referencing done between C++ elements

- New property `C++Implementation` added under `CORBA:Class` and `CORBA:Type`
  - The property defaults are set on the CORBA interface/type and can be overridden locally at the C++ referencing element (as done for C++ In/Out properties)
  - The property provides the Reference, Variable or Fixed mapping
- When referencing a CORBA interface or an enumeration/structure/typedef type Rhapsody automatically selects the appropriate `C++Mapping_<TYPE>` metaclass to be used for the referencing implementation
- The `CPP_in`, `CPP_inout`, `CPP_inout` and `CPP_return_value` properties (under `CORBA:Type`) were removed since they are no longer needed, however overridden values of these properties are taken into account in code generation
- `IDLSequence` was added under `CORBA:Type` to support automatic sequence generation for types (as done for interfaces)
- The values of the properties under the `C++Mapping_<TYPE>` metaclass were modified to take advantage of the automatic mapping provided using the `C++Implementation` property
- New mapping metaclasses were added:
  - `C++Mapping_CORBAFixedSequence` : provides mapping to a fixed sequence declaration
  - `C++Mapping_CORBAInterfaceVariable`: provides mapping to an interface used as a variable type. Also replaces the obsolete `C++Mapping_CORBAObjectReference` metaclass
- The `C++Mapping_CORBAObjectReference` metaclass was removed from the factory, it is defined in the `CGCompatibilityPre61Cpp` profile for backward compatibility
- The `CORBAStereotype` enumeration literal was updated to reflect the changes listed above
- Properties in the CORBA containers definitions (`RelationTargetType, CType, IterType and FullTypeDefinition`) were modified to support the automatic implementation and to support CORBA typedef generation
- Redundant namespace in referencing elements was removed

Features Disabled for Backward-Compatibility

Property Resolution

Applicable to: CORBA models

The search for overridden `CORBA::C++Mapping_CORBA<TYPE>` properties has been aligned with the search of other type referencing properties (such as `CG:Class:In`). This change let you override the values specified at the CORBA level in the C++ referencing element (Argument, Attribute, etc.).

This change is disabled by default for CORBA language types by the `CGCompatibilityPre61Cpp` profile (see General)

Code Generation

Generation of Dependencies from Arguments

Rhapsody's ability to generate dependencies (i.e. include/import statements) from arguments that exist in the model (type or class that were selected using the drop-down list) has been significantly enhanced.

You can select the dependency type by setting the argument property `CG::Argument::UsageType`.

For backward compatibility, the property is set to `None` by the code generation compatibility profiles.

Origin of #include

Applicable to C/C++

Rhapsody cannot generate a comment before an include statement, specifying the reason the include was generated.

This feature is disabled by the *GCompatibilityPre61* C/C++ profiles by setting the `<lang>_CG:Dependency:GenerateOriginComment` property to `False`.

CORBA

The `CORBA:Operation:C++DefaultThrow` property lets you specify a default throw statement for C++ operations that realize CORBA operations.

For backward compatibility, the *CGCompatibilityPre61Cpp* profile sets the property to an empty string.

Additional Changes

Framework

RiC++

Decoupling of animation from the OXF CoreImplementation

The animation libraries and the OXF coupling was reduced and formalized in order to enable users with customized implementation of the OXF Core (IOxfActive, IOxfReactive, etc.) to provide animation services based on the customized implementation.

This wide change affects many of the framework files, as well as adding and removing files as described in RiC++ Framework File Changes.

The OXF model includes the description of the AnimServices API (under the aom external package) as well as sequences of the main scenarios of the OXF-AOM integration (under the AnimAPI package)

The change included:

- OXF:
  - New packages AnimAPI and AnimImplementation were added under Design::oxf::Anim (Anim is a new package itself). These packages define an interfaces and concrete implementation of services that are required by the AOM.
  - Additional component: oxfAnimFiles was added to the model. This component defined by its scope the services that the AOM requires from the OXF. The scope includes the CoreAPI, AnimAPI and parts of the Services package. It does not include the CoreImplementation and the AnimImplementation packages.
  - `OMThreadManager` now inherits from `IOxfAnimThreadManager` and implements the Anim API
  - `OMTimerManager` now inherits from `IOxfAnimTimerManager` and implements the Anim API

  - `CoreImplementation` operations and members that existed only for AOM support were removed

- OMReactive:
  - `static bool isValid(const IOxfReactive* const)` (also removed: `isValidOMReactive()` definition from OMObsolete.h)
  - void registerWithOMReactive(const void *, AOMInstance*)
- OMThread:
  - `AOMEventQueue* getAOMEventQueue() const`
  - `AOMStepper* getStepper() const`
  - `void notifyTimeoutCanceled(IOxfTimeout*)`
  - `void notifyTimeoutSet(IOxfTimeout*)`
  - `AOMThread* getAOMThread() const`
  - `AOMThread* aomthread`
  - `OMThread` event queue implementation was simplified and is no longer dependent on the AOM in the instrumented versions of the framework
  - `getEventQueue() return type` changed from `OMEventQueue*` to `const OMEventQueue*`
  - Additional notifications maintain the synchronization of the AOM event queue representation
  - The AOM no longer uses the `OMOSMessageQueue::getMessageList()` API.
  - New framework event id `OMAnimWakeupEventId` was added, this id is used to identify animation wakeup events that are used to wakeup blocking threads. The id is used in `OMThread::execute()` instrumentation.
  - All the references to the AOM classes/services were replaced with references to AnimServices. All the references are wrapped with `#ifdef _OMINSTRUMENT`.
- Exception to the rule are
  - Passing `AOMInstance` and `AOMSState` as pointers without access to their definition.
  - Usage on `OMSData` in the OSAL adapters
  - Include to AOM were replaced by include to AnimServices
  - Instrumentation macros and direct calls to AOM services were replaced with calls to `AnimServices` operations
  - Additional instrumentation code was added to support the thin interface between the AOM and the OXF (wrapped with `#ifdef _OMINSTRUMENT`).
  - `OMTMMessageQueue` implementation was changes to support by-value allocation prior to the creation of the underlying `OMOSMessageQueue`

  - Default constructor was added, the constructor maintains the RTOS queue association empty
  - `init` and `cleanup` operations were that handles the initialization and cleanup of the RTOS queue
  - `isEmpty()` and `isFull()` are now const operations
  - `cleanupRelations()` was removed
  - `OMEventQueue` no longer inherits from `OMTMMessageQueue`. The class implements the same API. `OMTMMessageQueue` is maintained for backward compatibility.
  - `ommemorymanager.h` : the include to `OMMemoryManagerSwitchHelper.h` is now protected with `#if (!defined(OM_NO_FRAMEWORK_MEMORY_MANAGER) && !defined(OMOMATE))` in order to minimize the scope of the `oxfAnimFiles` component
  - `OMOSThread`: A new public virtual operation `void resetWrapperThreadOsHandle(void*)`
  - The operation resets the thread OS handle.
  - This operation should be used with care and only for wrapper threads.
  - The operation has an empty implementation by default, and is implemented in the VxWorks adapter.
  - The operation is called in instrumented mode by OXF::initialize() on the OS thread of the OMMainThread::instance() singleton, to ensure that the OS thread ID expected by the animation is set.

- `AOM`:
  - Replacing of references to the OXF CoreImplementation package (`IOxfReactive`, `OMEvent`, etc.) with referenced to the CoreAPI classes and to the new AnimAPI packages.
  - Defining a utility class `AnimServices` that is used as the *gateway* of the OXF to the animation.

IOxfReactive

- New protected operation `bool restartBehaviorEnabled() const` was added.

  The operation checks whether a second call to `startBehavior()` should cause a restart, i.e., restart of the statechart.

- `startBehavior()` `If restartBehaviorEnabled() returns true, ignores isBehaviorStarted() and restarts the statechart.`
- Two new public boolean attributes were added: `supportRestartBehavior` and `globalSupportRestartBehavior` (static).

For each, the default value is false.

For a given reactive instance, if any of these attributes is true, `restartBehaviorEnabled()` returns true.

`globalSupportRestartBehavior` is static and, therefore, enables restart for the entire system.

`supportRestartBehavior` is instance-specific.

Both attributes are implemented using private data members and public getters/setters.

OMStack

- The empty destructor was removed.
- An ability to disable the definition of the `rcsid` and `hrcsid` variables was added.

  You can disable the definition by compiling the framework with the `OM_NO_RCS_ID` compiler flag (e.g., `-DOM_NO_RCS_ID`). This flag is used in the VxWorks adapter to avoid a compilation warning (unused global variables).
- The Adapters implementation was adjusted to the OSAL signatures (see **Callback Function Pointers in the RiC++ OSAL**).

Linux

`LinuxOSFactory` creation was modified, and the friend declaration to `OMOSFactory` was removed.

Additional compilation flags

- `OM_FORCE_IOSTREAM`: Forces usage of iostream by the framework (can be overridden by the flags below)
- `ANIM_USE_IOSTREAM`: Adding `ANIM_USE_IOSTREAM` as a compilation flag (e.g. `-DANIM_USE_IOSTREAM`) to the animated makefiles and application will force animation to be compiled with iostream support.
- `ANIM_USE_STDIO`: Adding `ANIM_USE_STDIO` as a compilation flag (e.g. `-DANIM_USE_STDIO`) to the animated makefiles and application will force animation to be compiled without iostream support.
- `OM_ENABLE_STRING_SERIALIZATION`: Enables `OMString` iostream serialization operators
- `OM_NOTIFY_USE_IOSTREAM`: Forces `OMNotifier` notify operations to use iostream
- `OM_NOTIFY_USE_STDIO`: Forces `OMNotifier` notify operations to use stdio
- `OM_NOTIFY_SILENT`: Disables `OMNotifier` notify operations

Other changes

- `OMThread::dispatch()`: added call `OMEvent::Delete()` instead of `destroy()` in API compatibility mode
- `OXF` (the class): added initialization of the static associations
- timer.h: `typedef OMTimerManager OMThreadTimer` added for backward compatibility
- omiotypes.h: add include to `<iosfwd>` when `OM_STL` is defined
- `OPORT_AT()` macro definition was fixed

Adapters

GHS MULTI (WIN32/INTEGRITY) Compilation

Applicable to: Ada, C and C++

MULTI build scripts were modified to support both MULTI 3.5 `bld` format and MULTI 4.0 `gpj` format. As a result, the batch files invocation command used to build the framework libraries was modified.

Invoke the batch file without parameters to get a usage message.

WRS VxWorks Compilation

Applicable to: C/C++ users

The VxWorks framework makefiles were modified to support VxWorks 5.5 (Tornado 2.2) and VxWorks 6.0 (Workbench 2.2).

The VxWorks 6.0 support also includes in-house support for both `diab` and `gnu` tool families.

By default, build of all the framework libraries continue to build the VxWorks 5.5 adapters. To build the VxWorks 6.0 adapter add `VX_VER=6.0` and `TOOL=diab` to the make invocation command.

When building each library by itself the default makefile settings are VxWorks 6.0 with `diab` tool family.

Win32

Cygwin support was added to the adapter source files.

The support includes the ability to use `CreateThread()` instead of `_beginthreadex()` based on the `__USE_CREATE_THREAD__` compilation flag.

Nucleus

Applicable to: C++

`#define OM_NO_TYPENAME_SUPPORT` was added to the `omosconfig.h` and fix compiler warnings.

RiC

RiCReactive

Two new operations were added that provides access to the reactive internal state.

The operations were added as part of the statechart serialization support.

```
long RiCReactive_getState(const RiCReactive* const me)
void RiCReactive_setState(RiCReactive* const me, long oxfState)
```

RiCTimerManager and RiCHeap

The initialization (`RiCTimerManager_init()` and `RiCHeap_Init()`) was improved to prevent initialization errors.

RiCTypes:h

typedef of short was added to RhpShort**.**

RiCReactive.h

`RiCGui` was defined in non-instrumented (animation/tracing) mode in order to enable the usage of the `GEN_BY_GUI()` macro in non-instrumented applications.

An ability to disable the definition of the `rcsid` and `hrcsid` variables was added. You can disable the definition by compiling the framework with the `OM_NO_RCS_ID` compiler flag (e.g. `-DOM_NO_RCS_ID`).

INTEGRITY adapter

The `RiCOSEventFlag_wait()`, `RiCOSMutex_lock()` and `RiCOSMutex_free()` operations were modified to return 0/1 for fail/success instead of the RTOS return value.

Code Generation

C++

Exceptions *throw* declaration

The `ThrowException` property now affects constructors and destructors as well as regular operations.

C++ CORBA Implementation

The code for the accessors of Associations which are realizing CORBA Association getters has changed: the length is now stored in a local variable instead of getting it twice.Wrong setting of the association end was removed.

For example:

```
void B::setItsIA(IA_ptr p_IA) {
itsIA = IA::_duplicate(p_IA);
itsIA = p_IA;
}
```

Was replaced by:

```
void B::setItsIA(IA_ptr p_IA) {
itsIA = IA::_duplicate(p_IA);
}
```

Types with Identical Names

There has been a change in the way Rhapsody handles situations where there are types with identical names in user packages and in Rhapsody's PredefinedTypes package.

Now, when Rhapsody searches for types, it first looks in user packages and only afterwards in Rhapsody's PredefinedTypes package. Therefore, in cases where the same type name exists in both a user package and the PredefinedTypes package, the generated code will now contain an additional include statement for the relevant user package.

Statechart Code

Applicable to C, C++ and Java

Previously there were instances where Rhapsody generated code that called the macro OMSETPARAMS even though it was redundant. These cases have been eliminated.

This change may result in other changes in your code. Since the removal of this unnecessary line reduces the size of certain functions, these functions may now be lower than the defined "inlining" threshold. Where this occurs, you will see that these functions no longer appear in the code, and calls to these functions are replaced by use of the body of the relevant function.

Changed Properties

Property	Change	Reason
General:Model:Extension	Removed	Unused
CG:Attribute:IsConst	Changed from boolean to enumeration	Adds an option to make the return type const as well as the getter itself
CG:Operation:Generate	Changed from boolean to enumeration	To support generation of the operation specification without the implementation, and vise versa.
CG:Relation:IsConst	Changed from boolean to enumeration	Adds an option to make the return type const as well as the relation getters
<Containers>:<Implementation>: IterReturnType	Added in the C, C++ and Java container subjects	Lets the user specify the getters return type, when empty the RelationTargetType property is used
CPP_CG:Configuraion:Environment	New environments were added	Cygwin, MULTI 4.0 - Win32, MULTI 4.0 – INTEGRITY 5.0, MULTI 4.0 – INTEGRITY 5.0 ESTL, VxWorks 6.0 – diab, VxWorks 6.0 - gnu
GCC environments (C/C++) MakeFileContent property	Added comment that explains the multiple appearance of the instrumentation libraries in the link command	Documentation
CPP_CG:IntegrityESTL:BLDAdditionalOptions, EnvironmentVarName	Replaced $INTEGRITY_ROOT with $MULTI_ROOT	Align with MULTI 4.0 separation between MULTI and INTEGRITY
<TYPE>Containers:EmbeddedScalar:Get TYPE=STL, OM, OMU	The cast is removed	Takes advantage of the IterReturnType property and the $constRT keyword to add the cast only when required (see IsConst changes)
<TYPE>Containers:StaticArray:IterTest TYPE = OMU, OMCpp2Corba	Add parenthesis around the < test	Related to the parenthesis addition that was made for MISRA compliance
CORBA:Configuration:ORB	Orbix3.0.1 replaced by TAO	Change of the in-house supported ORB
CORBA:Class:InstanceNameInConstructor	Changed from True to False	TAO settings

CORBA:Type:CPP_*	Removed	See CORBA – Type Modeling Support
CORBA:Type:CORBAStereotype	Updated	See CORBA – Type Modeling Support
CORBA:C++Modeling_<TYPE>:*	Updated	See CORBA – Type Modeling Support
CORBA:Orbix3.0.1	The metaclass was removed	The ORB is no longer supported
CORBA:TAO	The metaclass was added	New supported ORB
<TYPE>Containers:<Implementation>: RelationTargetType/ CType/IterType/ FullTypeDefinition TYPE = OMCpp2Corba, OMCppOfCorba, OMCorba2Corba	Value changed	See CORBA – Type Modeling Support
C_CG:Configuraion:Environment	New environments were added	Cygwin, MULTI 4.0 – Win32, MULTI 4.0 – INTEGRITY 5.0, VxWorks 6.0 – diab, VxWorks 6.0 – gnu
RiCContainers properties	Parenthesis were added to conform with the MISRA standard	MISRA-C support
RiCContainers:StaticArray: RemoveAll	The value was set to set the array pointers to NULL.	Fix cleanup.

COM API

Additional capabilities were added to the API, see the COM API documentation.

MultiMakefileGenerator

Added features

- Both MULTI 3.5 (bld) and MULTI 4.0 (gpj) project files are now supported. This support affects most of the MultiMakefileGenerator operations
  - Hard-coded MULTI keywords are replaced by a reference to a table named Keys
  - The table is initialized in the Main() by calling InitKeys()

- Print of version information at the beginning of the generation
- Project file is not generated for components with «ExternalProgram» or «ExternalLibrary» stereotypes
  - Done by calling a new function `IsExternalComponent()` in the `Main()`
- Common keywords (see below) are now supported in various paths (component path, libraries, etc.)
  - The support is done by calling `ReplaceCommonStrings()` on path strings
  - The keyword format is `$<keyword>` or `$(<keyword>)`
  - The keywords are case sensitive
  - The keywords are
  - `projectPath, projectpath`: the model directory
  - `configPath`: the code generation directory
  - `INTEGRITY_ROOT, INTEGRITYROOT`: the INTEGRITY root directory
  - The operation also makes sure that the path separators are legal by replacing slash to backslash if needed
- Copy of the INTEGRITY.ld file was added by a new operation WriteLDFile() that is called by the Main(). When using INTEGRITY 5.0, the file is copied as INTEGRITY5.ld
  - The copy is done only once, so you can modify the file after it was copied in order to customize it for the specific application
  - The file is copied from `<Rhapsody>/Share/MakeTmpl` or from the INTEGRITY root directory. This lets you create a general customization for Rhapsody-generated applications without affecting the default supplied by GHS
- `AddMainBLDFile()`: support was added for
  - Adding the kernel project to the application. The kernel project is taken from the `<lang>_CG:<environment>:KernelProject` property
  - Adding related components to the build dependencies (link and search path) based on «Usage» dependencies. This is done by calling a new operation `getDependentComponents()`.

Additional Changes

- A new utility operation `addToCollection()` adds an element to a collection if the element is not already inserted.

Upgrading to Version 6.0 MR-2

Changes in Rhapsody 6.0 MR-2

Framework

RiC++

OMEvent

- A new public operation `bool isDeleteAfterConsume()` was added.
  - The operation existed in the pre-6.0 version of the OXF and was reintroduced for backward compatibility.
  - This is an inline operation that calls and returns the value from `shouldDeleteAfterConsume()`.

Constructor initializer

- The order of the attributes in the initializer was modified to be aligned with the order of declaration.
- This was done in the constructors of `OMIterator`, `OMQueue`, and `OMTimeout`.

oxf.h

A forward declaration of class `OMThread` was added for backward compatibility with pre-6.0 applications.

oxfFiles.list, oxfFiles_dll.list

The oxf object dependency line was updated.

Upgrading to Version 6.0 MR-1

Changes in Rhapsody 6.0 MR-1

CORBA

The mapping of the `CORBA::string` (defined in the `CORBA` package under `<Rhapsody>/Share/Properties`) to **C++ out argument** was modified from `char*&` to `CORBA::String_out` in order to align with the CORBA C++ mapping specification.

C++ OXF

- `IOxfReactive::OMTakeEventCompleted` and `IOxfReactive::OMTakeEventCompletedEventNotConsumed` visibility was changed to `public` (as in Rhapsody 5.2) for backward compatibility (the constants are kept in the framework for backward compatibility).
- `OMString` << and >> operators that were removed for the modeling of the OXF in Rhapsody 6.0 were re-introduced.
- The private attribute `OXFRefManager::totalReferences` was renamed to fix a spelling error.

Upgrading to Rhapsody 6.0

The changes in version 6.0 of Rhapsody are listed below.

For information on changes to Rhapsody in Ada, see the `RiA_Changes.pdf` document under `<Rhapsody>\Sodius\help`.

Changes that Require User Action

This subsection documents the changes that require you to perform some actions when you upgrade to Rhapsody 6.0.

Framework

See **Rhapsody in C++ Object eXecution Framework** for C++ framework-specific information.

Rhapsody in C

A new object, `AOMMessageSender`, was added to `<Rhapsody>/Share/LangC/aom` (`AomMessageSender.c/h`). This object is used to send all animation messages via a single task (activated based on an environment variable).

If you have a custom adapter that does not take advantage of the `aomFiles.list` file, you should add the new files to the aom makefile. For an example of how to use the `aomFiles.list`, see one of the adapters supplied on the Rhapsody distribution kit.

Rhapsody in C++

Applicability: Users with custom adapters that do not take advantage of the source files lists (e.g. `aomFiles.list`). These users must update the makefiles to reflect the changes.

The following files have changed:

- `oxf`

  See **Rhapsody in C++ Object eXecution Framework**.
- `aom`

The following files were added, which contain instrumentation classes and functionality that moved from the oxf:

- `OMAnimCommandLineParser.cpp/h`
- `OMAnimResourceGuardNotifier.cpp/h`
- `OMFriendStartBehaviorEvent.cpp/h`
- `OMFriendTimeout.cpp/h`
- `OMTime.cpp/h`
- `OXFInstrumentation.cpp/h`

- `omcom`

  The following empty source files were removed:
  - `RiCppAnimMessageTranslator.cpp`
  - `RiCppAnimMessages.cpp`
- `WebComponents`

  The following empty source file was removed:
  - `CppWebAdaptersPkg.cpp`

VxWorks Adapter

Applicability: Rhapsody in C and C++

The mutex creation flags of the OS binary semaphore were modified from (`SEM_Q_FIFO`) to (`SEM_Q_PRIORITY | SEM_INVERSION_SAFE`) in order to support priority inversion.

