Before using the information in this manual, be sure to read the IBM “Notices” section of the IBM Rational Help.

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


US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
Contents

The Rational Rhapsody API .................................................. 1
Information Available to the API ........................................... 1
Hierarchy of API Interfaces .................................................. 2

Rational Rhapsody Java API Basic Concepts .............................. 5
Java API Features .................................................................. 5
Java API Differences from COM API ........................................ 6
Rational Rhapsody Environment Initialization before Using Rational Rhapsody API on Linux .................................................. 6

COM API Basic Concepts and Examples .................................... 7
COM API Tools and Languages ............................................ 7
  COM API with Visual Basic ............................................. 7
  COM API with VBScript .................................................... 9
  Setting Up the COM Interface for Visual C++ ......................... 12
Manipulating Project Elements ............................................. 18
  Creating a Project Element ............................................... 18
  Modifying an Element ..................................................... 19
  Deleting an Element ....................................................... 19
Handling Properties Using the API ......................................... 20
  Propagation of Default Property Values ............................... 20
  Methods for Manipulating Properties. ................................. 21
Error Handling ................................................................. 22
  Catching an Error Condition in VB ..................................... 22
  Error Codes ................................................................... 23
Installing Custom Helpers .................................................... 24
Adding Helpers to Rational Rhapsody ...................................... 24

Rational Rhapsody API Interface .......................................... 25
Rational Rhapsody API Examples .......................................... 26
  Running the RPYReporter Example ..................................... 26
Table of Contents