This change might affect the behavior of the application when several tasks block on the same mutex. You should validate that your application behavior is still as expected.

DiffMerge of Diagrams

As a result of the change in the graphic editor infrastructure, you should upgrade your models to the Rhapsody 6.0 repository (by loading and saving the model with Rhapsody 6.0) before using DiffMerge 6.0 to compare the diagrams.

Comparing diagrams that are stored in the pre-6.0 repository will show differences between equal elements.

Code Generation

Class Specification Epilog and Namespace

Applicability: C++

The generation of the `CPP_CG::Class::SpecificationEpilog` property moved to the class namespace in order to comply with the prolog position.

If you are using the specification epilog and assumed that its content is generated outside the class namespace, you should modify the epilog content.

For example:

```
namespace PP {
    //## class A
    <prolog>class A {
        ...
    };
    <6.0 epilog>
}
<5.2 epilog>
```

Properties

PredefinedMacros

`OM_DECLARE_COMPOSITE_OFFSET` was added to the `CPP_Roundtrip::General::PredefinedMacros` property. If you have overridden this property, you should add the value to your override.

DescriptionEditorSupportsRTF

The `EditorSupportsRTF` property (under `General::Model`) was renamed to `DescriptionEditorSupportsRTF`. If you defined this property in your `site.prp` file, you must manually rename it.

COM API

Applicability: Users that access statechart or activity diagram information via the COM API

As part of the renaming of the connector between a state and a substatechart or activity from a stub connector to an entry or exit point, the COM API metaclass name was modified from `Stub` to `EnterExit`.

Clients that used the metaclass information to identify entry and exit points should modify their code to use the new metaclass name in order for the COM API to continue working.

Rhapsody in C++ Object eXecution Framework

The C++ OXF has changed significantly in Rhapsody 6.0—it is now developed using Rhapsody itself. The change also includes adjustments in the instrumentation libraries. The framework model is available as part of the installation under `<Rhapsody>/Share/LangCpp/oxf/model`.

For more information, see the *Release Notes*.

Backward Compatibility

A special effort has been made to ensure full backward compatibility of the new version of the framework with the Rhapsody 5.*x* framework. This means that the new framework features are disabled by when loading pre-6.0 models.

The backward compatibility provides:

- **Full code generation backward compatibility**—When loading a pre-6.0 model and generating code, the same code is generated to interface with the framework.
- **Framework customization support**—Any customization of the framework (made by inheriting from the framework classes and overriding virtual operations) continues to work.
- **Adapters**—If you used the OXF source files list (`oxfFiles.list`) in your OXF makefile, there is no need to update the adapter makefiles.
- **Mechanistic optimizations**—Full backward compatibility is provided so the framework works in the same manner as it did in Rhapsody 5.x.

Changes that Require User Action

Usage of the typename Keyword in Templates

The VC++.NET 2003 and GCC 3.2 compilers require that access to a type of a template parameter is prefixed by the `typename` keyword (ANSI C++). For example:

```
template<class T> class X {
   typename T::Y;    // treat Y as a type
   Y m_y;
};
```

To support these compilers and continue supporting compilers that do not support the `typename` keyword, the `omtypename` macro (added in Version 5.2) is used in the `OMSelfLinkedMemoryAllocator` template. The declaration is as follows:

```
template <class T, int INITNUM> class RP_FRAMEWORK_DLL
    OMSelfLinkedMemoryAllocator : public
    IOxfMemoryAllocator {
        public :
        //#[ ignore

        typedef omtypename T* (*AllocationCallback)(int);
        //#]
        ...
        };
```

If your compiler issues warnings about the usage of `typename` in the template, add `#define OM_NO_TYPENAME_SUPPORT` to your `omosconfig.h` file (under `Share/LangCpp/osconfig/<RTOS>`) to make `omtypename` be an empty `#define`.

Reusable Statechart Implementation

Applicability: Reusable statechart implementation in conjunction with:

- Custom adapters
- Projects where the `CPP_CG::<Environment>::MakeFileContent` property is overridden

The `#define _OMFLAT_IMPLEMENTATION 1` statement was removed from the generated code. It was replaced by a compilation switch that is added to the generated makefile of reusable statechart configurations.

The switch is added based on the `CPP_CG::<Environment>::ReusableStatechartSwitches` property and the `$OMReusableStatechartSwitches` keyword that was added to the `MakeFileContent` property. If you have a custom adapter or overrode the `MakeFileContent` property, you must change these values so reusable statechart configurations will compile.

For example:

```
Property ReusableStatechartSwitches String "-
DOM_REUSABLE_STATECHART_IMPLEMENTATION"

[MakeFileContent]

ConfigurationCPPCompileSwitches=
    $OMReusableStatechartSwitches
    $OMConfigurationCPPCompileSwitches
```

Compilation of the Framework with Custom Adapters

Applicability: Custom adapters that do not use the `oxfFiles.list` file to obtain the framework source files

The list of framework generic files was modified in Rhapsody 6.0. If you are not using the `oxfFiles.list`, you should update your OXF makefile based on the list content for your application to link properly.

Enabling the New Features of the Framework

This subsection describes how feature are disabled when loading pre-6.0 models and how to enable them.

Moving to the 6.0 Framework API

The 6.0 framework introduces a set of interfaces for the core behavioral framework. The interfaces define a concise API for the framework and enable you to replace the actual implementation of these interfaces while maintaining the framework behavior.

As a result of the interfaces' introduction, the framework behavioral classes (`IOxfReactive`, `OMThread`, and `OMEvent`) use a new set of virtual operations to implement the interfaces and provide the behavioral infrastructure.

To support existing customizations of these classes (made by inheriting and overriding the virtual operations), the framework can work in a mode where the pre-6.0 API virtual operations are called. When loading a pre-6.0 model, Rhapsody sets the project property `CPP_CG::Framework::UseRhp5CompatibilityAPI` to `True` to set the system-compatibility mode.

If you did not customize the framework behavioral classes, you should be able to move to the new API immediately (by setting this property to `False`). If you customized the framework, you should migrate to the new API before removing the backward compatibility:

- When the property is set to `False`, the framework calls only the new API and therefore overrides on the pre-6.0 virtual operations will compile but will not be called.
- The property affects the framework initialization (in the `main`); therefore, you should verify that when the change is made in the executable component, all library components are using the 6.0 API.
- If you set the property to `False`, you should also remove the overrides at the project level for the following `CPP_CG::Framework` properties:
  - `BooleanType`
  - `TimeoutId`
  - `NullTransitionId`
  - `ReactiveSetTask`

  Note: If you overrode this property yourself (this property was introduced prior to Version 6.0), do not remove the override.

  - `ReactiveCtorActiveArgType`
  - `ReactiveCtorActiveArgName`
  - `ReactiveCtorActiveArgDefaultValue`

Graceful Termination of Reactive Instances

The new version of the framework introduces a new destruction mechanism for reactive objects. The graceful termination of a reactive object is done by calling the `destroy()` operation instead of calling the `delete` operator.

When using `destroy()`, the object waits in a zombie mode until all the events that are designated to it are removed from the active context queue, and then self -destructs. In this scheme, there is no need to traverse the queue of the active context to cancel pending events, and there is no need to make the reactive destructor guarded to ensure safe deletion.

A reactive object can be either in a graceful termination or forced deletion (using the `delete` operator) state: you cannot use graceful deletion on an object that allows forced deletion, and vice versa.

You can set a single reactive object in a forced deletion state, or set the entire system (all reactive instances) in a forced deletion state (as is done for backward compatibility).

Graceful termination should not be used when a reactive part (of a composite class) runs in a context of an active object that is not part of, and different from, the composite active context.

When loading a pre-6.0 model, Rhapsody sets the project property `CPP_CG::Framework::UseDirectReactiveDeletion` to `True` in order to set the system-compatibility mode.

If you want to use graceful reactive termination, do the following:

- Replace the usage of the `delete` operator to destroy reactive objects with calls to the object `destroy()` operation (in the forced deletion state, the operation simply calls the `delete` operator).
- Set the `UseDirectReactiveDeletion` property to `False`.
- Because the property affects the framework initialization (in the `main`), verify that when the change is made in the executable component, all library components use graceful termination.

Timeout Management

In Rhapsody 6.0, the framework moves the responsibility for a timeout cancellation from the timer manager to the timeout client (the reactive object). This change reduces the timer manager responsibilities and the overhead in timeout management (thus improving timeout scheduling performance).

The change also includes changes in the generated code (the user reactive objects hold pointers to the waiting timeouts in order to enable canceling).

When loading a pre-6.0 model, Rhapsody sets the project `CPP_CG::Framework::UseManagedTimeoutCanceling` to `True` to set the system-compatibility mode.

If you want to use the improved timeout management, do the following:

- Set the property to `False`.

- Because the property affects the framework initialization (in the `main`), verify that when the change is made in the executable component, all library components are regenerated.

Usage of Rhapsody Library Components

If you are using a Rhapsody library component as part of an application where the `main` is not generated by Rhapsody (for example, GUI applications), the framework will initialize itself in full compatibility mode on the call to `OXF::init()`.

If you want to remove part or all of the compatibility features, call `OXF::initialize()` instead of `OXF::init()` (the operation takes the same arguments) and add independent, backward-compatibility activation calls prior to the `initialize()` call.

Backward Compatibility Feature	Call
API compatibility	`OXF::setRhp5CompatibleAPI(true)`
Direct reactive objects deletion	`OXF::supportExplicitReactiveDeletion()`
Framework managed timeout canceling	`OXF::setManagedTimeoutCanceling(true)`

Automatic Upgrades Performed by Rhapsody

This subsection documents the changes that Rhapsody performs automatically when you upgrade to Rhapsody 6.0.

Code Generation

Data Member Declarations

Applicability: C++

Code generation was improved to omit redundant namespace prefixes in data members declarations. For example:

```
[Rhapsody 6.0]
namespace PP {
    class C2;
}


namespace PP {
    //## class C1
```

```
    class C1 {
        ...
        C2* itsC2;   //## link itsC2
    };

}

[Rhapsody 5.2]

namespace PP {
    class C2;
}


namespace PP {
    //## class C1
    class C1 {
        ...
        PP::C2* itsC2;  //## link itsC2
    };
}
```

PublicQualifier Property

The `PublicQualifier` property (under `C_CG::Operation`) specifies the qualifier that is printed at the beginning of a public operation declaration or definition.

Note that the **Static** checkmark in the operation dialog UI is disabled in Rhapsody in C because the checkmark is associated with class-wide semantics that are not supported by Rhapsody in C.

When loading models from previous versions, the **Static** checkmark is unchecked; if the operation is public, the `C_CG::Operation::PublicQualifier` property value is set to `Static` in order to generate the same code.

See the *Properties Reference Guide* for more information on the `PublicQualifier` and `PrivateQualifier` properties.

Features Disabled for Backward-Compatibility

This subsection describes features that are automatically disabled when loading pre-6.0 models.

MultiMakefileGenerator

For backward compatibility, the generator continues to remove spaces from the user-specified file paths. You can prevent the removal of spaces by setting the `CG::Configuration::RemoveWhiteSpacesInBuildFile` property to `False`. This property is set to `True` at the project level when you load a pre-6.0 model.

Full Roundtrip

Applicability: C

Rhapsody 6.0 introduces full roundtrip capabilities to Rhapsody in C. This capability is disabled when loading pre-6.0 models because it requires new code generation annotations.

You can enable full roundtrip by setting the `C_Roundtrip::General::RoundtripScheme` property to `Full` and regenerating the code.

Note

It is not recommended to change the property and roundtripping without regenerating the code.

Additional Information

This subsection contains information on miscellaneous changes and enhancements in Rhapsody 6.0.

Code Generation

Instrumentation of Composite Classes

Applicability: C++ animation and tracing

When generating a non-reactive composite class without attributes, an `OM_DECLARE_COMPOSITE_OFFSET` macro is added to the class declaration. This macro ensures proper representation of the composite and the part in animation or tracing mode. You can still mix instrumented and non-instrumented code.

Flat Statechart Macro

Applicability: C++

The `#define _OMFLAT_IMPLEMENTATION 1` statement was removed from the generated code.

OMDECLARE_GUARDED

Applicability: C++

A redundant declaration of `OMDECLARE_GUARDED` in guarded classes that inherit from guarded classes was removed.The derived classes use the guards declared in the base classes.

Import Statements

Applicability: Rhapsody in J

Java import statements from dependencies with «`Usage`» stereotypes to a package are generated regardless of the package's `CG::Package::GeneratePackageCode` property value.

This means that you might find additional import `<full package name>.*` statements in your code.

Framework

C

The Version 6.0 framework for Rhapsody in C includes the following changes:

- An INTEGRITY adapter was added.
- `oxf`
  - `Ric.h`—The definition of `_OMINSTRUMENTED` was refined to avoid multiple definitions.
  - `RiCTask`—There is a new boolean flag, `deletionAllowed`, that is used to prevent deletion of the RTOS task. This is used to prevent early destruction of the `RiCHandleCloser` task until all client tasks are done (when terminating the application).
  - `RiCOSWrap.h`—The functions `RiCOSTask_InitCommunicationLayer()` and `RiCOSTask_CleanupCommunicationLayer()` were added to the OSAL definition. The functions were added to support INTEGRITY, which requires per-task IP initialization. The functions are reduced to empty macros if `RIC_NEED_INIT_COMMUNICATION_LAYER` is not defined.
  - `RiCHandleCloser_Init()`—The redundant `RiCTask` argument was removed, the task initialization arguments were modified, and the task name (`_omCloseHandler`) was specified. In addition, this version of the

framework sets the task `deletionAllowed` flag to `False` to prevent premature destruction. A client adapter should set the flag to `True` at the end of the `RiCOSEndApplication()` implementation. See the INTEGRITY adapter for a sample implementation.

- pSOS adapter—Now the function `RiCOSEndApplication()` cleans up the main task (by calling `RiCTask_cleanup()`) instead of destroying it. The main task cannot be destroyed because it is statically allocated.
- VxWorks, POSIX adapters—Moved the cleanup of the `m_SuspEventFlag` from `RiCOSTask_start()` to `preExecFunc()` to prevent a potential race between the owner thread and the signaling thread.

- `WebComponents`—`RPY_LaunchWebServer()` supports INTEGRITY (initialization of the IP layer).

C++

- In the QNX and VxWorks adapters, the cleanup of the `m_SuspEventFlag` was moved from the `OMOSThread::start()` implementation to `preExecFunc()` to prevent a potential race between the owner thread and the signaling thread.\
- The meaning of `OM_FROCE_STDIO` has been changed. Instead of locally disabling iostreams just for `OMNotifier` (was `OMOutput`) functions, it now disables iostreams throughout the OXF by acting within *OXFSelectiveInclude.h*. For example, code in *OXFInstrumentation.cpp* will be disabled.

Linux/MVL Adapters

Applicability: C and C++

A time-correction mechanism was added to overcome an accuracy issue in the RTOS.

Properties

VxWorks MakeFileContent

Applicability: C and C++

The property was modified to support compilation on a Linux/Solaris host, as well as on Windows.

C++

- The default value of the `CPP_CG::Framework::ReactiveSetTask` property was modified from `"setThread($task, $isActive);"` to `"setActiveContext($task, $isActive);"` to comply with the C++ framework API change. If you did not override this property, Rhapsody sets the property value back to the original value when loading pre-6.0 models.

- A spelling correction was made in the content of the `CPP_CG::<Environment>::ReusableStatechartSwitches` property.

Java

The `InvokeExecutable` property under the JDK environment was modified to support projects located in a path that contain spaces in directory names.

Ada

All the code generation properties (`Ada_CG`) were moved from the `factoryAda.prp` file to the `sodius.prp` file (which the factoryAda.prp includes).

MultiMakefileGenerator

This version of the MultiMakefileGenerator includes the following enhancements:

- Support for Ada and C, as well as C++
- Support for the following properties:
  - `CG::Configuration::GenerateDirectoryPerComponent`
  - `DefaultSpecificationDirectory` and `DefaultImplementationDirectory` configuration properties
  - `CG::File::ImpExtension` and `SpecExtention`
- Avoids removing spaces in include files and library paths
- Better support for external elements (including external component files)
- Support for ports with implicit contracts
- Support for the `linker_file` option using the environment property `LinkerFile`
- Support for the `integrate_file` option using the environment property `IntegrateFile`
- Support for the `bsp_description` option using the environment property `BSPFile`
- Support for the `connections` option using the environment property `ConnectionFile`
- Support for the `nobuild` option using the environment property `ResourceFile`
- Support for nested components (as subprograms)
- Supports the new version of the C++ OXF

To support these enhancements, the following modifications were made in the `IntegrityBuildScript.bas` file:

- The following operations were modified:
  - `Main()`

  - `AddMainBLDFile()`
  - `AddBuildFileHeader()`
  - `CreateBuildFile()`
  - `WriteBuildOptions()`
  - `AddUserIncludePath()`
  - `AddUserLibraries()`
  - `AddUserHeaders()`
  - `AddEXECompileProperties()`
  - `AddLIBCompileProperties()`
  - `WriteOXFDirs()`
  - `WriteUninstrumentedLibraries()`
  - `WriteTracerLibraries()`
  - `AddFile()`
  - `AddHeaderFile()`
  - `AddCFile()`
  - `AddLIBCompileProperties()`
  - `AddUserSources()`
  - `getImpExtension()`
  - `getSpecExtension()`
  - `PackageFileName()`
  - `WriteClass()`
  - `ClassFileName()`
  - `ElementFilePath()`
  - `WriteActor()`
  - `ActorFileName()`
  - `WritePackage()`
  - `AddBLDIncludeAdditionalLD()`
  - `WriteDefines()`
  - `AddReusableDefine()`
  - `AddWebFiles()`
  - `GeneratePackage()`

- The `isInSubdirectory()` operation was replaced with a new COM API service, `RPComponent.isDirectoryPerModelComponent()`, that performs a more accurate check.
- The following new operations were added:

- `AddPackageToComponentNestedElements()`—Supports the generation of a separate directory for each package.
- `getDefaultSpecDirectory()` and `getDefaultImpDirectory()`—Support the `DefaultSpecificationDirectory` and `DefaultImplementationDirectory` configuration properties.
- `AddImpFile()`—For multiple language support.
- `ElementInScope()`—Checks that the specified element is in the provided scope.
- `CreateParentDirectory()`—Creates the specified directory if it does not already exist.
- `removeSpaces()`—Removes spaces from the string if the `CG::Configuration::RemoveWhiteSpacesInBuildFile` property is set to `True`.
- `getIntegrityPath()`—Gets the integrity root. The operation first checks for the `INTEGRITY_ROOT` environment variable, then the `IntegrityRoot` environment property.
- `WriteSocketLibraries()`—Adds language-dependent socket libraries.
- `WriteWebLibraries()`—Adds language-dependent, Web-enabling libraries.

Upgrading to Version 5.2 MR-1

The changes in version 5.2 MR-1 of Rhapsody are listed below.

For information on changes to Rhapsody in Ada, see the `RiA_Changes.pdf` document under `<Rhapsody>\Sodius\help`.

Changes that Require User Action

This subsection documents the changes that require you to perform some actions when you upgrade to Version 5.2 MR-1.

Code Generation

Template-Based Descriptions

Applicability: C and C++ where template-based descriptions were used

Keywords in the headers or footers generated for the main configuration files (for example, `MainCmp.cpp` and `MainCmp.h`) are resolved based on the active *configuration* tags instead of the active *component* tags.

If you used tags in the component to control the main file's headers or footers, move the tags to the configuration to enable resolution of the header or footer keywords according to the tags' values.

C++ Properties

The following entries were added to the `CPP_Roundtrip::General::PredefinedMacros` property:

```
OM_DECLARE_COMPOSITE_OFFSET
IMPLEMENT_META_T_S_T(tname\\,IsSingleton\\,SuperClass\\
,animSuperClass\\,animTname)
IMPLEMENT_META_T_S_T_N(tname\\,IsSingleton\\,NameSpace\\,
SuperClass\\,animSuperClass\\,animTname)
```

If you have overridden this property, you must add the value to your override value to guarantee proper behavior of full roundtrip in instrumented mode.

Additional Information

This subsection contains information on miscellaneous changes and enhancements in Version 5.2 MR-1.

Code Generation

Instrumentation of Composite Classes

Applicability: C++ animation and tracing

When generating a non-reactive composite class without attributes, an `OM_DECLARE_COMPOSITE_OFFSET` macro is added to the class declaration. This macro ensures proper representation of the composite and the part in animation and tracing. You can still mix instrumented and non-instrumented code.

In addition, the `IMPLEMENT_REACTIVE_META_S` macro content was modified to reflect the model structure.

C++ Framework

The following table lists the changes to the C++ framework in Version 5.2 MR-1.

Interface	Change
`OMString` constructors	Handle cases when an empty string is passed as an argument.
`OMThread`	• `queueEvent()`—Handles attempts to queue an event into an instance that is under destruction. • `stopAllThreads()`—Iteration over active threads was corrected. • `doExecute()`—Calls `cleanupThread()` instead of `destroyThread()` if the thread returns `False` to the call to `allowDeleteInThreadsCleanup()`.
`OMMainThread`	• Calls `setDeletionAllowed(FALSE)` in the constructor to prevent attempts to dynamically destroy the statically allocated thread. • `destroyThread()`—Handles multiple calls to the operation.
`OMHandleCloser` constructor	• Calls `thread.setDeletionAllowed(FALSE)` to prevent attempts to dynamically destroy the statically allocated thread. • The INTEGRITY, Microsoft, and Nucleus adapters use `OMHandleCloser` for final cleanup of destroyed operating system threads.

Upgrading to Version 5.2

The changes in version 5.2 of Rhapsody are listed below.

For information on changes to Rhapsody in Ada, see the `RiA_Changes.pdf` document under `<Rhapsody>\Sodius\help`.

Changes that Require User Action

This subsection documents the changes that require you to perform some actions when you upgrade to Version 5.2.