Running the RPYExplorer Example ................................................................. 27
Running RPYReporter in Visual Basic .......................................................... 29
  VB Forms ....................................................................................... 31
  Running RPYReporter Step-by-Step ......................................................... 32
Starting and Saving Your Own VB IDE Work ............................................... 41
  Saving the Examples as New Projects ..................................................... 41
  Making Your Own New Projects .......................................................... 41
  Compiling and Making Your Executables ............................................... 41
Creating Applications with Microsoft Word VB IDE ................................. 42
  Specifying the Macro Content .............................................................. 43
  Comments on the Code ........................................................................... 45
  Modifying the Example to Print Classes .................................................. 46
Rhapsody API Interfaces ................................................................................ 49
Access to VB Properties ............................................................................... 50
API Conventions ......................................................................................... 50
Rhapsody Interfaces ..................................................................................... 51
  IRPAction Interface ............................................................................. 53
  IRPActor Interface ............................................................................... 54
  IRPAnnotation Interface ...................................................................... 54
  IRPApplication Interface .................................................................... 56
  IRPArgument Interface ........................................................................ 98
  IRPASCIIFile Interface ....................................................................... 100
  IRPAssociationClass Interface ............................................................ 103
  IRPAssociationRole Interface ............................................................... 103
  IRPAttribute Interface ......................................................................... 106
  IRPBlock Interface ............................................................................... 107
  IRPClass Interface ............................................................................... 108
  IRPClassifier Interface ......................................................................... 125
  IRPClassifierRole Interface ................................................................. 159
  IRPCollaboration Interface .................................................................. 160
  IRPCollaborationDiagram Interface ....................................................... 182
  IRPCollection Interface ....................................................................... 183
  IRPCollectionInterface ....................................................................... 185
  IRPCollaborationInterface ................................................................... 185
  IRPComponent Interface ...................................................................... 185
  IRPComponentDiagram Interface .......................................................... 203
  IRPComponentInstance Interface .......................................................... 203
  IRPConfiguration Interface .................................................................. 204
  IRPConnector Interface ......................................................................... 220
  IRPConstraint Interface ....................................................................... 232
  IRPControlledFile ................................................................................ 232
  IRPDependency Interface ...................................................................... 233
<table>
<thead>
<tr>
<th>Interface</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRPDeploymentDiagram Interface</td>
<td>233</td>
</tr>
<tr>
<td>IRPDiagram Interface</td>
<td>233</td>
</tr>
<tr>
<td>IRPEnumerationLiteral Interface</td>
<td>238</td>
</tr>
<tr>
<td>IRPEvent Interface</td>
<td>239</td>
</tr>
<tr>
<td>IRPEventReception Interface</td>
<td>240</td>
</tr>
<tr>
<td>IRPExecutionOccurrence Interface</td>
<td>241</td>
</tr>
<tr>
<td>IRPExternalCodeGenerator Interface</td>
<td>241</td>
</tr>
<tr>
<td>IRPExternalCodeGeneratorInvoker Interface</td>
<td>257</td>
</tr>
<tr>
<td>IRPFile Interface</td>
<td>259</td>
</tr>
<tr>
<td>IRPFlow Interface</td>
<td>269</td>
</tr>
<tr>
<td>IRPFlowchart Interface</td>
<td>274</td>
</tr>
<tr>
<td>IRPFlowItem Interface</td>
<td>277</td>
</tr>
<tr>
<td>IRPGeneralization Interface</td>
<td>280</td>
</tr>
<tr>
<td>IRPGraphEdge Interface</td>
<td>281</td>
</tr>
<tr>
<td>IRPGraphElement Interface</td>
<td>281</td>
</tr>
<tr>
<td>IRPGraphicalProperty Interface</td>
<td>286</td>
</tr>
<tr>
<td>IRPGraphNode Interface</td>
<td>286</td>
</tr>
<tr>
<td>IRPGuard Interface</td>
<td>286</td>
</tr>
<tr>
<td>IRPHyperLink Interface</td>
<td>287</td>
</tr>
<tr>
<td>IRPImageMap</td>
<td>290</td>
</tr>
<tr>
<td>IRPInstance Interface</td>
<td>291</td>
</tr>
<tr>
<td>IRPInteractionOccurrence Interface</td>
<td>296</td>
</tr>
<tr>
<td>IRPInterfaceItem Interface</td>
<td>297</td>
</tr>
<tr>
<td>IRPLink Interface</td>
<td>303</td>
</tr>
<tr>
<td>IRPMessage Interface</td>
<td>304</td>
</tr>
<tr>
<td>IRPMessagePoint Interface</td>
<td>306</td>
</tr>
<tr>
<td>IRPModelElement Interface</td>
<td>308</td>
</tr>
<tr>
<td>IRPModule Interface</td>
<td>341</td>
</tr>
<tr>
<td>IRPNode Interface</td>
<td>341</td>
</tr>
<tr>
<td>IRPObjectModelDiagram Interface</td>
<td>344</td>
</tr>
<tr>
<td>IRPOperation Interface</td>
<td>344</td>
</tr>
<tr>
<td>IRPPackage Interface</td>
<td>349</td>
</tr>
<tr>
<td>IRPPin Interface</td>
<td>402</td>
</tr>
<tr>
<td>IRPPort Interface</td>
<td>404</td>
</tr>
<tr>
<td>IRPProfile Interface</td>
<td>410</td>
</tr>
<tr>
<td>IRPPProject Interface</td>
<td>410</td>
</tr>
<tr>
<td>IRPRelation Interface</td>
<td>429</td>
</tr>
<tr>
<td>IRPRequirement Interface</td>
<td>434</td>
</tr>
<tr>
<td>IRPSequenceDiagram Interface</td>
<td>434</td>
</tr>
<tr>
<td>IRPState Interface</td>
<td>436</td>
</tr>
<tr>
<td>IRPStatechart Interface</td>
<td>457</td>
</tr>
<tr>
<td>IRPStateVertex Interface</td>
<td>465</td>
</tr>
<tr>
<td>IRPStereotype Interface</td>
<td>470</td>
</tr>
<tr>
<td>IRPStructureDiagram Interface</td>
<td>470</td>
</tr>
<tr>
<td>Interface</td>
<td>Page</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>IRPSwimlane Interface</td>
<td>470</td>
</tr>
<tr>
<td>IRPTag Interface</td>
<td>471</td>
</tr>
<tr>
<td>IRPTemplateInstantiation Interface</td>
<td>471</td>
</tr>
<tr>
<td>IRPTemplateInstantiationParameter Interface</td>
<td>472</td>
</tr>
<tr>
<td>IRPTemplateParameter Interface</td>
<td>472</td>
</tr>
<tr>
<td>IRPTransition Interface</td>
<td>474</td>
</tr>
<tr>
<td>IRPTTrigger Interface</td>
<td>487</td>
</tr>
<tr>
<td>IRPType Interface</td>
<td>490</td>
</tr>
<tr>
<td>IRPUunit Interface</td>
<td>506</td>
</tr>
<tr>
<td>IRPUseCase Interface</td>
<td>512</td>
</tr>
<tr>
<td>IRPUseCaseDiagram Interface</td>
<td>519</td>
</tr>
<tr>
<td>IRPInternalOEMPlugin</td>
<td>519</td>
</tr>
<tr>
<td>IRPVvariable Interface</td>
<td>519</td>
</tr>
</tbody>
</table>