Code Generation

GenerateDirectoryPerModelComponent Property

Applicability: C and C++

The generation of a separate directory for each package is controlled by the `CG::Configuration::GenerateDirectoryPerModelComponent` property. Because of a defect, it was possible to control the behavior for a specific package by overriding this property at the package level. However, this defect was fixed in Rhapsody 5.2. This means that the property now affects code generation only when the override is done at the *configuration* level or higher.

If you overrode the property at the package level, you can maintain this behavior by defining the `CG::Component::PackageCtrlDPMC` property (type: `Bool`, value: `True`) in your `site.prp` file.

If you set the `PackageCtrlDPMC` property, note that when Rhapsody calculates file names of elements mapped to related components (via dependencies between components), overrides on the `GenerateDirectoryPerModelComponent` property in the related component or configuration context are ignored. Therefore, any overrides of the `GenerateDirectoryPerModelComponent` property should be made at the property file level or package level.

Generation of Variables

Applicability: C and C++

Prior to Rhapsody 5.2, variables under a package that were marked as protected or private (often because they had been attributes of a class and were then moved under the package as variables) were erroneously generated as public in the header file. In Rhapsody 5.2, such variables are generated according to their settings. Therefore, compilation errors might occur if the variable was marked as protected or private (prior to Version 5.2), but other code relied on that variable to be generated into the header file. In such cases, you should change the variable to be public.

Ports

Applicability: C++ with ports

A defect in code generation for ports ignored the port contract in the case of a single interface (provided or required), which declared event receptions as fixed. This means that only event receptions that were part of the contract were processed—all other events were ignored.

If your application took advantage of this defect, you should complete the specification of the port contract.

Automatic Upgrades Performed by Rhapsody

This subsection documents the changes that Rhapsody performs automatically when you upgrade to Version 5.2.

Modeling of External Elements

Applicability: All languages

Rhapsody modeling of external elements was enhanced in both code generation and reverse engineering.

To provide simpler modeling of external packages, Rhapsody 5.2 introduces the property `CG::Package::UseAsExternal`. When this property is set to `True`, all the package aggregates are considered external.

If a package in a pre-Version 5.2 model had the `CG::Class::UseAsExternal` property set to `True`, the override moves to the `CG::Package::UseAsExternal` property when the model is loaded. This causes all other aggregates of the package (for example, its types) to be external.

Code Generation

Composite Classes

Applicability: C and C++

In previous releases, every composite class was considered a reactive class. In Version 5.2, this scheme has been refined (and is aligned with Rhapsody in J) so a composite class is reactive only if it is reactive by itself (for example, it has a statechart) or one of its parts is reactive. As a result, a non-reactive composite with non-reactive parts is no longer generated as a reactive class.

This change is disabled on load of pre-Version 5.2 models by setting the `CG::Class::ReactiveSimpleComposites` property to `True` at the project level.

Constant Variables as #define

Applicability: C

Starting with Version 5.2 of Rhapsody in C, constant variables (variables with the **Constant** modifier checked) are generated as `#define` statements by default.

When pre-Version 5.2 models with constant variables are loaded in 5.2, the `C_CG::Attribute::ConstantVariableAsDefine` property is set to `False` at the variable level to avoid changes in the generated code.

Reverse Engineering

In reverse engineering, if you specify the option to map each directory to a package (the default behavior starting with Rhapsody 5.2), the `CG::Configuration::GenerateDirectoryPerModelComponent` property is automatically set to `True` in the context of the active configuration.

Features Disabled for Backward-Compatibility

This subsection documents the 5.2 functionality that Rhapsody disables for models created using previous versions.

Code Generation

External Elements

Applicability: C, C++ and Java

As part of the enhancements to external element modeling, code generation was modified to take advantage of information in modeled external elements. This enhancement enables you to specify how to initialize objects of external classes, create links to external elements, and so on.

Rhapsody automatically disables these enhancements on loading of pre-5.2 models by setting the following properties at the project level:

- `CG::Configuration::StrictExternalElementsGeneration` is set to `True`.
- `CG::Configuration::SupportExternalElementsInScope` is set to `False`.

Generating the Makefile Search Path

Applicability: C and C++

Makefile generation was enhanced to add the **Include Path** field of the related components and configurations to the makefile search path, in addition to the component path. The include path information is added when the components are associated with a «`Usage`» dependency and the `CG::Dependency::UsageType` property is set to `Specification`.

When the `UsageType` property is set to `Implementation` or `Existence`, only the related component path is added to the makefile search path.

The enhanced search path is disabled on load of pre-Version 5.2 models by setting the `CG::Component::RelatedComponentsIncludePathInMakefile` property to `False` at the project level.

Reverse Engineering

Import of Structures

Applicability: C and C++

A new option was added to reverse engineering that enables you to import structures (`struct`) as Structure types.

By default, the property `<lang>_ReverseEngineering::ImplementationTrait::ImportStructAsClass` is set to `False` for models created in Version 5.2. Rhapsody sets this property to `True` for a configuration in a pre-Version 5.2 model for which reverse engineering settings were defined.

Additional Information

This subsection contains information on miscellaneous changes and enhancements in Version 5.2.

Code Generation

Order of Attribute Initialization

Applicability: C, C++ and Java

The order of attribute initialization in the constructor initializer (C++) and body was fixed for the ordering scheme of default attributes to match the order of the attributes generated in the code.

This change was done primarily to align with common, good practice in C++.

Template-Based Descriptions

Applicability: C and C++

The replacement of tags in the description was modified to take the default tag values into account. As a result, if `$MyTag` appears in a description template of an element and `MyTag` is not overridden in the specified element, the default value of `MyTag` will be used.

Ports

Applicability: C++

The event argument name in the code generated for ports was modified from `evt` to `event` to resolve a compiler warning related to differences between the argument names in overridden virtual operations (GHS MULTI environment).

New Lines

Applicability: C

The generation of new lines has been modified to increase the readability of the generated code.

Reactive V-Table Initialization

Applicability: C

The `ROOT_STATE_SERIALIZE_STATES` macro in the v-table initialization has been modified to `ROOT_STATE_SERIALIZE_STATES(<serializeStates name>)` to support the naming pattern of the `serializeStates` function name for the files modeling feature.

Annotations

Applicability: C++

In full roundtrip, the types' ignore annotations are now generated as a single ignore block to increase the readability of the generated code.

Framework

Applicability: C++

Minor changes were made in the framework to comply with GCC 3.2. These changes have no behavioral effect.

Upgrading to Version 5.0.x

The changes in version 5.0.1*x* of Rhapsody are listed below.

Upgrading to Version 5.0.1 MR2

This subsection describes behavior and functionality changes between versions of Rhapsody that you must consider when upgrading your installation to Version 5.0.1 MR2.

Changes that Require User Action

The Rhapsody model checker was fixed to check nested objects and blocks. As a result, the model checker might detect new errors and warnings in your model. Before generating code, you must fix the errors.

Keyword Behavior Changes

Code generation for the `$FullCodeGeneratedFileName` keyword in file header/footer properties was fixed. As a result, redundant "." and ".." in the file paths are removed.

For example, for class A's specification, when the active configuration directory is set to ".", the keyword is extracted to `DefaultComponent\A.h` instead of `DefaultComponent\.\A.h`.

Property Changes

In Rhapsody in C++, the new keyword `$OMReusableStatechartSwitches` was added to the `<lang>_CG::<Environment>::MakeFileContent` properties. This is a reserved keyword that expands to nothing.

In Rhapsody in J, the Java containers' `StaticArray::Add` property was fixed to generate correct code for the static array `add` operation.

Upgrading to Version 5.0.1 MR1

No user actions are required.

Upgrading to Version 5.0.1

This subsection describes behavior and functionality changes between versions of Rhapsody that you must consider when upgrading your installation to Version 5.0.1.

Changes that Require User Action

This documents the changes that require you to perform some actions when you upgrade to Version 5.0.1.

Adapters

The VxWorks adapter (C/C++) was upgraded to VxWorks 5.5 (Tornado 2.2).

In order for the IDE integration with Tornado 2.2 to work, you must add the following to the path:

```
<Tornado 2.2>\host\x86-win32\bin
```

Code Generation (RiC)

The method `RIC_SET_EVENT_DESTROY_OP(me, <event name>);` is called in `<event>_Init()` in every event that has the `RiC_Destroy_<event>()` operation. This means that the generated destroy operation is called instead of the generic one. This change was already done for events with memory pools, but was missing for other events. If you modified the `C_CG::Event::FreeMemory` property, the Version 5.0.1 changes will affect the behavior of your application.

Framework Changes

This subsection contains information on the changes to the framework for Rhapsody in C and C++.

RiC Framework Changes

In `RiCTimeout_cleanup()`, a call to `RiCTimerManager_softUnschedTm()` was added to guarantee cleanup of the timeout from the matured list.

RiC++ Framework Changes

The Version 5.0.1 changes are as follows:

- In `ntos.h`, the include to `<afx.h>` was replaced with an include to `<windows.h>` (to remove the redundant dependency on MFC).

- In `OMString`, the operation `GetBuffer(int)` was added (in addition to the existing `GetBuffer(int) const`). The new operation has the same semantics as the MFC method `CString::GetBuffer(int)`.
- In `VxOS.cpp`, the `VX_FP_TASK` flag was added to `taskSpawn()` calls.

Upgrading to Version 5.0

The changes in version 4.0 of Rhapsody are listed below.

For information on changes to Rhapsody in Ada, see the `RiA_Changes.pdf` document under `<Rhapsody>\Sodius\help`.

Changes that Require User Action

This subsection documents the changes that require you to perform some actions when you upgrade to Version 5.0.

Changes in the Framework Files

If you are using a customized environment that includes framework makefiles, you must make the changes described in here to your framework files.

The following changes were made to the framework files:

Language	Affected File	Description of Change
C (Share\LangC)	`oxf/RiCHdlCls.*`	New files. This file contains a generalization of the thread closer class.
	`oxf/RiCNTHdlCls.*`	Removed. This change affects Win32 adapters only.
C++ (Share\LangCpp)	`oxf/HdlCls.*`, `oxf/ThdSup.*`	New files. These files contain a generalization of the thread closer class.
	`oxf/IntHdlCls.*`, `oxf/IntThdSup.*`	Removed. These files affect the INTEGRITY adapters only.
	`oxf/NTHdlCls.*`, `oxf/NTThdSup.*`	Removed. This change affects Win32 adapters only.
	`oxf/OMDefaultReactivePort.*`	New files. This file supports ports modeling.

COM API

Rhapsody 5.0 includes the following changes to the COM API.

Changes in Hierarchy

The changes in the hierarchy of the COM API requires you to recompile type-safe COM clients (such as COM clients written in C++) before you can use them with Rhapsody 5.0; other clients (such as VB COM clients) do not need to be recompiled.

The changes are as follows:

- The base interface of `IRPAttribute`, `IRPArgument`, and `IRPTemplateParameter` was modified from `IRPModelElement` to `IRPVariable`. This change aligns with the UML where `IRPVariable` is the Rhapsody COM API representation of the UML `TypedElement`.
- The base interface of `IRPConstraint` was modified from `IRPModelElement` to `IRPAnnotation`.

Renamed Metaclasses

The following metaclasses were renamed:

- `activityDiagram` was renamed to `ActivityDiagram`.
- `LinkInstance` was renamed to `Link`.
- `Relation` was renamed to `AssociationEnd`.

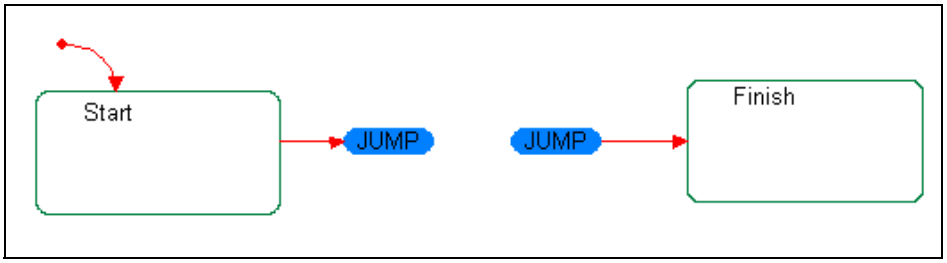
You must rename the metaclasses in your code to maintain the correct behavior.

Changes to Statecharts and Activity Diagrams

This change applies to users who traverse or create statecharts and activity diagrams using the COM API.

Diagram, stub, and junction connectors for statecharts and activity diagrams were added to the Rhapsody 5.0 repository. This causes some changes in the COM API behavior. The properties `itsSource` and `itsTarget` of `RPTransition` in Rhapsody 4.2 never showed connectors—they always jumped to the source or target of the incoming or outgoing transition for the connector. Rhapsody 5.0 uses the properties `itsSource` and `itsTarget` to show the connector itself.

Consider the following example:



In Rhapsody 4.2, the property `itsTarget` of the outgoing transition of the Start state will show the Finish state; the property `itsSource` of the incoming transition of the Finish state will show the Start state.

In Rhapsody 5.0, the property `itsTarget` of the outgoing transition of the Start state will show the connector JUMP; the property `itsSource` of the incoming transition of state Finish will show the connector JUMP.

To get the old behavior in Rhapsody 5.0, use the following new functions for `RPConnector`:

- `getDerivedInEdges() As RPCollection`—Returns all incoming transitions for the corresponding connector
- `getDerivedOutEdge() As RPTransition`—Returns the outgoing transition for the corresponding connector (there can be only one)

Using the example, to get the state Finish as the target of the outgoing transition of state Start, do following:

```
* RPConnector connector=transition.itsTarget //connector JUMP
* RPTransition outgingTransition=connector.getDerivedOutEdge()
// transition from JUMP to Final
* RPState state = outgingTransition.itsTarget // state Final
```

Use similar code to get the Start state as the source of the incoming transition of the Final state.

IRPUnit::load()

The `IRPUnit::load()` method includes the following new argument:

```
[out, retval] IRPUnit** ret
```

The argument returns the loaded unit. If you are using the API in a C++ COM client, you must update the arguments of the call to reflect this change.

DOORS

Beginning with Rhapsody 5.0, the element's description is exported into a new text attribute in DOORS called **Description**.

If you have a pre-5.0 Rhapsody model that was exported to DOORS, you should re-export the model to add this new attribute, and have the description exported to it. The description in the old attribute will be deleted.

Alternatively (but less recommended), you can open the old, exported model and select **Check Data**. The new attribute will be added to DOORS. The check will prompt you for each element that has a description (because the new attribute will be empty) and you can update. After the update, the description in the old attribute will be deleted.

C++ Interfaces

Interfaces (classes stereotyped as «`Interface`») are implemented as reactive interfaces—as if they were stereotyped «`Reactive Interface`» — if they have event receptions or other reactive features. The implementation of interfaces without reactive features remains the same.

This behavior can be disabled by setting the property `CPP_CG::Class::IsReactiveInterface` to `FALSE`. For 5.0, the default value of this property is `TRUE` for all classes. However, while loading pre-5.0 models this property is overridden to `FALSE` to maintain backward compatibility (unless you previously overrode the property).

To use the new behavior, you must remove the override this property for the project.

HeaderDirectivePattern Property Value

The default value of the property `CG::File::HeaderDirectivePattern` property was changed to `"$FULLFILENAME_H"` in Version 5.

If you overrode this property value in your `.prp` files, you will see a change in the generated `#ifndef <file name>_H` statements (in C and C++). In addition, there is the potential for collision of the `#ifndef` that might result in compilation errors.

DiffMerge of Pre-Version 5.0 Models

If you try comparing a Version 4.*x* model to itself or to a previous revision of the model and it has statecharts in which a single state has more than one diagram connector with the same name, DiffMerge erroneously detects differences.

For example, in the state doorClosed in the class `Dishwasher` statechart, there are two diagram connector objects with same name (DONE). If you compare the model to itself, DiffMerge will detect a difference.

To overcome the problem, load the model in Rhapsody 5.0 and save it.

EmbeddedScalar::Set Property

The following change affects users who have their own container set (`XXContainers` instead of `OMContainers`) or who override the containers properties):

> The property `<C++ container set>::EmbeddedScalar::Set` was removed as part of the support for classes as attribute types.

Code Generation

Rhapsody 5.0 includes the following changes to code generation:

- C++ and Java changes

  When you set the kind of an attribute or relation (C++ only) to **Abstract** (using the `Kind` property), only the accessors and mutators of the attribute or relation are generated (as abstract operations)—the data member itself is no longer generated. This change enables you to model interfaces using attributes and relations as well as operations. If you used a data member created for an abstract attribute or relation, do one of the following:

  - Change the kind to **Virtual**.
  - Fix the model by recreating the attribute or relation in the derived classes.

- Now the property `CG::Generalization::Generate` affects the generated code. If you have set the property in your model, you must undo the change to generate the same code.

C++ Framework

Copy constructors and assignment operators were added for all containers. (The copy is done using an assignment operator).

If you are using `OMList` or `OMStack` to hold elements by value (for example, `OMList<Point> points`), the contained elements must define a `bool` operator that checks its address against the `OMNullValue<>` template instantiation (defined in `oxf/abscond.h`). For example:

```
operator bool() {
    return (this != &OMNullValue<Point>::get());
}
```

Automatic Upgrades Performed by Rhapsody

This documents the changes that Rhapsody performs automatically when you upgrade to Version 5.0.

Explicit Initial Instances

Beginning with Version 5.0, Rhapsody does not include explicit initial instances as part of the scope. In other words, in explicit mode, code is not generated for a class just because it is in the list of initial instances for the configuration.

For existing models, Rhapsody sets the `CG::Configuration::AddExplicitInitialInstancesToScope` property to `True` at the project level to maintain the old behavior.

This change enables you to use the list of initial instances to create instances that their classes defined in related components (libraries).

Code Generation Format

Redundant spaces added by code generation at the end of actions that contained one or more spaces were removed. In addition, redundant spaces in argument declarations were removed.

GenerateWithAggregates Property

The `CG::Package::GenerateWithAggregates` property was removed. This property was used to map packages without their descendants to code. Now, the mapping is stored as part of the scope repository.

When you load an existing model, Rhapsody removes any overrides of this property.

Enabling the Rhapsody 5.0 Features

This documents the changes that require you to perform some actions in order to use the new Version 5.0 functionality.

Attribute Modifiers

In Rhapsody 5.0, you can set additional modifiers for attributes using the check boxes **Multiplicity**, **Constant**, and **Reference** in the Features dialog box. In addition, the generation of

attributes was modified so Rhapsody often uses a container (as in relation generation) to generate the code.

The containers map **Constant** and **Reference** to code using the new keywords `$constant` and `$reference`. You must add these keywords to the containers properties (as is done in the `factory<lang>.prp` files) to enable correct code generation for these modifiers.

This change affects users who have their own container set (`XXContainers` instead of `OMContainers`) or who override the containers properties).

Typedef Modeling

Rhapsody 5.0 introduces composite type modeling, including enumerations and typedefs.

Typedef code generation is based on a new property in the containers called `FullTypeDefinition`. If you use your own container set (`XXContainers` instead of `OMContainers`), you must add this property to enable generation of Typedef types. Otherwise, Typedef types will not be generated.

For examples, refer to the container properties in the `factory<lang>.prp` files.

Cross-Package Links

Rhapsody 5.0 supports automatic runtime connection of instances across packages. To enable correct code generation, you must set the property `CG::Component::InitializationScheme` to `ByComponent`.

Note that there is a check (warning) to alert you if your setting is incorrect.

Additional Changes

This subsection contains information on additional changes.

Framework Changes

The following sections describe additional changes to the framework.

C Framework

Rhapsody 5.0 includes the following changes to the C framework:

- The new file `oxfFiles.list` lists all the common files in the framework. This list is included by the makefiles.

- Typedef statements were added to `RiCTypes.h` to support language-independent types (for example, `typedef char* RhpString`).
- The thread cleanup classes (which clean up thread resources after self-termination of a thread) that were available in several RTOSes was generalized and is now available for all RTOSes. The generalized class name is `RiCHandleCloser`. The general closer is located in the files `RiCHdlCls.*`, which replace the operating-specific files.

  The mechanism works by instantiation of the thread closer singleton and registering a cleanup function. The closer is supported on Win32 (Windows) and Nucleus.
- In `RiCReactive`, the return type of the `RiCDispatchEvent` function pointer was modified to conform to changes in the code generation for MISRA compliance.
- The definition of `RIC_EMPTY_STRUCT` was modified from `char` to `RiCBoolean`.
- A cast to `RiCBoolean` was added to the definition of `RiCTRUE` and `RiCFALSE`.
- The term "`object_type`" is obsolete; "`class`" is used instead.

C++ Framework

Rhapsody 5.0 includes the following changes to the C++ framework:

- In `OMEvent`, a new attribute port was added to the event so you can access the port that the event was sent on (using the method `getPort()`).
- Three new classes were added to support port modeling:
  - `OMDefaultReactivePort`
  - `OMDefaultOutBound`
  - `OMDefaultInBound`

  ```
  These classes are defined in the file
  OMDefaultReactivePort.*
  ```
- New macros were added to `oxf.h` and `ioxfreactive.h` to support port actions.
- Typedef statements were added to `rawtypes.h` to support language-independent types (for example, `typedef OMString RhpString`).
- Copy constructors and assignment operators were added for all containers, The copy is done using the `operator =()` of the contained class.
- The thread cleanup classes (which clean up thread resources after self-termination of a thread) that were available in several RTOSes was generalized and is now available for all RTOSes. The generalized class name is `OMHandleCloser`. The general closer is located in the files `HdlCls.*` and `ThdSup.*`, which replace the operating-specific files.

  The mechanism works by instantiation of the thread closer singleton and registering a cleanup function. The closer is supported on Win32 (Windows), INTEGRITY, and Nucleus.

- The `__DIAB` compilation flag was replaced with `__DCC__`, a flag defined by WindRiver Diab compiler. This change avoids the definition of an additional flag.
- In the file `osconfig/WIN32/osconfig.h`. multiple definitions of `OM_WIN32_COMPILER_DISABLE_WARNING_4244` were removed so now there is a single definition.
- In the `OMTMMessageQueue` interface, the method `isFull()` was added for the event queue API.
- `IOxfReactive` contains the following changes:
  - A new status flag, `OMRBehaviorStarted`, was added to the `omrStatus` attribute. The `OMRBehaviorStarted` flag signals that `startBehavior()` was called.
  - `OMRBehaviorStarted` includes the following new methods:

```
bool isBehaviorStarted() const
void setBehaviorStarted()
```

These methods are called by `startBehavior()` to prevent multiple taking of the default transitions on multiple calls.

- The definition of `NEW_DUMMY_PARAM` was removed from `MemAlloc.h`. The framework uses the definition in `ommemorymanager.h`.
- The `omendl` definition was added to the `omiotypes.h` file. This macro enables you to use `endl` or `std::end1`, based on the value of the `OM_STL` flag.

Java Framework

The `RiJStateReactive` class includes a new, private, Boolean attribute: `isBehaviorStarted`. This attribute is checked and set in `startBehavior()` to avoid multiple taking of the default transition on multiple calls.

Code Generation

Rhapsody 5.0 includes the following changes to code generation:

- The `EntryPointDeclarationModifier` property now affects any environment for which it is defined.
- Any generated block statement (`for`, `while`, `if`, `switch`) action code is generated wrapped in parenthesis. For example:

```
{<code>}
```

- In Rhapsody in C and C++, dependencies with `<<Usage>>` stereotypes between components result in dependencies within the makefile.

This means that if you rebuild the dependent component, the dependent makefile will automatically cause a re-link.

- In Rhapsody C and C++, when the property `CG::Package::GeneratePackageCode` is set to a value other than `Always`, code will not be generated for empty packages in Instrumented mode. “Empty” packages are packages that have no elements or classes, but might contain other packages.

Generation of MULTI Build Files

You can add switches to the build file using the property `CPP_CG.::<Environment>::BLDIncludeAdditionalBLD`. See the *Properties Reference Guide* for more information.

MISRA Compliance Changes (RiC)

This version contains the following changes:

- The values `FALSE` and `TRUE` were replaced with `RiCFALSE` and `RiCTRUE`, respectively.
- The return type of `<class>_dispatchEvent()` and `<class>_takeEvent()` was changed from `int` to `RiCTakeEventStatus`.

  The type of the `res` local variable declared within these methods was modified accordingly.

Changes in Default Property Values

The following sections list the properties whose default values were changed in Version 5.0. When loading pre-5.0 models, the property values are converted to maintain backward compatibility.

See the *Properties Reference Guide* for detailed information on the Rhapsody properties.

General Changes

Version 5.0 includes the following property changes:

- The properties `In`, `InOut`, `Out`, and `ReturnType` (under `CG::Type`) were moved to `<lang>_CG::Type` and were assigned language-dependent values.
- The `<lang>_CG::Attribute::MutatorGenerate` property was changed from a Boolean value to an enumeration. The enumeration values are `Always`, `Never`, and `Smart`.
- The containers (`RiCContainers`, `OMContainers`, and so on) properties were modified to support attribute modifiers and Typedef types.

Ada

The `Ada_CG::GNAT::InvokeMake` property was modified to take advantage of the `GnatMake.bat` file to set the environment for the build.

C

Version 5.0 includes the following changes to the C properties:

- A new environment for Borland was added to the supported environments.
- To conform to MISRA rules, the default value for the property `C_CG::Framework::ActiveInit` was changed to the following:

  ```
  "$base_init($member, RiCFALSE, $Vtbl)"
  ```

- The default value for the property `RiCContainers::EmbeddedScalar::RelationTargetType` was changed to `"$CType*"` to support the accessor/mutator for a by-value attribute using a class as its base type.
- The default value for the property `RiCContainers::EmbeddedScalar::Set` was changed to support a mutator for a by-value attribute using a class as its base type. The new value is as follows:

```
"memcpy((void*)&$me$cname,(void*)$item, sizeof($target))"
```

C++

Version 5.0 includes the following property changes:

- The default value for `CPP_CG::Class::Embeddable` was changed to `True`. This means that by-value allocation of objects is preferred over dynamic allocation.

  Note that in existing models, this value is automatically set to the Version 4.2 value (`False`) to maintain backward compatibility.
- The default value for `CPP_CG::Class::IsReactiveInterface` was modified to `True`. See **C++ Interfaces** for more information.
- The default value for `CPP_CG::Relation::ImplementWithStaticArray` was changed to `FixedAndBounded`. This means that C-style arrays are preferred over container classes.

  Note that in existing models, this value is automatically set to the Version 4.2 value (`Default`) to maintain backward compatibility.
- Brackets were added in ATL and COM properties.
- The metaclass `CORBAObjectRefrence` was renamed to `CORBAObjectReference`.
- The environment `CPP_CG::MicrosoftWinCE` was removed: WinCE 3.X is no longer supported.

- For the property `CORBA::<ORB>::CPP_StandardInclude`, the Rhapsody code generator now generates the value `<CORBA.h>` instead of `"CORBA.h"`.
- The `CORBA::` prefix was removed from the default values of the following properties:

Metaclass	Property
`C++Mapping_CORBABasic`	• `in` • `inout` • `out` • `ReturnValue` • `TriggerArgument`
`C++Mapping_CORBAEnum`	• `in` • `inout` • `out` • `ReturnValue` • `TriggerArgument`

Deprecated COM APIs

As part of the Rhapsody 5.0 type composition feature, `IRPVariable` was changed from `IRPType` to `IRPClassifier`, and a new API was introduced to support the change. You can still use the existing APIs related to `IRPType` as long as the model does not violate this assumption.

The following COM API properties are deprecated and should not be used:

Property	Action
`IRPVariable.typeOf`	Use `type` instead. This change is also true for `IRPVariable`-derived interfaces (`IRPAttribute`, `IRPArgument`, and `IRPTemplateParameter`).
`IRPOperation.returnType`	Use `returns` instead.
`IRPTemplateParameter.typeName`	Use `type` instead.
`IRPConstraint.constraintsByMe` (RO)	Use `anchoredByMe` instead.

Note

The change in the hierarchy of `IRPConstraint` made the `constraintsByMe` property a duplicate of the `IRPAnnotation` `anchoredByMe`.

Upgrading to Version 4.2

The changes in version 4.2 of Rhapsody are listed below.

For information on changes to Rhapsody in Ada, see the `RiA_Changes.pdf` document under `<Rhapsody>\Sodius\help`.

Changes that Require User Action

This subsection documents the changes that require you to perform some actions when you upgrade to Version 4.2.

Static Relations (C++ and Java)

Rhapsody 4.2 introduces the ability to model static (class-wide) relations. To set a relation as static, you set its `CPP_` or `JAVA_CG::Relation::Static` property to `True`. The feature also supports reverse engineering and full roundtrip of static relations.

As part of the feature, two new properties were introduced to the containers' implementation metaclasses (`Fixed`, `BoundedOrdered`, and so on). Rhapsody in C++ or Rhapsody in J users who want to use the new static relations feature, and have their own relation implementation container properties (`OMContainers`) must add these properties to their user-defined containers.

The new properties are as follows:

- `CreateStatic`—Specifies container creation for the static relation. This property is used when the `CG::Relation::Containment` property is set to `Reference`.

  For example, the value of `OMContainers.BoundedOrdered.CreateStatic` is `"new OMList<$target*>"`.
- `InitStatic`—Specifies the initialization of the container in case of a static relation. This property is required only for Rhapsody in J.

  For example, the value of the `Java(1.2)Containers.StaticArray.InitStatic` property is `"new $target[$multiplicity]"`.

Automatic Glue Generations (Ada)

Version 4.2 includes the following enhancements to Rhapsody in Ada:

- Generation of the `main`
- Automatic run of the main event loop
- Support for parts within composite classes
- Auto-instantiation of links
- Object instantiation at any level

Rhapsody in Ada code generation was enhanced to automatically generate the entry point, as well as creation and run-time connection of instances.

To avoid the automatic generation of the entry point (for 4.1 models), set the `CG::Configuration::MainGenerationScheme` property to `UserInitializationOnly`.

OSE Support (C++)

In Rhapsody 4.2, OSE support was upgraded to OSE 4.5, with Diab 5.0.3.

The corresponding changes are as follows:

- New `#define` statements were added to `<Rhapsody install>/Share/LangCpp/osconfig/ose/omosconfig.h` under `#ifndef __DIAB:`
  - `__DISABLE_LONG_LONG`
  - `NEED_INLINE_IN_TEMPLATE`
  - `NEED_DELETE_OPERATOR_FOR_STATIC_ALLOC`
- A new `delete` operator was added to `OMTimeout`, wrapped in `#ifdef NEED_DELETE_OPERATOR_FOR_STATIC_ALLOC`.
- The archive log was removed from `oseev.sig`.
- The `PRIORITY_LOW` value was changed from 255 to 31.
- The `OSE` properties were modified.
- The `SFK` and `PPC` adapter framework makefiles were updated.

If you are using an older version of OSE or Diab, you should either upgrade your RTOS or make the appropriate changes to the 4.1 OSE framework-related files and properties.

QNX Adapter Message Queues

The QNX default message queue was changed from the POSIX message queue, which was designed for cross-process communication, to the Rhapsody "native" message queue (used in other adapters such as `Linux`, `Win32`, `pSOS`, and so on).

To use POSIX queues, rebuild the OXF libraries with the `OM_POSIX_QUEUES` flag set in the makefile `QNXCWoxf.mak` or `QNXoxf.mak`.

Animation Enhancements (C++)

New files were added to `<Rhapsody install>/Share/LangCpp/omcom` to support new animation messages. The new files are as follows:

- `AnimForeignMessage.cpp/h`
- `AnimNameValueData.cpp/h`
- `AnimOpReturn.cpp/h`

If you are using a custom adapter that does not take advantage of the `omcom .list` files, add the files to your `omcom` makefile.

Automatic Upgrades Performed by Rhapsody

This documents the changes that Rhapsody performs automatically when you upgrade to Version 4.2.

Changes in Generated Code

Rhapsody 4.2 includes the following general changes in the generated code:

- Parentheses are added to every auto-generated `if`, `for`, and `while` statement. This change was done to comply with commonly used "best practices" and as part of the Rhapsody in C conformance to the Motor Industry Software Reliability Association (MISRA®) standard.
- In calls to `schedTm()` (timeout scheduling related to the `tm(X)` instruction) in non-instrumented configurations, the last parameter is set to `NULL` (`null` in Rhapsody in J) instead of a string with the state name. This change was done as part of the constraint memory environment support.

C++-Specific Changes

Rhapsody in C++ includes the following changes:

- Spaces were added between the template declaration and the operation return type for template operations. For example, `template<class T>void f()` is replaced with `template <class T> void f()`.
- An additional argument was added to `DECLARE_MEMORY_ALLOCATOR()` when using memory pools. The additional argument is the initial size of the memory pool. This change was done as part of the support of memory pools for nested classes.
- By default, a pure-virtual destructor body is generated in the implementation (`.cpp`) file to comply with ANSI-C++. This is particularly true for «`Reactive Interface`» destructors.

C-Specific Changes

Rhapsody in C includes the following changes:

- In inline operations, the redundant backslash (“\”) was removed from the last line of the generated macro.
- Reactive classes `<state>_IN` operations are now constant (that is, the `me` parameter is passed as a `const pointer`).

Changes in Full Roundtrip (C++)

Version 4.2 includes the following changes to full roundtrip in Rhapsody in C++:

- Template operations are now supported.
- Static relations are now supported.
- Roundtrip does not set the `Inline` property for template operations and functions.

Additional Information

This subsection contains additional information.

Adapters

Version 4.2 includes the following changes to adapters:

- `pSOS X86` support was terminated and the adapter was removed (C and C++).
- The `IntegrityESTL` (EC++ with Templates) adapter was added, based on INTEGRITY 4.0.4 (C++).
- OSE adapters were upgraded to OSE 4.5 (Soft Kernel, using VC++ 6.0 SP-3 compiler and PPC using Diab 5.0.3 compiler).

Rhapsody in C Framework

Version 4.2 includes the following changes to the C framework:

- The `RiCHeap` implementation was modified to prevent errors when the heap is empty and `trim()` is called.
- A potential mutual exclusion problem was corrected in the `RiCTimer post()` operation.

Animation Enhancements (C++)

Version 4.2 includes the following changes in animation:

- The overloaded `out2String()` methods were added to `om2str.cpp/h` (`<Rhapsody install>/Share/LangCpp/omcom`).
- New animation messages were added (`<Rhapsody install>/Share/omcom/omnote.h`).
- A new macro, `OM_RETURN()`, was added to `aommacros.h`.

GHS MULTI Build Files Generation (C++)

For `MultiWin32`, the adapter search path is now taken from the `CPP_CG::<Environment>::AdaptorSearchPath` property instead of being based on the environment name, if the property exists and has a significant (not empty) value.

ESTL Support (C++)

The framework was modified to support Embedded C++ with templates (ESTL). In addition, a predefined ESTL environment was added for INTEGRITY ESTL by Green Hills® Software, Inc.

The corresponding changes are as follows:

- Multiple inheritance was replaced by delegation.
  - In `IntegrityHandleCloser` and `NTHandleCloser`, inheritance from `OMThread` was replaced with aggregation; the aggregate name is `thread`.
  - In `AOMEventQueue`, inheritance from `OMEventQueue` was replaced with aggregation; the aggregate name is `omQueue`.
  - In `TOMClass`, inheritance from `TOMClassNameGiver` was replaced with aggregation; the aggregate name is `tomNameGiver`.
  - In `TOMThreadManager`, inheritance from `OMList<TOMThread *>` was replaced with aggregation; the aggregate name is `threadList`.
- The `OMThread` changes are as follows:
  - An `OMBoolean` attribute named `deletionAllowed` was added. It is used to delay the deletion of the `HandleCloser` classes since inheritance of these classes from `OMThread` was replaced with aggregation. The default attribute value is `True`, and can be modified using the `setDeletionAllowed()` method. The virtual method `allowDeleteInThreadsCleanup()` was modified to return this new attribute value, instead of `True`.
  - The `eventQueue` attribute pointer type was modified in animation to `AOMEventQueue` as a result of the replacement of the `AOMEventQueue` inheritance from `OMEventQueue` with aggregation.
- In `OMNotifier::notifyToError()`, `omcout` is used instead of `omcerr` under `#ifdef NO_STDERR`.
- There is a new version of `OMREGISTER_REACTIVE_CLASS` under `#ifdef ESTL (aommacros.h)`.
- In `<Rhapsody install>/Share/LangCpp/osconfig/integrity/omosconfig.h`, additional `#define` statements were added under `#ifdef` ESTL.
- In order to generate instrumented code and activate checks that verify ESTL compliance, you should set the `CPP_CG::<Environment>::ESTLCompliance` property to `True`.

Upgrading to Version 4.1

The changes in version 4.1 of Rhapsody are listed below.

Changes that Require User Action

This subsection documents the changes that require you to perform some actions when you upgrade to Version 4.1.

Compiler and RTOS Changes

In Version 4.1, the framework files were cleaned up so there are two sets of files:

- **Generic files**

  The generic files contain generic #ifdef statements whose values are set in the compiler- and RTOS-specific files.

- **Compiler- and RTOS-specific files**

  These files include RTOS adapter files and a new adapter configuration file. See **C++ Framework Changes** and **C Framework Changes** for detailed information about the new configuration file.

  Note: Although there are two sets of files, some of the files co-exist in the same directory.

This change increases portability. If you are using a custom adapter (not an “out-of-the-box” adapter) or a custom environment (for code generation), you must perform some upgrade actions.

There are two ways to upgrade a custom adapter:

- Create an RTOS configuration file for your adapter. If your adapter is based on one of the out-of-the box adapters, you can reuse its RTOS configuration file.

  It is recommended that you use this method.

- Merge the compiler- and RTOS-specific `#ifdef` statements back into the framework code (as required by previous versions of Rhapsody). If you use this method, take into account any compiler macros (such as `__DIAB`) on which your code relies.

Upgrading Your Custom Environment

To upgrade your custom environment, you must do one of the following:

1. Upgrade the code generation environment by setting the property `<lang>_CG.<Environment>.AdaptorSearchPath` to the path to the RTOS configuration file directory. For example, `"$(OMROOT)/LangCpp/osconfig/VxWorks"`.

2. Upgrade the framework adapter by creating your own configuration file and defining the relevant flags. Update the generated makefiles by adding the new search path.

C++ Framework Changes

For each RTOS supported in Version 4.1, there is a corresponding file (`omosconfig.h`) that contains RTOS-specific definitions (such as `#define OM_NO_OS_ASSERT`), include statements, and macros. The file is located in a `<Rhapsody>\Share\LangCpp\osconfig\<RTOS>` directory, where <*RTOS*> is the name of the adapter (for example, `INTEGRITY`, `Linux`, `Nucleus`, and so on).

Note

Microsoft, MicrosoftDLL, MSStandardLibrary, MicrosoftWinCE and Borland use the file `Win32/omosconfig.h`.

The location of the `omosconfig.h` file was added to the search path of the adapter makefiles and generated makefiles. See the abstract operating system definition file (`<Rhapsody>\Share\LangCpp\oxf\os.h`) for the list of the generic `#define` statements added to support adapter portability.

The following table lists the files under `<Rhapsody>\Share\LangCpp` that have modified `#ifdef` statements because of the cleanup.

Subdirectory	File
`aom`	• `aomdisp.cpp` • `aommacro.h` • `aommsg.h` • `amothread.h`
`omcom`	• `om2str.h` • `omexp.cpp/h` • `omnote.cpp` • `omsdata.cpp/h`

Subdirectory	File
oxf	• event.cpp • MemAlloc.h • omabscon.h • ommemorymanager.h • omputput.cpp • omprotected.h • omstring.h • omunicode.h • OMValueCompare.h • os.h • oxf.cpp • rawtypes.h • rp_framework_dll_definition.h
tom	• tomC.cpp/h • tominst.cpp • tommask.h • tomproxy.cpp • tomstep.cpp • tomsys.cpp/h

C Framework Changes

For each RTOS supported in Version 4.1, there is a corresponding file (`ricosconfig.h`) that contains RTOS-specific definitions (such as `#define RIC_OS_MUTEX_LOCK_AS_OPERATION`), include statements, and macros. In addition, the `RiCOS.h` file was split and moved from `<Rhapsody>\Share\LangC\oxf` so each RTOS directory has its own copy.

For each RTOS, the files are located in a `<Rhapsody>\Share\LangC\osconfig\<RTOS>` directory, where *<RTOS>* is the name of the adapter (for example, `Nucleus`, `pSOS`, and so on). The location of the `ricosconfig.h` file was added to the search path of the adapter makefiles and generated makefiles. See the abstract operating system definition file (`<Rhapsody>\Share\LangC\oxf\RiCOSWrap.h`) for the list of the generic `#define` statements added to support adapter portability.

The following table lists the files under `<Rhapsody>\Share\LangC` that have modified `#ifdef` statements because of the cleanup.

Subdirectory	File
`aom`	• `aomcalls.c` • `aomeque.h` • `aomstep.h` • `aomthrd.c/h`
`omcom`	• `om2str.h` • `ommask.h` • `omnote.c` • `omsdata.c`
`oxf`	• `RiCOS.h` – Split and moved to the `osconfig\<RTOS>` directories. • `RiCOSWrap.h` • `RiCOxf.c` • `RiCProtected.h` • `RiCString.c` • `RiCTypes.h`

Framework File Changes

Several files were added to and removed from the C and C++ framework in Version 4.1. If you are using custom adapters, you must update your makefiles accordingly.

Note

The Rhapsody in J framework file structure was not modified.

Rhapsody in C++ File Changes

For Rhapsody in C++, `.list` files were added to all the libraries. These files contain the list of common files that should be compiled as part of all the adapters.
It is recommended that you use these files to increase the tolerance to file changes.

The following table lists the new, common source files located under `<Rhapsody>\Share\LangCpp`.

Subdirectory	File Name
`aom`	• `AOMMessageSender.cpp/h` • `aomoperation.cpp/h`
`omcom`	• `AnimDebuggerBreakPoint.cpp/h` • `AnimOpCallReply.cpp/h` • `AnimOpCallRequest.cpp/h` • `AnimOperationData.cpp/h` • `AnimRegisterOperations.cpp/h` • `AnimStringOrPointerField.cpp/h`
`osconfig`	`*/*`
`oxf`	`omiotypes.h`
`tom`	`tomoperation.cpp/h`
`WebComponents`	• `StaticClassElementsAdapters.cpp` • `TemplatedAdapters.cpp` • `WebComponentsTypes.cpp`

Rhapsody in C File Changes

The following table lists the new, common source files located under `<Rhapsody>\Share\LangC`.

Subdirectory	File Name
`osconfig`	`*/*`
`WebComponents`	• `StaticCharAttrWebAdapter.c/h` • `StaticCStrAttrWebAdapter.c/h` • `StaticDoubleAttrWebAdapter.c/h` • `StaticEventReceptionWebAdapter.c/h` • `StaticIntAttrWebAdapter.c/h` • `StaticLongAttrWebAdapter.c/h` • `StaticOperationWebAdapter.c/h` • `StaticRiCBooleanAttrWebAdapter.c/h` • `StaticRiCStringAttrWebAdapter.c/h` • `StaticShortAttrWebAdapter.c/h` • `StaticUCharAttrWebAdapter.c/h` • `StaticUIntAttrWebAdapter.c/h` • `StaticULongAttrWebAdapter.c/h` • `StaticUShortAttrWebAdapter.c/h` • `StaticWebAdapters.c/h`

The following table lists the files that were removed from `<Rhapsody>\Share\LangC`.

Subdirectory	File Name
`omcom`	`omexp.c` (was an empty file)
`oxf`	`RiCOS.h` (moved to the `osconfig\*` directories)

Default Directories for Specification and Implementation Files (C and C++)

This feature enables you to specify different directories for the specification (`.h`) and implementation (`.cpp`) files generated by Rhapsody. You must perform upgrade actions if you want to use this feature and you have a custom environment, or if the following properties are overridden in your model:

- `<lang>_CG::<Environment>::CompileSwitches`
- `<lang>_CG::<Environment>::MakeFileContent`

To upgrade, do the following:

1. Add `"<include qualifier> $OMDefaultSpecificationDirectory"` to the `<lang>_CG.<Environment>.CompileSwitches` property.
2. Replace the main file source name `"$(TARGET_MAIN)$(CPP_EXT)"`) with `"$OMMainImplementationFile"` in the `<lang>_CG.<Environment>.MakeFileContent` property.

Model Checking

- In Rhapsody 4.1, the “Dangling transition” check was modified from Warning to Error. This change was made to prevent erroneous run-time behavior. As a result, if you have a statechart that has dangling transitions, you must fix the errors before being able to generate code. To correct the errors, delete the dangling transitions and redraw them. If the transitions are going into or coming from a diagram connector, delete and re-create the diagram connectors.
- Now, statechart checks apply to the classifiers’ activity diagrams as well. As a result, statechart error checks will affect activity diagrams and prevent code generation. However, these checks do not apply to:
  - Activity diagrams of operations
  - Activity diagrams in analysis mode
  - Activity diagrams and statecharts whose owner class `CG::Class::ImplementStatechart` property is set to `False`
  - Activity diagrams and statecharts owned by a use case or non-generated actor

  The statechart checks are as follows:
  - Reference to unresolved statechart
  - Attribute named the same as a state
  - State named the same as its own class, super class or related class
  - Dangling transition

- Default transition not targeted to its state's substate
- Event and generated state in a class have conflicting names
- Implement statechart property differs for derived and base class
- Or state with no default state
- Join from non-orthogonal states
- Fork to non-orthogonal states
- Reference to unresolved statechart
- Reference to unresolved stereotype
- Static reaction without guard or trigger
- Reactive interface with a statechart or an activity diagram; code cannot be generated

Rhapsody COM API Changes

This subsection describes the changes made in Rhapsody COM API for Version 4.1. To use the modified API with your Rhapsody COM clients (such as VB or VBA), you must make these changes.

New Interfaces

Two new interfaces were added to the API:

- `IRPInstance`—Represents a classifier instance in the model
- `IRPLink`—Represents a link between two instances in a relation

Name Changes

Version 4.1 of the COM API includes the following name changes:

- The `nestedComponenets` property of the `IRPComponent` interface was renamed to `nestedComponents` to correct the spelling error.
- The `metaclass` property of `IRPComponentInstance` was renamed from `Instance` to `ComponentInstance`.

  The `Instance` value is now used as the metaclass of the `IRPInstance` interface.

Behavior Changes

The behavior of the `IRPApplication` methods `getSelectedElement()` and `getListOfSelectedElements()` was modified to support the new interfaces and collaboration model for sequence diagrams:

- When an instance is selected in the context of an object model diagram (OMD), the methods will return `IRPInstance` instead of `IRPClass` or `IRPActor`.
- When a link is selected in the context of an OMD, the methods will return `IRPLink` instead of `IRPRelation`.
- When an instance is selected in the context of a sequence diagram, the methods will return `IRPClassifierRole` instead of `IRPClass`.
- When a message is selected in the context of a sequence diagram, the methods will return `IRPMessage` instead of `IRPInterfaceItem`.

DiffMerge Changes

This subsection describes the changes to the DiffMerge utility that require user actions:

- The metaclass `General::DiffMerge` and its properties (`MergeOutput`, `DiffInvocation`, and `DiffMergeInvocation`) were removed in Version 4.1 of Rhapsody. Therefore, if you previously overrode those properties in your `site.prp` file, Rhapsody will ignore them (they will have no effect) unless you move them under `DiffMerge::TextDiffMerge`.

  Rhapsody searches for the properties for the invocation of the external, textual DiffMerge tool in the following order:

1. If you overrode the properties using the **View > Preferences**, Rhapsody looks in the `DiffMerge`.ini file.
2. Rhapsody searches through the current configuration management tool metaclass.
3. Rhapsody searches under `DiffMerge::TextDiffMerge` in the properties file.

- When you upgrade from one version of Rhapsody to a higher one and compare an old repository file with a new one (without changing either file), the DiffMerge utility might detect differences due to changes in the Rhapsody repository.

  For example, if you compare a version 4.0 class with a 4.1 class, DiffMerge might show differences resulting from a change in the repository's main diagram link. To eliminate the problem, open the old repository in the new version of Rhapsody, and save. This ensures that the saved unit complies with the new repository structure.

Features that Are Disabled on Load

This subsection describes the new features that are disabled on load of pre-4.1 models for backwards compatibility. Each topic includes a description on how the feature is disabled so you can enable it in your model.

Ignore Code in Prolog/Epilog Properties on Roundtrip (C++)

This feature addresses a full roundtrip problem where Rhapsody attempted to roundtrip code that was added via the prolog/epilog properties, which could result in unwanted elements being added to the model.

This feature wraps the prolog/epilog properties with ignore annotations, instructing the roundtrip tool to not add these parts back to the model.

The feature is disabled by setting all the `CPP_CG::<Metaclass>::MarkPrologEpilogInAnnotations` properties to `None` at the project level (for each metaclass that contains the prolog/epilog properties).

Robust Type Instrumentation (C and C++)

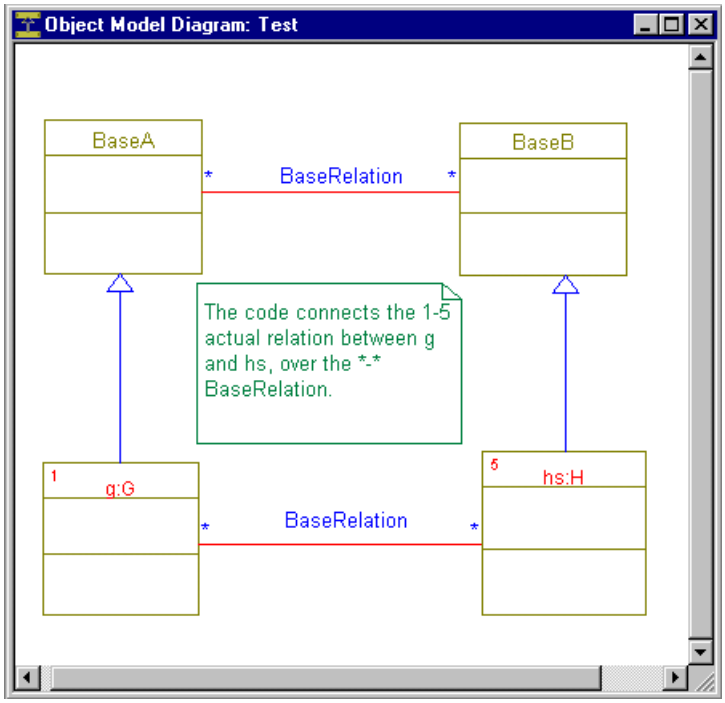
This feature addresses compilation errors in the generated code due to incorrect instrumentations of types. The feature is disabled to support users who manually defined serialize and deserialize operations to resolve the errors.

The feature is disabled by setting the `CG::Type::Animate` property to `Force` at the project level, and for any type where the property was overridden to `True`.

Instance-Based Linking

This feature causes Rhapsody to connect relations based on the instance multiplicity instead of the relation multiplicity, which enables you to connect instances based on relations with a multiplicity of “*”.

Consider the following example:



The feature is disabled to prevent creation of new run-time connections that are not expected in the existing models. It is disabled by setting the `CG::Relation::InstanceBasedLinking` property to `False` at the project level. The feature is *not* disabled if the property is already overridden.

Reflect Data Members in Reverse Engineering

This feature imports attributes as data members (thereby separating the data member from the accessor and mutator, as well as disabling the generation of the getters and setters).

The feature is disabled in configurations that were used for reverse engineering to maintain existing behavior. It is disabled by setting the property `<lang>_ReverseEngineering::ImplementationTrait::ReflectDataMembers` to `None` on configurations that were used for reverse engineering.

Advanced Webify Toolkit Settings

This feature enables you to fine-tune the Web server settings on a per-configuration basis (using the configuration's Settings tab). This feature was added to resolve limitations related to per-component settings done by the `webconfig.c` file.

The feature is disabled for components associated with the `webconfig.c` file.

Analysis Sequence Diagrams

Using this feature, you can create sequence diagrams with instances and messages that are not realized by model elements.

The feature is disabled to maintain the pre-4.1 behavior of sequence diagram modeling. It is disabled by setting the property `SequenceDiagram::General::ClassCentricMode` to `True` at the project level.

Property Changes

This subsection documents the property changes for Version 4.1. Note that the properties are upgraded automatically—they do not require any action on your part.

Renamed Properties

The following properties were renamed in Version 4.1 to clarify their roles:

- `CG::Type::InstrumentationFunctionName` was renamed (and moved) to `<lang>_CG::Type::AnimSerializeOperation`.
- The `MscGe` subject was renamed to `SequenceDiagram`.
- `General::Graphics::ScaleToFitExportedDiagram` was renamed to `General::Graphics::ExportedDiagramScale`.
- `MscGe::General::display_formals` was renamed to `SequenceDiagram::General::ShowArguments`.

- `<lang>_ReverseEngineering::ImplementationTrait::IgnoreIncludes` was renamed to `<lang>_ReverseEngineering::ImplementationTrait::AnalyzeIncludeFiles`.

Moved Properties

The following properties were moved in Version 4.1:

- `PackageEventIdRange` moved from `CG::Component` to `CG::Package` to increase the flexibility and user control over the events ID range reserved for a given package.
- `<lang>_CG::Type::Animate` moved to `CG::Type::Animate` as part of the robust type instrumentation functionality.
- `ImplementationEpilog`, `ImplementationProlog`, `SpecificationEpilog`, and `SpecificationProlog` moved from `CG::Dependency` to `<lang>_CG::Dependency` to match the location in other metaclasses.

Superseded Properties

The `CG::File::GenerateInMakefileOnly` property was replaced by `CG::File::AddToMakefile`. The new property enables you to control the generation of a file separate from the generation of the makefile, so you can handle a file that contains only text as a source file.

Properties Deleted from the Factory File

The following properties were removed from the factory properties file in Version 4.1:

- `CG::Attribute::InstrumentationFunctionName` was removed because of redundancy. You can specify a serialization operation for a specific argument by overriding the `CG::Type::AnimSerializeOperation` for the given attribute.
- `CG::Component::PackageEventBaseIdAlgorithm` was removed. If you overrode this property, the override still affects the generated code and the property will move automatically to the `CG::Package` metaclass.
- `General::Graphics::FlickerFree` was removed because it had no effect.
- `General::Model::SearchPath` was removed because it had no effect.

Changed Properties

In Version 4.0, the `CG::Class::CreateImplicitDependencies` property did not affect the generated code (therefore, the `#include` was generated even when the property was set to

`False`). This problem was corrected in Version 4.1, so type declarations are no longer analyzed when the property is set to `False`—and, as a result, the `#include` is not generated.

Additional Information

This subsection contains additional information.

Enhanced C++ Standard Library (STL) Support

This feature was added as part of the support for the Microsoft Visual Studio.NET environment, but it can be useful for other RTOS adapters as well. This feature enables you to:

- Compile the framework to use the standard library streams without the command `use namespace std;` by compiling the framework with the `OM_STL` compilation flag.
- Use generic stream types mapped to either the vendor streams or standard library streams based on the `OM_STL` compilation flag. This is done by using `om<stream element>` instead of `<stream element>`. For example, use `omcout` instead of `cout`. The `omstreams` are defined in `<Rhapsody>/Share/LangCpp/oxf/omiotypes.h`.

Reverse Engineering of #include Statements Not Found by the Parser (C and C++)

This feature imports the `#include` statements that were not found by the parser into the `SpecIncludes` and `ImpIncludes` properties.

You can disable the feature by setting the property `<lang>_ReverseEngineering::ImplementationTrait::CreateDependencies` to `DependenciesOnly`.

C++ Framework Changes

In addition to the changes described in **C++ Framework Changes**, the Rhapsody 4.1 C++ framework includes the following changes:

- The attribute `count_ (unsigned long)` was added to `OMList` and `OMMap` to improve the performance of `getCount()` from `O(N)` to `O(1)`.
- Two virtual methods were added to `OMOSThread` to support RTOSes that require socket initialization and cleanup for each task (`GHS INTEGRITY`). The methods are not pure virtual; therefore, there is no need to implement them on other adapters. The new methods are as follows:
  - `virtual void InitCommunicationLayer() {}`
  - `virtual void cleanupCommunicationLayer() {}`
- Two changes were made to the INTEGRITY adapter:
  - The task (`_omCloseHandle`) is responsible for cleanup after terminating tasks. The cleanup is required to prevent RTOS resource leaks.
  - In animation, only two predefined tasks initialize the TCP/IP layer. User tasks no longer send animation messages directly.

Modeling Changes

The following sections describe modeling behavior changes to consider when upgrading to Rhapsody 4.1.

Inherited Statechart Coloring

- Overridden entry and exit actions cause the state to change coloring to None inherited.
- An inherited And state previously colored as None inherited will be colored with inheritance coloring due to the addition of orthogonal states.
- Some transitions will be colored as overridden even though in previous versions they were shown as not overridden. This occurs because the color reflects the internal state of the transition that was overridden or modified in the derived statechart.

Sequence Diagrams

- In instance line text, the format `<Name>` that referred to `<Class Name>` now refers to `<Classifier Role Name>`. Old sequence diagrams with that name format will change to `:<Class Name>`.
- When a Rhapsody 4.0.1 model has a sequence diagram that contains an instance line that represents a static instance a path name of the form “`<package_name>::<instance_name>:<class_name>`”, you might need to save and reopen the model in order to have Rhapsody display the instance name properly in the sequence diagram.

- When set to `True`, the property `SequenceDiagram::General::ClassCentricMode` enables you to create a class by typing `<Class Name>`, which in turn changes the label on the instance line to `:<Class Name>`.
- When you delete an operation or class in the browser, the only way to remove them automatically from the sequence diagram is to set the property `SequenceDiagram::General::CleanupRealized` to `True`.
- To realize messages automatically when you rename them, set the property `SequenceDiagram::General::RealizeMessages` to `True`.
- By default, names of instance lines are enclosed in a bounding box. To remove these boxes, set the property `AddBoxesAroundInstanceNames` (under `SequenceDiagram::InstanceLine`) to `False`.

Configuration Management Changes

The Check Out Branch button was removed from the CM and List Archive dialog boxes. To reactivate it, add the following property:

```
ConfigurationManagement::General::EnableCheckoutBranch ="True"
```

By default, this property is not available in the `.prp` file.

Code Generation Changes

- There was a defect that made the code generator ignore new lines added to the end of user code. As a result of the fix made in Rhapsody 4.1, some new lines might appear in your code after the first generation.
- In large models, event IDs might change if you did not explicitly set them.
- When an active or reactive class (`A`) inherits from another active or reactive class (`B`), the call to `start()` is removed from `startBehavior()` operation of class `A`. This is done because `B::startBehavior()` is called from `A::startBehavior()`—therefore, the call to `start()` in `A::startBehavior()` is redundant.
- A fix in code generation remove redundant commas at the end of block statements. In Rhapsody 4.1, `"if (...) {...};"` is replaced with `"if (...) {...}"` in the automatically generated code.

Upgrading to Version 4.0.1 MRx

The changes in versions 4.0.1 MR*x* of Rhapsody are listed below.

Upgrading to Version 4.0.1 MR1

This subsection documents the changes to the framework and properties between Rhapsody 4.0 and Rhapsody 4.0.1 MR1. Note that when you upgrade to Version 4.0.1 MR1, no additional user modifications are necessary.

Properties

The following properties have been added to this release of Rhapsody:

- `CG::Argument::Animate`—Enables or disables instrumentation of a specific argument.
- `ConfigurationManagement::ClearCase/PVCS/SCC/SourceIntegrity`
  - `DiffInvocation`—Specifies the command to invoke the external, textual DiffMerge tool
  - `DiffMergeInvocation`—Specifies the command to invoke the external, textual DiffMerge tool
  - `MergeOutput`—Specifies the file that will hold the results of a merge operation

See the *Properties Reference Guide* for more information.

Rhapsody in C++-Specific Changes

Version 4.0.1 MR1 includes the following changes to the C++ framework:

- The misspelled `OMUAbstructContainer` was changed to `OMUAbstractContainer`; a typedef was added for backward compatibility.

- In `OMGuard`, a parameter (with a default value) was added to the constructor for animation support.
- The QNX and Nucleus (C and C++) adapters were upgraded.
- In `OMTimerManager`, simulated time support was enhanced by the following changes:
  - `void incNonIdleThreadCounter()`—This method increases the `nonIdleThreadCounter` attribute.
  - `void decNonIdleThreadCounter()`—This method decreases the `nonIdleThreadCounter` attribute.
  - `long nonIdleThreadCounter`—Is a counter of the non-idle threads in the system. This is used in simulated time to determine whether a timer tick should be issued.

Properties

The following properties have been added:

- In the property `STLContainers::Qualified::Remove`, the `OMValueCompare` usage was modified from `"OMValueCompare<$keyType,$target*>($item)"` to `"OMValueCompare<const $keyType,$target*>($item)"` to conform to the C++ standard.
- There are three new properties under `CORBA::<ORB>`:
  - `InitializeORB`—Specifies the ORB initialization routines
  - `InitialInstance`—Specifies any additional initial instance routines required by an ORB
  - `ClientMainLineTemplate`—Enables you to add code to the `main` function of a CORBA client

See the *Properties Reference Guide* for more information about these new properties.

Rhapsody in J-Specific Changes

The `isIn()` methods are generated to all states, regardless of whether the states are inherited. This behavior is necessary because the derived class has no access to the super inner class that implements the `isIn()`.

Upgrading to Version 4.0.1 MR2

When you upgrade to Version 4.0.1 MR2, no additional user modifications are necessary.

Rhapsody in C-Specific Changes

In `RiCMap`, `RiCMapKeyIsGrater` was renamed to `RiCMapKeyIsGreater` to correct the spelling error. The old name (`RiCMapKeyIsGrater`) is still available via `typedef`.

Upgrading to Version 4.0

The changes in version 4.0 of Rhapsody are listed below.

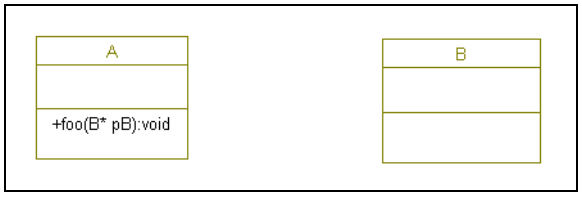
Changes that Require Model Modifications

This subsection documents the changes that require you to modify your model.

Generation of Implicit Dependencies

Rhapsody tries to understand the user model and adds missing dependencies in the code based on type declarations.

Consider the following model:



Rhapsody will add a dependency (`#include`) from `A` to `B`, due to the use of `B*` in `A::foo()`.

This is not new behavior. However, in previous versions of Rhapsody, if the model has two classes named `B` (say, in different packages), Rhapsody would have created a dependency to one of the `B` classes randomly. Rhapsody 4.0 corrects this behavior and does not create any dependency—letting you figure out the correct dependency.

This change of behavior might result in compilation errors in models that relied on the implicit dependencies—and got away with it.

Note

To make Rhapsody stop generating implicit dependencies, set the `CG::Type::GenerateDeclarationDependency` property to `False` at the project level.

Calling an Overridden initRelations() Operation

Rhapsody 4.0 calls user-overridden `initRelations()` operations. To avoid backward compatibility issues, this ability is disabled in pre-4.0 models by setting the `CG::Class/Package::CallUserInitRelations` property to `False` at the project level.

However, because this property was introduced in Rhapsody version 3.0.1 without this backward compatibility, clients that upgrade from version 3.0.1 should consider removing the override. Clients that upgrade from version 3.0 will get the same behavior.

Relation Properties

Rhapsody 4.0 relation properties were improved to give you more flexibility and control over custom relation implementation. This change has two side effects, as described in the following subsections.

Keywords of Relations' Signature Properties

The `$target` keyword in the relation signature properties is now resolved correctly to the relation target class name. Therefore, you should replace the keyword with `$cname` in the site properties (`site.prp`) file. Note that Rhapsody does this automatically for properties that are overridden in the model).

The affected properties (under `<lang>_CG/CG::Relation`) are `Add`, `Clear`, `CreateComponent`, `DeleteComponent`, `Find`, `Get`, `GetAt`, `GetEnd`, `GetKey`, `Remove`, `RemoveAt`, `RemoveKey`, and `Set`.

For example:

```
// If your site file has this...
Subject CG
   Metaclass Relation
          Property Get String "$targetGet"
   end
end

// You must replace it with this.

Subject CG
   Metaclass Relation
          Property Get String "$cnameGet"
```

```
    end
end
```

Keywords Used in the Set Property

In the `Set` property of the `EmbeddedScalar` and `Scalar` metaclasses under the relations implementation properties, the `$target` keyword is not resolved correctly as the relation target type. Therefore, it should be replaced with `$item`—the set method argument name.

The keywords were replaced in the factory properties, but if either of the following cases applies to your model, you must upgrade the properties manually:

- You are using custom relation implementation properties, defined at the site properties level.
- The `Set` property is overridden in the model.

For example:

```
// Pre-4.0 property value
Subject MyContainers
    Metaclass Scalar
           Property Set String "$cname = $target"
    end
end


// Should be replaced with
Subject MyContainers
    Metaclass Scalar
           Property Set String "$cname = $item"
    end
end
```

Framework Event Consumption API Changes (C and C++)

The Rhapsody 4.0 framework was enhanced to allow you to handle unconsumed events and triggered operations. See **Handling Unconsumed Events and Triggered Operations** for the description of the additional API. To support this new functionality, the reactive `consumeEvent()` signature was modified to return the event consumption status.

If your model customizes the event consumption by overriding `consumeEvent()`, you must modify the return type of the overridden method, as follows:

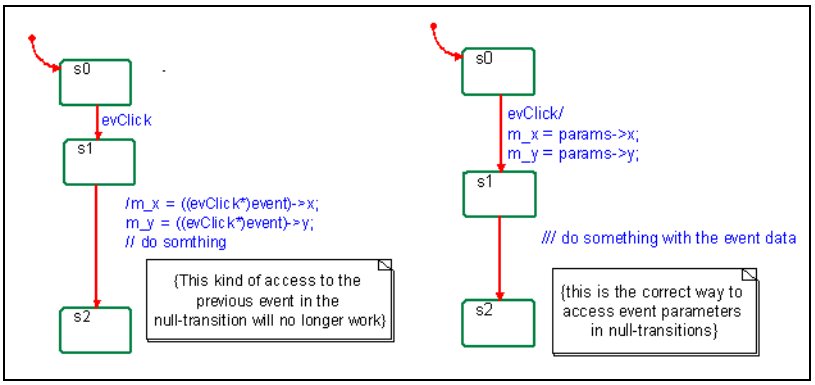
- For Rhapsody in C++, modify the `consumeEvent()` return type to the enum `OMReactive::TakeEventStatus`.
- For Rhapsody in C, modify the `consumeEvent()` return type to `RiCTakeEventStatus`.

Event Handling in Null Transitions (C and C++)

Rhapsody 4.0 introduces generic handling of derived events. See **Generic Handling of Derived Events** for more information.

A side-effect of this change is that the event consumed before a null transition cannot be accessed from the null transition. If your model uses this kind of access, you must modify the model and store the event data in a user-defined attribute.

The following example shows such a statechart.



Guarded Class Implementation (C++)

In order to resolve several issues related to inheritance from protected classes, Rhapsody 4.0 changes the implementation of guarded classes. The new guarded classes' implementation replaces the inheritance from `OMProtected` to aggregation.

As a result of this change, the polymorphism of both active classes and guarded classes to `OMProtected` no longer exists—that is, you can no longer pass an active or guarded class to an interface that expects an `OMProtected` class. If your model relies on this polymorphism, use the new interface for active/guarded classes, `getGuard()`.

The change from inheritance to aggregation involves the following addition to the guarded class API (by adding the `OMDECLARE_GUARDED` macro to your class declaration):

```
public:
    inline void lock() const;
    inline void unlock() const;
    inline const OMProtected& getGuard() const;

private:
    OMProtected m_omGuard;
```

The same API was added to `OMThread`.

This additional API means that calling `lock()` and `unlock()` on a guarded class will still work.

Consider the following code:

```
#define GUARD_RESOURCE(guardedResource) OMGuard
    guard(guardedResource)
```

Replace the macro implementation to the one shown in the following code:

```
#define GUARD_RESOURCE(guardedResource) \
    OMGuard guard((guardedResource).getGuard())
```

Configuration Management of the RPY File in SCC Mode

This change affects the backward compatibility of Rhapsody.

You must complete the following upgrade steps for existing Rhapsody projects that have already been checked in to an SCC archive:

1. Create a directory/folder in the CM tool with the name of the directory that holds the `.rpy` file.
2. In the CM tool, copy the `XXX_rpy` directory to the directory you just created.
3. Disconnect from the existing archive.
4. Change the value of the property `ConfigurationManagement::SCC::SupportTreeRepository` to an empty string.
5. Reconnect to the archive.

Automatic Upgrades Done by Rhapsody

Rhapsody 4.0 will upgrade your model the first time it is loaded. The upgrade of the model is done by setting properties at the project level, in order to conform to the existing (pre-4.0) code.

Note

Unless explicitly stated otherwise, the upgrade is done *only* for pre-4.0 models, not for models saved in the beta version of 4.0.

Clean Default Values for Attributes (C and C++)

Due to a defect in previous versions of Rhapsody, the default value field for attributes sometimes contained invalid values. Because Rhapsody 4.0 generates initialization code based on the default value of attributes, the value is set to an empty string when you first load the model in Rhapsody 4.0.

For Rhapsody in C++, modifying the model does not affect static attributes.

Note that for any attribute whose default value is cleaned, a message is added to the load log file.

Smart Generation of Package Code

Rhapsody 4.0 generates the package code only when it is meaningful (when the package contain significant elements such as instances, types, functions, and so on.). However, to avoid upgrade issues, this will not affect pre-4.0 models.

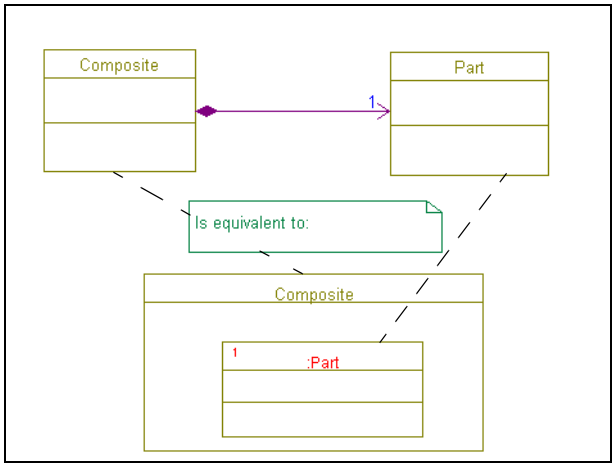
To enable this feature, uncheck the override on the `CG::Package::GeneratePackageCode` property (at the project level).

Generation of Filled-Diamond Relations

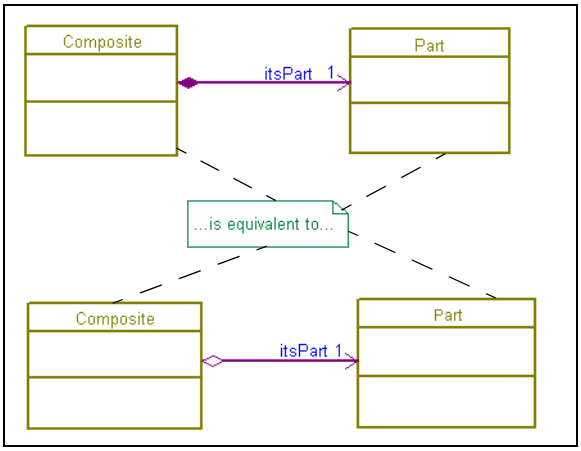
Rhapsody 4.0 implements filled-diamond relations. When creating a filled-diamond relation, you can select whether it will be implemented as a composite or an aggregation by setting the property `CG::Relation::FilledDiamondScheme` to `Composition` (the default value) or `Aggregation`.

Because the preview versions (beta and prerelease) of Rhapsody 4.0 created only aggregation code, the property is set to `Aggregation` when loading models created with the preview versions.

When implementing a composite, Rhapsody generates the same code as it would generate for a composite relation. For example:



When implementing a composite, Rhapsody generates the same code as it would generate for aggregation relations. For example:


Composite
itsPart 1
Part
...is equivalent to...
Composite
itsPart 1
Part

Relation Properties

Rhapsody 4.0 relation properties were improved to give you more flexibility and control over custom relation implementations. As a result, Rhapsody will make the following changes in your model:

- **Keywords of relations' signature properties**

  The `$target` keyword in the relation signature properties is now resolved correctly to the relation target class name. Therefore, the keyword is replaced with `$cname` for any overridden property.

  The affected properties (under `<lang>_CG/CG::Relation`) are `Add`, `Clear`, `CreateComponent`, `DeleteComponent`, `Find`, `Get`, `GetAt`, `GetEnd`, `GetKey`, `Remove`, `RemoveAt`, `RemoveKey`, and `Set`.

- **Rhapsody in C++/Java GetAt default signature**

  The `GetAt` default signature was modified from `get<relation name>()` to `get<relation name>At`. This change was made to avoid possible collision between `Get` and `GetAt`.

  To override this change at the project level, set the `<lang>_CG::Relation::GetAt` property value to `"get$cname:c"`.

- **Change in the Type property name**

  The `Type` property name in the `EmbeddedScalar` and `Scalar` metaclasses (under the relations' implementation properties) was renamed to `CType`). However, if the `CType` property is not found, the `Type` property is used to generate the code.

Calling an Overridden initRelations() Operation

Rhapsody 4.0 calls user-overridden `initRelations()` operations. To avoid backward compatibility issues, this ability is disabled in pre-4.0 models by setting the `CG::Package/Class::CallUserInitRelations` property to `False` at the project level.

However, because this property was introduced in Rhapsody version 3.0.1 without this backward compatibility, clients that upgrade from version 3.0.1 should consider removing the override. Clients that upgrade from version 3.0 will get the same behavior.

Generalization (C++)

Rhapsody 4.0 introduces full support in modeling of generalization (inheritance). As a result, the properties `CPP_CG::Class::VirtualInherits/PrivateInherits` became obsolete. The property content is converted to the model elements, and their content is deleted.

The properties' content will remain only when Rhapsody cannot convert all the content to model elements.

Cleanup of the OXF Namespace (C++)

As part of Rhapsody 4.0 development, a major effort was done to clean up the OXF namespace. See **Global Namespace Cleanup** for details on the cleanup.

To support pre-4.0 users of the framework, a new file, `OMObsolete.h,` was added to the framework. This file enables user code to continue using the pre-4.0 framework API (generated code uses the new API).

The `OMObsolete.h` file is included automatically as a standard header.

To remove this include, remove the override on the `CPP_CG::Component::StandardHeaders` at the project level.

Generated Class Name for Packages (Java)

In Rhapsody 4.0, the default name of the class generated for a package is `<package name>_pkgClass`.

When loading pre-4.0 models that did not modify their default, it is set back to `<package name>` by overriding the property `Java_CG::Package::PackageClassNamePolicy` at the project level, and setting the property value to `Default`.

Changes in Property Names or Locations

This subsection documents the properties that were moved or renamed in the Rhapsody factory properties file (`factory.prp`). If the properties are overridden in your model, Rhapsody will automatically move or rename the properties.

If the properties were overridden in your site properties (`site.prp`), Rhapsody will detect the properties in their previous location and name—there is no need to modify the site file.

Note

If you overrode these properties in the site file (`site.prp`), Rhapsody will find the property without any modification to the file.

The affected properties are as follows:

- `CG::Configuration::AllowCollusionWithComponentName` was renamed `AllowCollisionWithComponentName`.
- `CG::Attribute::AnimateAttributes` was renamed `Animate`.

- `CG::Package::EventsBaseId` moved to the language-specific subjects (`<lang>_CG`).
- `<lang>_CG::<environment>::CPPCompileSwitches` was renamed `<lang>_CG::<environment>::CompileSwitches`.
- `CPP_CG::Class::VirtualInherits/PrivateInherits` were removed from the `factoryC++.prp` as part of Rhapsody 4.0 generalization functionality. Overridden properties values will be added to your model.

VariableInitializationFile Property

Rhapsody 4.0 recognizes a constant global variable when the declaration begins with `const`, and initializes the constant global variable in the package specification file.

Consider the following example:

The model contains a package `P` with a constant global variable `MAX_SIZE`, which is declared as `const int %s` and has an initial value of 50. Rhapsody will generate the following code in `P.h`:

```
const int MAX_SIZE(50);
```

This is new behavior for Rhapsody. To avoid backward compatibility issues, Rhapsody will override this behavior when loading pre-4.0 models, forcing the initialization to be in the package implementation file. This is done by overriding the `<lang>_CG::Attribute::VariableInitializationFile` property at the project level, and setting its value to `Implementation`.

The generated code for pre-4.0 models would be:

`P.h:`

```
extern const int MAX_SIZE;
```

`P.cpp:`

```
const int MAX_SIZE(50);
```

Changes in the Framework API

This subsection describes the changes to the framework API.

Rhapsody in C++-Specific OXF Changes

This documents Rhapsody in C++-specific changes to the OXF.

Global Namespace Cleanup

A special effort was made in Rhapsody 4.0 to clean up the usage of the global namespace. This entailed the following changes:

- All framework classes use the `OM` prefix. The only exception is the `OXF` class.
- Most of the global functions were moved to be static operations of the appropriate classes.
- A new file, `OMObsolete.h`, was added to the framework. This file holds a set of `typedef` and `#define` statements that redefine the old methods and type to the new names. You can include this file instead of changing your code. Note that the auto-generated code uses the new type/operations names.

The following table lists the name changes made to the OXF between version 3.0.1 and 4.0.

Rhapsody 3.0.1 OXF Element	Rhapsody 4.0 OXF Element
Classes and Types	
`State`	`OMState`
`AndState`	`OMAndState`
`ComponentState`	`OMComponentState`
`FinalState`	`OMFinalState`
`LeafState`	`OMLeafState`
`OrState`	`OMOrState`
`Timeout`	`OMTimeout`
`NullValue`	`OMNullValue`
Constants	
`containersNullBlock`	`OMContainersNullBlock`
`Null_id`	`OMEventNullId`
`Timeout_id`	`OMEventTimeoutId`
`CancelledEvent_id`	`OMEventCancelledEventId`
`AnyEvent_id`	`OMEventAnyEventId`
`OMStartBehavior_id`	`OMEventStartBehaviorId`
`OXFEndEvent_id`	`OMEventOXFEndEventId`
Global Variables	
`theSysTimer`	`OMThreadTimer::instance()`
Global Reactive Operations	
`isCurrentEvent()`	`OMReactive::IsCurrentEvent()`
`isValidOMReactive()`	`OMReactive::isValid()`
Notifications	
`NotifyToError()`	`OMNotifier::notifyToError()`

Rhapsody 3.0.1 OXF Element	Rhapsody 4.0 OXF Element
`OMNotifyToError()`	`OMNotifier::notifyToError()`
`NotifyToOutput()`	`OMNotifier::notifyToOutput()`
`OMNotifyToOutput()`	`OMNotifier::notifyToOutput()`
`NOTIFY_TO_ERROR()`	`OM_NOTIFY_TO_ERROR()`
`NOTIFY_TO_OUTPUT()`	`OM_NOTIFY_TO_OUTPUT()`
Framework Main Global Functions	
`OXFInit()`	`OXF::init()`
`OXFStart()`	`OXF::start()`
`OXFEnd()`	`OXF::end()`
`OXFDelay()`	`OXF::delay()`
OS Layer	
`theOSFactory()`	`OMOSFactory::instance()`
`OSOXFInitEpilog()`	`OMOS::initEpilog()`
`OSOXFEndProlog()`	`OMOS::endProlog()`
`OSOXFEndApplication()`	`OMOS::endApplication()`
`String Manipulation Global Functions`	
`strcmpNoCase()`	`OMStrcmpNoCase()`
`destructiveString2X()`	`OMDestructiveString2X()`

OMAbstractMemoryAllocator

An empty virtual destructor was added to support user-defined memory managers.

OMEvent

The inheritance from `AOMEvent` in instrumentation (animation and tracing) was removed as part of the support for partial animation (see **Partial Animation**).

A new virtual operation, `OMBoolean isTypeOf(short)`, was added to support generic handling of derived events (see **Generic Handling of Derived Events**).

OMTimeout

The class name was modified from `Timeout` to `OMTimeout` as part of the global namespace cleanup (see **Global Namespace Cleanup**).

The following changes were made to support partial animation:

- A new friend class, `OMFriendTimeout`, was added to animate the timeout class in instrumented mode. The `friend` class declaration is empty for non-instrumented mode.
- The following instrumentation methods were removed from the class interface:
  - `getEventClass()`
  - `cserialize()`

See **Partial Animation**for more information.

The `timeout` state attribute that was added only in instrumentation mode is now part of the `OMTimeout` interface in non-instrumented mode as well, and is set to `NULL`.

OMStartBehaviorEvent

The following changes were made to support partial animation:

- A new friend class, `OMFriendStartBehaviorEvent`, was added to animate the start behavior event class in instrumented mode. The `friend` class declaration is empty for non-instrumented mode.
- The following instrumentation methods were removed from the class interface:
  - `getEventClass()`
  - `cserialize()`

See **Partial Animation**for more information.

OMCollection

The default, initial collection size was reduced from 256 elements to 20 elements to reduce the default memory usage by the collection.

The `size` attribute moved to the base template class (`OMStaticArray`) to support the enhancement for user control over memory allocated by the framework (see **User Control over Framework Memory Management (C++)**).

OMMap

The private `remove()` method was renamed `removeItem()` to support the OSE soft-kernel.

OMMemoryManager

The `OMDELETE` macro declaration was modified from `OMDELETE(object)` to `OMDELETE(object, size)`. The new parameter was added to support the enhancement for user control over memory allocation (see **User Control over Framework Memory Management (C++)**).

The signatures of the `new` and `delete` operators declared in the macro `OM_DECLARE_FRAMEWORK_MEMORY_ALLOCATION_OPERATORS` were modified to support the

enhancement for user control over memory allocation (see **User Control over Framework Memory Management (C++)**). The signature changes are as follows:

- `void* operator new(size_t)` was modified to:

```
static void* operator new(size_t NEW_DUMMY_PARAM)
```

In this syntax, NEW_DUMMY_PARAM is set to "size_t=0" for every compiler except for DIAB (where it is set to nothing).

- `void* operator new[] (size_t)` was modified to:

```
static void* operator new[] (size_t size
    NEW_DUMMY_PARAM)
```

- `void operator delete (void * object)` was modified to:

```
static void operator delete (void * object,
    size_t size)
```

- `void operator delete[] (void * object)` was modified to:

```
static void operator delete[] (void * object,
    size_t size)
```

Protection against early destruction on application exit was added. This protection ensures that the internal memory manager singleton will be valid throughout the termination of the application. To achieve this guarantee, the following members were added to the class:

- `OMMemoryManager(bool)`—A constructor
- `~OMMemoryManagerManager()`—A destructor
- `static bool _singletonDestroyed`—A destruction indicator flag

OMNotifier

This is a new class that encapsulates the `notifyToError()` and `notifyToOutput()` operations. It was added as part of the global namespace cleanup (see **Global Namespace Cleanup**).

OMProtected

As part of the enhancements made for user control over framework memory allocation (see **User Control over Framework Memory Management (C++)**), there was a need to create a protected object, but postpone the creation of the RTOS mutex. The following operations were added to `OMProtected` interface to support this need:

- A new constructor was added to the class, `OMProtected(OMBoolean createMutex)`, to allow creation of the RTOS mutex later in the protected object lifetime, by calling the new `initializeMutex()` operation.
- A new operation, `void initializeMutex()`, was added to `OMProtected` to create the RTOS mutex (if it is not already created).

As part of the changes of the implementation of protected classes (see **Temporary Files**), a new operation, `const OMProtected& getGuard() const`, was added to allow handling of guarded classes and classes that inherit from `OMProtected` uniformly.

A new macro, `OMDECLARE_GUARDED`, is defined. This macro is used to aggregate `OMProtected` objects inside guarded classes instead of inheritance from `OMProtected`. The macro is defined as follows:

```
#define OMDECLARE_GUARDED                                  \
public:                                                    \
   inline void lock() const { m_omGuard.lock(); }          \
   inline void unlock() const { m_omGuard.unlock(); }      \
   inline const OMProtected& getGuard() const {            \
   return m_omGuard; }                                     \
private:                                                   \
   OMProtected m_omGuard;
```

OMGuard

The copy constructor and assignment operator of `OMGuard` were explicitly disabled to avoid erroneous unlock of the guarded object mutex.

The `GUARD_OPERATION` macro was modified to support the aggregation of `OMProtected` in guarded classes as well as inheritance from `OMProtected` by guarded classes.

OMReactive

The inheritance of `OMReactive` from `AOMInstance` in instrumented mode was removed as part of partial animation support (see **Partial Animation**).

The `consumeEvent()` return type was modified to `TakeEventStatus`. This change was made to support handling of unconsumed events and triggered operations (see **Handling Unconsumed Events and Triggered Operations**).

A new value was added to the `OMReactive::TakeEventStatus` enum to support handling of unconsumed events (see **Handling Unconsumed Events and Triggered Operations**). The new value is `OMTakeEventCompletedEventNotConsumed` and its integer value is 0. The existing values were increased by one.

The following operations were added to `OMReactive`:

- `OMBoolean IsCurrentEvent(short eventId) const`—Checks whether a given event id matches the currently processed event. This operation replaces the global function as part of the global namespace cleanup (see **Global Namespace Cleanup**).
- `const OMEvent* getCurrentEvent() const`—Gets the currently processed event.
- `void setEventGuard(const OMProtected&)`—Sets the event handling guard. This method is in addition to the method `void setEventGuard(const OMProtected*)`.

- `void handleEventNotConsumed(OMEvent*)`—This is a virtual method that is called when an event is not consumed by the reactive class. This method is part of the framework support for handling unconsumed events (see **Handling Unconsumed Events and Triggered Operations**).
- `void handleTONotConsumed(OMEvent*)`—This is a virtual method that is called when a triggered operation is not consumed by the reactive class. This method is part of the framework support for handling unconsumed triggered operations (see **Handling Unconsumed Events and Triggered Operations**).

The `eventNotConsumed` definition moved from `state.h` to `omreactive.h`, and was modified from `0` to `OMReactive::OMTakeEventCompletedEventNotConsumed` (which also equals `0`). This was done to support handling of unconsumed events (see **Handling Unconsumed Events and Triggered Operations**).

The `eventConsumed` definition moved from `state.h` to `omreactive.h`, and was modified from `1` to `OMReactive::OMTakeEventCompleted` (which is also equals `1`). This was done to support handling of unconsumed events (see **Handling Unconsumed Events and Triggered Operations**).

The `rootState_serializeStates()` method, which is declared only in instrumented mode, was modified from virtual to regular (non-virtual) operation as part of the support for partial animation.

OMStaticArray

A new attribute, `int size`, was added to the template class. This attribute moved from the derived `OMCollection` template class to support user control over framework memory allocation (see **User Control over Framework Memory Management (C++)**).

In addition, a `getSize()` operation was added.

OMString

All string operators that could be part of the class direct interface were moved into the class declaration.

OMThread

The inheritance from `OMProtected` was replaced with aggregation. As a result, the following were added to the `OMThread` interface:

- `void lock() const`—Puts a lock on the thread mutex.
- `void unlock() const`—Unlocks the thread mutex.
- `const OMProtected& getGuard() const`—Gets the reference to the `OMProtected` part.
- `OMProtected m_omGuard`—Is a private `OMProtected` part.

`omGetEventQueue()`, a virtual, public method was added. This method returns the event queue. It is not used inside the framework.

The private `init()` method was renamed `_initializeOMThread()`.

The `execute()` method implementation was modified to improve event dispatching performance.

OMUCollection

The default initial size of a collection was reduced from 256 to 20 elements. This change was made to reduce the default memory requirements of the collection.

OMState

The class name was modified from `State` to `OMState` as part of the global namespace cleanup (see **Global Namespace Cleanup**). In addition, the "`OM`" prefix was added to all the classes derived from `OMState`.

A new macro, `IS_EVENT_TYPE_OF(id)`, was added to support generic derived event handling both in flat and reusable statechart implementation (see **Generic Handling of Derived Events**).

The macro `OM_DECLARE_FRAMEWORK_MEMORY_ALLOCATION_OPERATORS` was added to the class declaration to support enhanced user control over framework memory allocation (see **User Control over Framework Memory Management (C++)**).

The following elements that were defined only in instrumented mode are now defined in non-instrumented mode as well to support partial animation (see **Partial Animation**):

- The `stateHandle` attribute

  In non-instrumented mode, the attribute value is always set to NULL.
- The `getConcept()` virtual method

  In non-instrumented mode, the operation always returns NULL.
- The `serializeStates()` virtual method

  In non-instrumented mode, the operation implementation is empty.

OMTimerManager

The `OMThreadTimer` class, which was derived from `OMTimerManager`, was merged into the base class because the separation of the timer manager into two classes was artificial. You can still use `OMThreadTimer`, which is a `typedef` of `OMTimerManager`.

`theSysTimer` global instance was replaced with a singleton instance. As a result, the following operations were added to the class interface:

- `static OMTimerManager* getStaticTimerManager()`—This method has two overrides, one that actually creates the singleton instance, and the other that lets you get a reference to the instance, if it was created.
- `static void clearInstance()`—Cleans up the singleton instance of the timer manager.
- `static OMBoolean m_timerManagerSingletonDestroyed`—This static attribute is used to indicate that the timer manager singleton is destroyed and should not be accessed.

The following elements that were defined only in instrumented mode are now defined in non-instrumented mode as well to support partial animation (see **Partial Animation**):

- `void suspend()`—Sets the suspended attribute to `TRUE`
- `void resume()`—Sets the suspended attribute to `FALSE`
- `OMBoolean suspended`—Used by animation to control the application execution

OMValueCompare

The `OMValueCompare` template class moved from `rawtypes.h` to `OMValueCompare.h` to allow its usage when the framework is not compiled with the `OM_USE_STL` compilation flag.

The class now uses the `std:: namespace` directly. If your compiler does not support the `std` namespace and you are using qualified relations and `STLContainers` for relation implementation, you must compile your application with the `NO_STD_NAMESPACE` compilation flag.

Adapter Changes

OSAL global functions were replaced with static member functions as part of the global namespace cleanup (see **Global Namespace Cleanup**).

The macro `OM_DECLARE_FRAMEWORK_MEMORY_ALLOCATION_OPERATORS` was added to the declaration of the following classes to support enhanced user control over framework memory allocations (see **User Control over Framework Memory Management (C++)**):

- `OMOSTimer`
- `OMOSThread`
- `OMOSEventFlag`
- `OMOSMutex`
- `OMOSSemaphore`
- `OMOSMessageQueue`

- `OMOSConnectionPort`
- `OMOSSocket`
- `OMEventQueue`

For `OMOSSocket`, the `lsbFirst` attribute was removed from all the implementations of the in-house adaptors. The attribute is redundant because the `htons()` standard function is used instead.

For Microsoft adapters, the `OM_DECLARE_FRAMEWORK_MEMORY_ALLOCATION_OPERATORS` macro was added to the declaration of the following classes to extend user control over memory allocated in the framework (see **User Control over Framework Memory Management (C++)**):

- `NTHandleCloser`
- `OMNTCloseHandleEvent`

For the VxWorks adapter, the `OM_DECLARE_FRAMEWORK_MEMORY_ALLOCATION_OPERATORS` macro was removed from the adapter classes' declaration because it is no longer needed (the macros were added in the generic base classes).

For the Linux adapter, a new static method was added to `LinuxThread` to handle deletion of active classes. The method signature is as follows:

```
static void endThreadHandler(int)
```

Rhapsody in C-Specific OXF Changes

This subsection documents Rhapsody in C-specific changes to the OXF.

RiCCollection

The default, initial size for collections was reduced from 256 to 20 elements to reduce the memory requirements of the collection.

A new method was added to the collection to remove the element in the specified position, which might modify the collection order. The method signature is as follows:

```
void RiCCollection_removeAt(RiCCollection *const, unsigned int)
```

RiCEvent

A new inline operation (`#define`), `RiCEvent_isTypeOf(event,id)`, was added as part of the generic, derived event handling. Because Rhapsody in C does not support inheritance, this API was added mainly for future use.

RiCReactive

The following changes were made to support handling of unconsumed events (see **Handling Unconsumed Events and Triggered Operations**):

- The `RiCEventResult` enum was merged into the `RiCTakeEventStatus` enum.
- The `RiCReactive_consumeEvent()` return type was modified from `void` to `RiCTakeEventStatus`.
- `RiCReactive_Vtbl` was changed, as follows:
  - The `consumeEvent()` return type was modified from `void` to `RiCTakeEventStatus`.
  - Two new entries were added to the table:
    - `handleEventNotConsumed()` is called when the reactive class fails to consume an event. The signature is as follows:

      ```
      void (*)(struct RiCReactive * const,
            struct RiCEvent*).
      ```

    - `handleTONotConsumed()` is called when the reactive class fails to consume a triggered operation. The signature is as follows:

      ```
      void (*)(struct RiCReactive * const,
            struct RiCEvent*).
      ```

RiCTask

The `execute()` method implementation was modified to improve event dispatching performance.

Adapter Changes

For `RiCOSSocket`, the `lsbFirst` attribute was removed from all the implementations of the in-house adapters. The attribute is redundant because the `htons()` standard function is used instead.

Rhapsody in J-Specific OXF Changes

This subsection documents Rhapsody in J-specific changes to the OXF.

RiJEvent

A new public method, `boolean isTypeOf(long id)`, was added to support generic handling of derived events (see **Generic Handling of Derived Events**).

The following attribute was added:

```
private boolean isTriggeredOperation
```

This attribute is used internally by the framework to distinguish between events and triggered operations. This is done as part of unconsumed event handling.

The attribute has a public getter and setter, as follows:

```
public boolean getIsTriggeredOperation()
```

```
public void setIsTriggeredOperation (boolean)
```

RiJStateReactive

The following elements were added to the class interface to support handling of unconsumed events (see **Handling Unconsumed Events and Triggered Operations**):

- `public void handleEventNotConsumed(RiJEvent event)`—Is called when the consumption of an event fails. This method has an empty implementation; it is up to the client to override this method in order to handle unconsumed events.
- `public void handleTONotConsumed(RiJEvent event)`—Is called when the consumption of a triggered operation fails. This method has an empty implementation; it is up to the client to override this method in order to handle unconsumed triggered operations.

Additional Information

This subsection describes additional changes in the prerelease version of Rhapsody.

Incremental Code Generation

Rhapsody 4.0 introduces a significant improvement in code generation performance by generating code only for elements that were modified after the last code generation.

This change takes effect only after a complete regeneration of a model. To force regeneration of the code, use the **Re Generate** option in the Code menu. Forced code generation behaves like code generation in Rhapsody 3.0.1.

Rhapsody generates a new file in the configuration directory named `<configuration name>.cg_info`. This internal file is needed for incremental code generation. It should *not* be under CM control.

Event IDs

As part of the support of complex model collaboration, Rhapsody ensures that event IDs will not collide.

Any event IDs and package base event IDs that you did not explicitly set will be modified during the first code generation of the model in Rhapsody 4.0.

To disable this behavior, set the value of the property `CG::Component::CalculatePackageEventBaseId` to `OnCodeGeneration`. Rhapsody will use the same event IDs as in Rhapsody 3.0.1.

Note

Rhapsody 3.0.1 does not guarantee to keep event IDs unchanged (that is, a specific event ID can vary across time).

Derived Statecharts (Flat)

Rhapsody 4.0 does not duplicate the state attributes and `IS_IN` methods of base classes. Instead, the state attribute's type was modified to integer (`int`), and each class state enumeration hold only the additional states added by the derived class.

This scheme cases a change in the code of classes with derived statecharts.

Note that this change does not affect code generated for activity diagrams.

Temporary Files

During code generation, Rhapsody creates temporary files. In Rhapsody 3.0.1, these files were created in the model directory. Rhapsody 4.0 creates these files in the system temporary directory.

You can modify the location of the temporary files by setting the location in Rhapsody `INI` file, as follows:

```
[CodeGen]
TemporaryFilesDirectory=<the temporary directory>
```

Partial Animation

Rhapsody 4.0 supports partial animation in Rhapsody in C and Rhapsody in C++. This feature is `not` supported in Rhapsody in J.

There are two ways to use partial animation:

- In the same selected component, using properties to enable/disable the animation of specific packages, classes, and so on.
- Mix animated and non-animated components in the same executable.

This feature also supports tracing.

To support partial animation, the following changes were made in Rhapsody in C++:

- **Code generation**
  - Inheritance of user classes and events from `AOM` elements was canceled.

  - For each animated user class (event), a friend class is created in the code. The friend class is responsible for the animation of the user class.
  - All the animation-specific methods are now part of the animation friend class.

- **OXF**
  - Inheritance from `AOM` classes was canceled (`OMEvent` and `OMReactive`).
  - Attributes that were protected by `#ifdef _OMINSTRUMENT` are now regular attributes, with default values that can be handled by the non-animated version of the framework.
  - Animation friend classes were added for the framework-visible events.

Generalization

Rhapsody 4.0 introduces full support in modeling of generalization (inheritance). As a result, the `CPP_CG::Class::VirtualInherits/PrivateInherits` properties became obsolete. The property content is converted to the model elements, and their content is deleted.

The properties' content will remain only if Rhapsody cannot convert all their content to model elements.

If these properties were used, the model is updated to store the information.

Handling Unconsumed Events and Triggered Operations

All three versions of the Rhapsody framework now include the ability to handle events and triggered operations that were not consumed. This addition is conceptually a callback method that you must override to define the actual handling of unconsumed events.

To support this modification, the `consumeEvent()` signature in Rhapsody in C and Rhapsody in C++ was modified.

User Control over Framework Memory Management (C++)

Rhapsody 4.0 enhances the 3.0.1 facility of application control over memory allocated in the framework. The enhancement were in two areas:

- Complete the memory management coverage, so every memory allocation in the generic framework as well as all the RTOS adaptors is using the memory management mechanism.
- Complete the usage of the `returnMemory()` interface, so the memory size returned is passed (as opposed to 0 in version 3.0.1).

Generic Handling of Derived Events

Rhapsody 4.0 introduces a generic way to handle the consumption of derived events.

In previous versions of Rhapsody, any change in the hierarchy of the events required regeneration of every class that consumed one of the base events whose hierarchy was modified. Rhapsody 4.0 reduces the coupling between the consumer class and the event hierarchy, so there is no longer a need to regenerate the consumer when changing the event hierarchy.

The support in generic handling of derived events was done by adding a new method, `isTypeOf()`, for every event, and modifying the generated code to check the event using this method. The `isTypeOf()` method returns `True` for derived events, as well as for the actual event.

Upgrading to Version 3.0.1

The changes in version 3.0.1 of Rhapsody are listed below.

Properties

This subsection describes the changes made to properties for Rhapsody 3.0.1. Note that in the `<lang>_CG` subject, the `<lang>` placeholder can be `C`, `CPP`, or `JAVA`.

Modified Properties

The default values for the following properties have been changed:

- The `<lang>_CG::VxWorks::CPPCompileDebug` property was changed to "`-O0 -g`" to avoid GNU compiler crashes when compiling PPC CPUs.
- The `CPP_CG::Solaris2/SolarisGNU::InvokeExecutable` property was modified to "`xterm -e $executable`" to correct execution problems under Solaris.
- The error message parsing string for the `CPP_CG::OseSfk::ParserErrorMessage` property was changed to correct an error highlighting problem under the OSE soft kernel.
- The `CPP_CG::QNXNeutrinoGCC::InvokeMake` property was changed to an empty string (" ") because the application must be built on the QNX target, and the Rhapsody UI does not support building of QNX applications.
- In Rhapsody in C++, the property `<ContainerTypes>::EmbeddedFixed::IterGetCurrent` was changed to "`(($target *)&$cname[$iterator])`" to fix compilation errors.

New Properties

The following properties have been added:

- `CG::Configuration::PreFrameworkInitCode (MultiLine)`

  This property enables you to add code to the generated `main()` before the call to the framework initialization (`OXFInit()` in C++).

- `CG::Package/Class::CallUserInitRelations (Bool)`

  This property disables calls to overridden `initRelations()` methods. See **Code Generation** for more information.

- `<lang>_CG::Microsoft/VxWorks::GetConnectedRuntimeLibraries (String)`

  See the documentation for the Web-enabled devices feature in the *User Guide*.

- `<lang>_CG::Solaris2/SolarisGNU/JDK::UnixLineTerminationStyle (Bool)`

  This environment property enables you to generate UNIX end-of-line style instead of DOS style. Using this property, you can generate code from a Windows host to a UNIX target without having to preprocess the generated files before compilation. In addition, you can add this property to other environments to generate UNIX end-of-line style under these environments.

- `ConfigurationManagement::ClearCase::Delete (MultiLine)`

  This optional property specifies the script that deletes a particular item from the current ClearCase directory element when you delete that item from Rhapsody.

- `ConfigurationManagement::ClearCase::DeleteActivation (enum)`

  This optional property controls whether the delete operation (specified by the `Delete` property) is enabled.

- `ConfigurationManagement::ClearCase::History (String)`

  This ClearCase-specific property specifies the batch script that enables you to view the version tree of a given item.

- `ConfigurationManagement::ClearCase::Rename (MultiLine)`

  This optional property specifies the script that renames a particular item in the current ClearCase directory element when you rename that item in Rhapsody.

- `ConfigurationManagement::ClearCase::RenameActivation (enum)`

  This optional property controls whether the rename operation (specified by the `Rename` property) is enabled.

- `ConfigurationManagement::ClearCase::ShowNewItemsInSynchronize (Bool)`

  This ClearCase-specific property is directly related to what you see in the Synchronize dialog box.

If this property is set to `No`, new items that are added (by another member of the team) to the archive after the Rhapsody project is open are not displayed.

Code Generation

In Rhapsody 3.0, if you override the generate `initRelations()` method (by creating your own method with the same name or using the **Synthesized Code in Model** option), Rhapsody will no longer automatically call the method in the class constructors.

In Rhapsody 3.0.1, the code generation behavior was modified so the `initRelations()` method is called, even if it is overridden in the model.

You can disable the calls to overridden `initRelations()` methods by setting the value of the `CG::Package/Class::CallUserInitRelations` property to `False`.

Framework

The Rhapsody 3.0.1 framework supports two additional customization enhancements:

- **Application-level control over the timer used by the framework**

  This feature enables you to register a timer factory on the framework, causing the framework to use the user-defined timers instead of the predefined timers. You can register a timer factory that does not create any timers, causing the timing mechanisms of the framework to be disabled. For example:

  ```
  disable tm()
  ```

  To have an effect, the user factory must be registered before the framework initialization (`OXFInit()`).

  The calls for registering the timer factory (as well as the definition of the timer factory itself) are language-dependent. The method names are as follows:

Rhapsody Edition	Method Name
Rhapsody in C	`RiCOXF_setTickTimerFactory()`
Rhapsody in C++	`OXF::setTheTickTimerFactory()`
Rhapsody in J	`RiJOXF.setTheTimerFactory()`

- **The ability to replace the default active class (main thread) of the framework**

  This feature enables you to register an alternate default active object on the framework.

This is useful when you customize the behavior of application active classes.

To have an effect, the user factory must be registered before the framework initialization (`OXFInit()`).

The calls for registering an alternate default active object are language-dependent. The method names are as follows:

Rhapsody Edition	Method Name
Rhapsody in C	`RiCOXF_setTheDefaultActiveObject()`
Rhapsody in C++	`OXF::setTheDefaultActiveClass()`
Rhapsody in J	`RiJOXF.setTheDefaultActiveClass()`

For more information, refer to the Rhapsody framework documentation.

Rhapsody in C++ Framework

The following sections describe upgrade issues that affect Rhapsody in C++ only.

Memory Control

The Rhapsody 3.0.1 C++ framework introduces the ability to control memory allocated in the framework at the application level (for example, when adding an object to a relation implemented as `OMList`).

To control the allocated memory, you must register a memory manager for the framework using the call `OXF::setMemoryManager()`. If you do not register a memory manager, the framework uses the global `new` and `delete` operators.

For more information, refer to the Rhapsody framework documentation.

A new class, `OMMemoryManager`, was added to support this functionality. This class is located in the files `ommemorymanager.cpp/h`. For custom adaptors, you must add these files to the OXF makefile.

The 3.0.1 OXF has built-in memory control support for the following elements:

- All generic types except for states. There is no full support for reusable state machines.
- OS adaptor support for VxWorks. To add support to other OS adaptors, add `OM_DECLARE_FRAMEWORK_MEMORY_ALLOCATION_OPERATORS` in the adaptor classes' declaration, and use the `OMNEW` and `OMDELETE` macros for buffer allocation and deletion.

You can compile the memory management out of the framework (using the standard `new` and `delete` operators directly) by defining the `OM_NO_FRAMEWORK_MEMORY_MANAGER` compilation

flag in the framework and user makefiles. This option reduces the overhead created by the framework attempt to obtain the user memory manager.

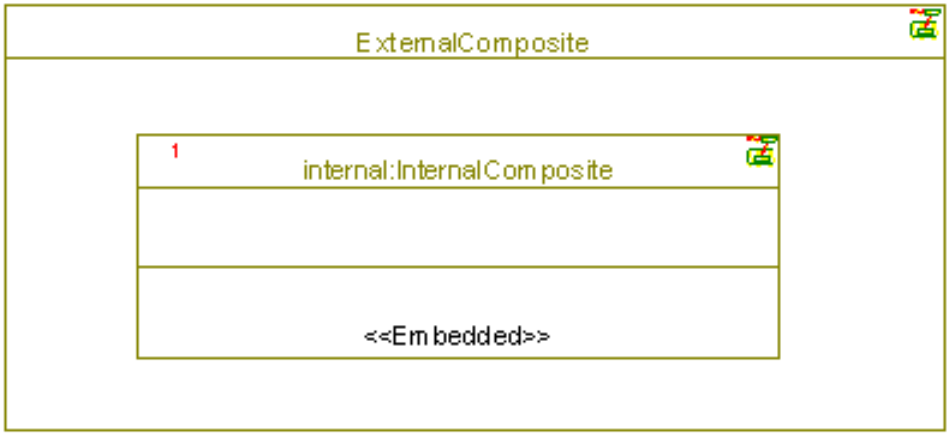
OMReactive

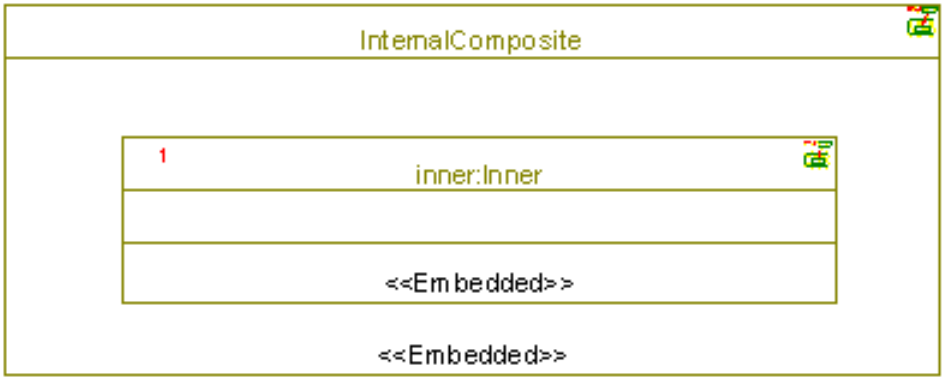
The `OMReactive::setThread()` is now public and virtual to support correct thread passing in complex embeddable composition.

The modified declaration of `OMReactive` is as follows:

```
class RP_FRAMEWORK_DLL OMReactive {

    ...

    // The setThread function is virtual and public
    // to allow the user to initialize nested, embedded
    // reactive components of an active class manually.
    // Rhapsody supports only one level of nesting in
    // such cases.
    // Changing the reactive thread is highly dangerous,
    // and should be done only before calling
    // startBehavior().

       virtual void setThread(OMThread *t,
           OMBoolean active = FALSE);

    ...
};
```

Consider the `ExternalComposite` embedded class:





The code generated for `ExternalComposite` is as follows:

```
ExternalComposite::ExternalComposite(OMThread* p_thread)
{
    setThread(p_thread, FALSE);
    {
        internal.setShouldDelete(FALSE);
        internal.setThread(p_thread, FALSE);
    }
    initStatechart();
}
```

To make the inner instance run on the same active class as `ExternalComposite` and `InternalComposite`, you must create `InternalComposite::setThread()` to do the necessary call to `inner.setThread()`.

OMThread

Rhapsody 3.0.1 includes the following changes to the `OMThread` class:

- The signature of `start()` is now virtual to support user customization of the default active class. The change might affect the behavior of user active classes that have `start()` methods with the same signature.
- Three methods were added to support static instances of active classes (particularly the static instance of `OMMainThread`). These methods are `destroyThread()`, `cleanupThread()`, and `_cleanupThread()`.

  If you have a custom RTOS adaptor that deletes threads in `OSEndApplication()`, modify the adapter to call `destroyThread()` instead of the `delete` operator.

  If you create by-value instances of an active class, you should override the `destroyThread()` method to prevent the system from attempting to delete the static instances.
- The method `getStaticThreadsPtrList()` was added to provide a static instance of the threads list, instead of dynamic creation.

The modified declaration of `OMThread` is as follows:

```
class RP_FRAMEWORK_DLL OMThread : public OMProtected {

    ...

    public:

    // Start the thread and the event loop.
    //
    // IMPORTANT: OMThread ignores the start parameter!!
    // The parameter should be checked only in default
    // application threads (OMMainThread).
    //
    // When creating an alternative default thread,
    // when doFork is set to 0, the framework is expected
    // to use the OS main thread.
    // When doFork is set to any other value, it should
```

```
        // create a new thread.

           virtual void start(int = 0);

        // API to destroy the active class
        // The method is used to destroy the framework
        // default active class object.
        // If you are setting your own default active
        // class object, and the object was not created
        // dynamically (by calling new), you must override
        // this method to avoid its deletion.

           virtual void destroyThread() {delete this;}
           protected:


        // Cleanup - hook to allow cleanup of a thread
        // without calling the DTOR.
        // This method is needed to allow cleanup without
        // destroying the v-table.

           virtual void cleanupThread() {_cleanupThread();}

        private:


        // Return a static instance of the threads list.

           static OMThreadPtrList* getStaticThreadsPtrList();

        // Cleanup - called from the DTOR and from
        // cleanupThread().

           void _cleanupThread();

          ...
};
```

OMMainThread

Rhapsody 3.0.1 includes the following changes to the `OMMainThread` class:

- The return type was modified from `OMMainThread*` to `OMThread*` to support the default active class customization.

  If you are using a custom RTOS adaptor that calls `OMMainThread::instance()`, modify the expected return type.
- The `OMMainThread` singleton implementation was modified, and the instance is now statically allocated (on the stack instead of the heap) by the `getInstance()` method.
- If you are using a custom RTOS adaptor, and the adaptor attempts to delete threads in `OSEndApplication()`, modify the adaptor to call `destroyThread()` instead of the `delete` operator.

The modified declaration of `OMMainThread` is as follows:

```
class RP_FRAMEWORK_DLL OMMainThread : public OMThread {
...
public:

    static OMThread* instance(int create=1);

    // Override destroyThread() to disable deletion
    // of a statically allocated instance.
    // Call cleanupThread() to perform cleanup.

       virtual void destroyThread() {
          this->cleanupThread();
    }
private:

    // Actually get the main thread instance.

       static OMMainThread* getInstance();
       ...
};
```

OMTimerManager

In Rhapsody 3.0.1, the timer manager singleton (`theSysTimer`) implementation was modified, and the instance is now statically allocated.

Custom RTOS adaptors that try to delete the system timer manager should modify the call from `delete theSysTimer;` to:

```
theSysTimer-> destroyTimer();
```

The modified declaration of `OMTimerManager` is as follows:

```
class OMTimerManager {

public:
    ...

    RP_FRAMEWORK_DLL void destroyTimer() {
        this->~OMTimerManager();
    }

    ...

};
```

Rhapsody in J Framework

Rhapsody Version 3.0.1 includes major improvements to the threading behavior, and the Java equivalent of C++ framework functionality that was missing from previous versions of the product.

The following sections describe upgrade issues that affect Rhapsody in J only.

RiJThread

The `RiJThread` class includes new methods that allow access to the thread status from other classes, particularly by the animation framework.

The modified declaration of `RiJThread` is as follows:

```
public class RiJThread extends RiJEventDispatcher
    implements RiJActive {
    ...
    //## operation isStarted()

    public boolean isStarted() {
        //#[ operation isStarted()
            return wasStarted;
        //#]
    }

    //## operation isSuspended()

    public boolean isSuspended() {
        //#[ operation isSuspended()
            return threadSuspend;
```

```
        //#]
    }
    ...
}
```

RiJStateReactive

`RiJStateReactive` event consumption was upgraded to provide more deterministic event consumption. All the changes in this class interface are related to these changes.

The changes to `RiJStateReactive` are as follows:

- The call of a triggered operation from the consumption of another `event/TO` was blocked (the second `TO` is ignored). This prevents the consumption of a `TO` while the state machine is in an undefined state.

  This behavior is controlled by the `busy` attribute and the `*Busy()` operations. To disable the behavior, override the `isBusy()` operation so it returns `False` at all times. However, this is not recommended.
- The `mutualExclusionOfTrigOp()` method (attribute, getter and setter) is used by the animation framework.
- The `lockEventMutex()` and `freeEventMutex()` operations are reserved for future use.
- The `_takeEvent()` method is the common code for synchronized and nonsynchronized event consumption.
- The `eventMutex` attribute and the `createEventMutex()` operation are used to support mutual exclusion between events and `TO` consumed in the same state machine.
- When you make `TO` guarded in the generated code call `createEventMutex()` in the user class `CTOR`, the result is a call to `synchronized(eventMutex)` in `takeEvent()` before `event/TO` is consumed. This call ensures the mutual exclusion between events and `TO` consumption, but also makes the event consumption slower. Therefore, make `TO` guarded only if there is a real need to enable the mutual exclusion between events and `TO` consumption (that is, events and `TO` are consumed in the same state machine).

The modified declaration of `RiJStateReactive` is as follows:

```
public class RiJStateReactive extends RiJReactive {

    ...
    private Boolean busy = Boolean.FALSE;
    private RiJMutex eventMutex = null;
    protected boolean mutualExclusionOfTrigOp = true;
    protected int _takeEvent(RiJEvent event) { ... }
    public void createEventMutex() { ... }
    public void lockEventMutex(){ ... }
    public short freeEventMutex() { ... }
    public void doBusy() { ... }
    public void undoBusy() { ... }
    public boolean isBusy() { ... }
```

```
        public boolean getMutualExclusionOfTrigOp(){ ... }
        public void setMutualExclusionOfTrigOp(boolean) { ... }

    }
```

RiJTimer

`RiJTimer` is a new interface, added for the support of application level control over the timer used by the framework. Any code in the framework that referred to `RiJSimpleTimer` now refers to `RiJTimer` and `RiJSimpleTimer` to implement the new interface.

Upgrading to Version 3.0 MR1

The changes in version 3.0 MR1 of Rhapsody are listed below.

Rhapsody in C++ 3.0 MR1 supports OSE Delta version 4.3.1 instead of 4.0.1. Due to a compatibility issues, some parts of the framework API were modified.

OMOSMutex Interface Changes

The changes to the C++ framework `OMOSMutex` interface are as follows:

- The `free()` method was renamed to `unlock()`.
- A new, non-virtual method `free()` was introduced to support backward compatibility with user applications that use `OMOSMutex` directly. This method is not available in OSE Delta.

If you have your own adaptor, you must rename `OMOSMutex::free()` to `OMOSMutex::unlock()`.

The new `OMOSMutex` interface is defined as follows:

```
class RP_FRAMEWORK_DLL OMOSMutex
{
   public:

   virtual ~OMOSMutex(){};
   virtual void lock() = 0;
   virtual void unlock() = 0;
   virtual void* getOsHandle() const = 0;
      // get the real OS element

   #ifndef OSE_DELTA

   // backward compatibility support for non-OSE
   // applications

   void free()
   {
      unlock();
   }

   #endif

};
```

State Interface Changes

The changes to the C++ framework API `State` interface are as follows:

- The `enterState()` method replaces the `enter()` method.
- The `exitState()` method replaces the `exit()` method.

The changes were applied to all the classes derived from `State`. In addition, the generated code for reusable statecharts has been modified to conform to the framework changes.

Note

If you are using reusable statechart implementation, you must regenerate all your reactive classes.

The modified interface is as follows:

```
class RP_FRAMEWORK_DLL State
{

   ...

public:

   ...

   virtual void enterState()=0;
   virtual void exitState()=0;
   ...

};
```

Upgrading to Version 3.0

The changes in version 3.0 of Rhapsody are listed below.

Code Generation

Note the following upgrade issues related to code generation:

- **Guarded Destructor**—In version 2.3, the concurrency of a destructor effectively behaved as if it were set to guarded whenever there was a guarded operation in the same class. In 3.0, the concurrency of the destructor is set to guarded only when you explicitly set it that way. If your model has reactive classes that might be deleted at run time (not via a Termination connector), you must explicitly set their destructors to be guarded.

  It is recommended that you use a Termination connector for self-destruction.

- **Event IDs**—Rhapsody 3.0 changes the way event IDs are generated. As part of the upgrade process, you must regenerate the model to prevent behavioral errors related to event ID collisions.

- **Code Generation for Actors**—Rhapsody 3.0 supports code generation for actors. When you open a version 2.3 model in 3.0, code generation for actors is disabled for backward compatibility and to protect you from illegal code in legacy actors.

  To enable code generation for actors, do the following:

1. Add the relevant actors to the relevant component scope.
2. In each legacy configuration that should generate actors, go to the configuration Initialization tab and check **Generate Code for Actors**.

   In new configurations, the **Generate Code for Actors** option is enabled by default.

- **Class Layout**—Rhapsody 3.0 improves the generated file layout, giving you control over the order in which attributes and relations are generated in the class. If you want to preserve the version 2.3 layout, set the `InitialLayoutAs23` property (under `CG::CGGeneral`) to `True` at the project level.

Framework

Rhapsody 3.0 moves the simulated time support from link time to run time via a parameter provided to the framework at application initialization. However, to switch between real and simulated time, you must still regenerate and build the code.

Properties

This subsection describes the changes made to properties for Rhapsody 3.0. For ease of use, the properties are grouped by subject.

Note that if you try to use property files from Rhapsody 2.3 with version 3.0, the following error is displayed:

```
Exception occurred in <model component>, it may indicate an integrity problem
of your model. It is recommended that you save your model to a new location,
exit Rhapsody and re-open your model to check integrity.
```

This happens if the version of the `\Properties` directory does not match the version of Rhapsody you are using. For example, you installed a newer version of Rhapsody, but kept an existing version of the `\Share\Properties` directory, including files contained within the directory.

Make sure you use the version of the `\Properties` directory that matches the version of Rhapsody you are using.

CG

The following properties were moved from `CG` to `<lang>_CG` (for example, `CPP_CG` for C++):

- `GetAt` property (under the `Relation` metaclass)
- `GetAtGenerate` property (under the `Relation` metaclass)

If these properties are overridden in your `site.prp` file, you must modify the subject names manually. For overrides within the model, the conversion is done automatically.

<lang>_CG

- The `CPPAdditionalReservedWords` property (under the `<environment>` metaclasses) was replaced with `AdditionalReservedWords`. If this property is overridden in your `site.prp` file, you must rename the property manually.
- The `MakeFileGenerationScheme` property (under the `Configuration` metaclass) was removed. If you are using the Rhapsody 2.0 makefile generation scheme in Rhapsody

in C++ (with external flags and rules makefiles), you must upgrade your makefile generation scheme.

- The spelling of the `QouteOMROOT` property (under the `<environment>` metaclasses) was corrected to `QuoteOMROOT`. If this property is overridden in your `site.prp` file, you must rename the property manually.
- The `TimerMaxTimeouts` property was removed. If you have an override on this property, it will still affect the code generation. However, the property is obsolete. Use the `TimerMaxTimeouts` property (under `<lang>_CG::Framework`) instead.
- The `TimerTicktim` property was removed. If you have an override on this property, it will still affect the code generation. However, the property is obsolete. Use the `TimerResolution` property (under `<lang>_CG::Framework`) instead.
- The `TimerTickTime` property (under the `Framework` metaclass) was replaced with the `TimerResolution` property. If this property is overridden in your `site.prp` file, you must rename the property manually.

ClassImporter

All `ClassImporter` subjects were changed to `<lang>_ReverseEngineering` subjects with the same properties. If any properties in these subjects are overridden in your `site.prp` file, you must rename the subjects manually.

In addition, the `CreateImplicitClassifier` property (under the `<lang>_ClassImporter` metaclasses) was removed.

General

The `CppReservedWords` property (under the `Model` metaclass) was removed.

Checks

Rhapsody 3.0 adds a check that prevents code generation when a class, actor, event, or global variable within the component scope has the same name as the component. In Rhapsody in J, the check prevents generation of a class with the name `Main<component>`.

This check was added because Rhapsody generates a class for the component. When the model has global instances, multiple definitions of the same class are generated, one for the user class and the other for the generated component class. This means that if your model has elements and component with the same name, you must modify the class name, or the component name, in order to generate code.

You can disable the check by setting the `AllowCollisionWithComponentName` property (under `CG::Configuration`) to `True`. However, if you do this, Rhapsody will not protect you from redefinition and name collision at the code level.

Upgrading Rhapsody in C++ Models

This subsection describes additional upgrade considerations that are specific to Rhapsody in C++.

Framework

Changes were made to the OS Abstraction Layer (OSAL) for all supported OS adapters. If you are using a custom adapter, you must implement these changes yourself. The process of adapting Rhapsody to a new OS, and the OSAL itself, are described in the *RTOS Adapter Guide*, which is part of the Rhapsody documentation set.

In general, there are three changes to the OSAL:

- `getOsHandle()` methods were added to every primitive in the OSAL.
- `OMOSMessageQueue::get()` signatures were modified to support a success status return value from `GEN()` and `GEN_ISR()` calls. Additional changes in return type were made in `OMReactive` and `OMThread` for the same reason. These changes are shown in the following section.
- Interface support was added for terminating Rhapsody-related parts of the application without terminating other parts.

The framework changes are shown in the following code.

```
class OMOSTimer
{
  public:
  virtual void* getOsHandle() const = 0;
};
class OMOSThread
{
  public:
  virtual void* getOsHandle(void*& osHandle) const = 0;
};
class OMOSFactory
{
  public:
  virtual OMBoolean waitOnThread(void* osHandle,
  timeUnit ms) = 0;
};
extern void OSOXFEndProlog();
```

```
class OMOSEventFlag
{
   public:
  virtual void* getOsHandle() const = 0;
};
class OMOSMutex
{
   public:
  virtual void* getOsHandle() const = 0;
);
class OMOSSemaphore
{
   public:
  virtual void* getOsHandle() const = 0;
};
class OMOSMessageQueue
{
   public:
  virtual OMBoolean put(void* m,
      OMBoolean fromISR = FALSE) = 0;
            virtual void* getOsHandle() const = 0;
};
template<class Msg> put(Msg *m, OMBoolean fromISR = FALSE)
   {
          return theQueue->put((void *) m, fromISR);
   }
   virtual void* getOsHandle() const {
   return theQueue->getOsHandle();
   }
};
class OMOSSocket
{
   public:
  virtual void Close() {}
};
```

To extend framework customization, the following methods were set to virtual:

`OMReactive`:

```
virtual OMBoolean _gen(OMEvent *event,
                OMBoolean genFromISR = FALSE);
          virtual OMBoolean gen(OMEvent *event,
   OMBoolean genFromISR = FALSE);
          virtual OMBoolean gen(OMEvent *event,
                void *sender);
OMThread:


virtual OMBoolean queueEvent(OMEvent *ev,
                OMBoolean fromISR = FALSE);
          virtual void cancelEvent(OMEvent *ev);
          virtual void cancelEvents(OMReactive *destination);
          virtual void schedTm(timeUnit delteTime, short id,
          OMReactive *instance, const OMHandle *state = NULL);
          virtual void unschedTm(short id, OMReactive *c);
```

Code Generation

Rhapsody 3.0 does not support generation of makefiles in Rhapsody 2.0 Compatibility mode. If you are using the backward compatibility mode, you must upgrade your makefile generation scheme by setting the `MakeFileGenerationScheme` property (under `CPP_CG::Configuration`) in any of the model's configurations. If the property exists and its value is 2.0, you must upgrade the makefile generation scheme.

Generation of dependencies with a stereotype of «`Friend`» was modified in version 3.0 to conform to the UML standard. For backward compatibility, earlier Rhapsody models are set to use version 2.3-style dependency generation. To modify the code generation scheme for dependencies with a «`Friend`» stereotype, use the `<<Friend>>ImplementationScheme` property (under `CPP_CG::General`).

Properties

The spelling of the `SpecFilesInDependencyRules` property (under `CPP_CG::Osesfk`) was corrected.

Roundtrip

Rhapsody 3.0 supports enhanced dynamic model/code associativity (see the code generation and round-trip documentation). However, when you load version 2.3 models, the new roundtrip

capabilities are disabled because the new capabilities require additional information (generated annotations) in the code.

To use the new roundtrip scheme in a model being upgraded from version 2.3, do the following:

1. Modify the `RoundtripScheme` property (under `CPP_Roundtrip::General`) from `Basic` to `Full`.
2. Regenerate the code. If you do not regenerate the code, the model will be overpopulated when the roundtrip executes.

STL Support

The code for STL containers was upgraded to use the `std` namespace prefix when needed. This change allows the use of STL containers without a special STL version of the framework. To use an STL-oriented framework, you must recompile the framework with the `USE_STL` switch in the makefile compiler switches.

If you are already using STL containers, you should expect changes in your code. To prevent these changes, override the properties in the `STLContainers` subject in your `site.prp` file using the property values from the same subject in your version 2.3 `factoryC++.prp` file.

To use qualified relations in STL containers without an STL-oriented Rhapsody framework, you must define `OMValueCompare`. This value is used in the generated code for the relation. The code as defined in the STL-oriented framework is:

```
template<class Key, class Value> class OMValueCompare
{
    public:
              OMValueCompare(Value value) : m_value(value) {}
              operator() (const pair<Key,Value>& p1)
   {
                        return p1.second == m_value;
   }
       private:
              Value m_value;
};
```

Upgrading Rhapsody in C Models

In the Rhapsody in C framework, the `RiCOSMessageQueue_isEmpty()` method was replaced with a macro to improve performance. If you are using your own adapter, you will need to do the same.

If your implementation of this method is too complex to replace with a macro, you can declare the method in the `RiCOS<env>.h` file for your adapter using the following method signature:

```
RiCBoolean RiCOSMessagQueue_isEmpty (
    RiCOSMessageQueue * const)
```

Upgrading Rhapsody in J Models

This subsection describes additional upgrade considerations that are specific to Rhapsody in J.

Framework

The location of the Rhapsody in J framework model was changed to `$OMROOT\LangJava\model`. The new model contains the animation framework as well as the Object eXecution Framework (OXF). If you added the framework model as a reference package to your version 2.3 model, you must remove it and add the new package from the new location.

Code Generation

In Version 3.0, the following changes were made to the build (make) file for the Rhapsody in J framework model:

- The list of source files is generated to an external file, which is used as input to the Java compiler. This change was made to solve the problem of large models with too many files for a single `javac` command line.
- If you have overridden the `MakeFileContent` property (under `JAVA_CG::JDK`), you should modify the `javac` call to:

```
> javac -g @$SourceListFile
```

Using Rhapsody 2.3 and Rhapsody 3.0 Concurrently

You can switch between versions 2.3 and 3.0 of Rhapsody if you follow these steps:

1. Install Rhapsody 3.0 into a different directory than your Rhapsody 2.3 directory.

The Rhapsody 3.0 installation will not overwrite your existing `rhapsody.ini` file; it will only rename it to `rhapsody.ini.orig`.

2. If you want to work with Rhapsody 2.3, rename your existing `rhapsody.ini` file to `rhapsody.ini.30` and rename `rhapsody.ini.orig` back to `rhapsody.ini`.

3. From your Rhapsody 2.3 installation directory, execute the following command:

```
rhapsody.exe /RegServer
```

You can create simple batch files to switch from version 2.3 to 3.0, and from 3.0 to 2.3. Before switching versions, back up the `.ini` file of the previous version to keep any changes to it, such as additional helpers or the last files opened.

Switching from Version 3.0 to 2.3

The body of the batch file should be:

```
copy c:\winnt\RhapsodyV23.ini c:\winnt\Rhapsody.ini
rhapsody.exe /RegServer
pause
```

Switching from Version 2.3 to 3.0

The body of the batch file should be:

```
copy c:\winnt\RhapsodyV30.ini c:\winnt\Rhapsody.ini
rhapsody.exe /RegServer
pause
```

Upgrading from 1.x and 2.x

This section describes behavior and functionality changes between versions of Rhapsody that you must consider when upgrading your installation from version 1.*x* or 2.*x*.

Upgrading from Version 1.x

C++ language models created in Rhapsody before version 2.0 cannot be loaded directly into 3.0 without going through an intermediate conversion. To upgrade a pre-v2.0 model, you must first load it into any version of Rhapsody 2.0 or later and save it. Then load the converted model into the current version, once it is installed (you might require a new license).

Note that this restriction does not apply to the C or Java versions of Rhapsody.

Upgrading from Version 2.x

Upgrading from version 2.*x* requires a new license.

All 2.0 configurations are translated into components with a single configuration having the same name. You cannot add version 2.0 or earlier configurations to version 2.2 or later models because configurations are no longer collaboration units in these models. You must first load the configurations into a version 2.0 model, and convert them to 2.2 or later as part of the 2.0 model.

Rhapsody Version 2.1 and later fixed corruptions in 2.0 models where transitions in the view did not have a representation in the model. In cases where the corruption could not be fixed, the transitions are colored in black and should be reentered.

Configuration switches for version 2.2 models do not include the `LangCpp` directory. If the configuration switches are coming from the `factory.prp` file, you must delete the configuration and create a new one. If they are coming from the `site.prp` file or property settings, you must make the changes there, delete the configuration, and then create a new one.

In addition, language-specific code generation properties moved from the `CG` subject to the language-specific `<lang>_CG` subjects. You must update these properties in the appropriate `site<lang>.prp` file (for example, `siteC++.prp` for C++ models). It is recommended that you

run a report on the version 2.2 model to get a list of modified properties, then check their locations in the new property files to see whether a property moved from one subject to another.

Index

Symbols

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W