### The Callback API 521

- **Callback API Introduction** 521
- **Events with Callback Methods** 522
- **API Details** 523
  - IRPApplicationListener 523
  - IRPRoundTripListener 525
  - IRPCodeGeneratorListener 525
- **Callback Logging** 526
- **Disabling Callback Notification** 526
- **Disabling Cancellable Actions** 526
- **Sample Client Applications** 527

### Quick Reference 529

### Index 545
The Rational Rhapsody API allows you write applications that access and manipulate Rational Rhapsody model elements. Two versions of the API are provided with Rational Rhapsody:

- COM
- Java

Information Available to the API

The Rational Rhapsody API facilitates reading, changing, adding to, and deleting from all model elements that are available in the Rational Rhapsody browser. The browser displays the static elements of a model including, but not limited to, the following:

- Model information
- Descriptions and other information within browser forms
- Information describing the model hierarchy, components, and packages
- Configurations and profiles
- Features and properties
- File and directory names
- Diagrams in a form that can be printed or included in external files for printing, such as Microsoft® Word®
Hierarchy of API Interfaces

The class diagram depicts the hierarchical relationships between the API interfaces. The application (IRPApplication) is the top-level object of the Rational Rhapsody object model. The hierarchy of the API interfaces is as follows:

IRPApplication
IRPASCIFile
IRPCollection
IRPExternalCodeGenerator
IRPExternalCodeGeneratorInvoker
IRPFlow
IRPGraphElement
IRPGraphEdge
IRPGraphNode
IRPGraphicalProperty
IRPModelElement
IRPAction
IRPAnnotation
  IRPComment
  IRPConstraint
  IRPRequirement
IRPAssociationRole
IRPClassifierRole
IRPCollaboration
IRPComponentInstance
IRPConfiguration
IRPDependency
  IRPHyperLink
IRPEnumerationLiteral
IRPExecutionOccurrence
IRPFile
IRPGeneralization
IRPGuard
IRPInteractionOccurrence
IRPInterfaceItem
  IRPEvent
  IRPEventReception
  IRPOperation
IRPLink
IRPMessage
IRPMessagePoint
IRPStateVertex
  IRPConnector
  IRPState
IRPStereotype
IRPSwimlane
IRPTemplateInstantiation
IRPTemplateInstantiationParameter
IRPTransition
IRPTrigger
IRPUnit
IRPClassifier
  IRPActor
  IRPClass
  IRPAssociationClass
  IRPFlowItem
  IRPNode
  IRPType
  IRPUseCase
IRPComponent
IRPDiagram
  IRPCollaborationDiagram
  IRPComponentDiagram
  IRPDeploymentDiagram
  IRPObjectModelDiagram
  IRPSequenceDiagram
  IRPStatechart
  IRPFlowchart
  IRPStructureDiagram
  IRPUseCaseDiagram
IRPPackage
  IRPProfile
  IRPProject
IRPRelation
  IRPInstance
  IRPBlock
  IRPModule
  IRPPort
IRPVariable
  IRPArgument
  IRPAttribute
  IRPTag
  IRPTemplateParameter
In terms of its capabilities, the Rational Rhapsody Java API is identical to the Rational Rhapsody COM API. The reference material for the COM API can be used to see what you can do with the Java API. The names of the objects, attributes, and methods in the Java API are more or less the same as those in the COM API.

For the details of the Rational Rhapsody Java API, see the Javadoc output for the API, which can be found at [rhapsody installation directory]\Doc\java_api\index.html.

A sample that uses the Java version of the Rational Rhapsody API can be found in the directory:

[Rhapsody installation directory]\Samples\JavaAPI

A more advanced sample can be found in the directory:

[Rhapsody installation directory]\Samples\CustomCG
   Samples\Statechart_Simplifier_Writer\Statechart_Java_Simplifier

Java API Features

Rational Rhapsody includes a Java version of the Rational Rhapsody API that can be used for working with Rational Rhapsody models. Since the Java API can be used on both Windows and Linux, this API allows you to write cross-platform applications.

Rational Rhapsody provides two files that can be found in the directory [installation directory]\Share\JavaAPI:

- Rhapsody.jar—contains the Java classes and interfaces
- Rhapsody.dll (or Rhapsody.so for Linux)—native implementation of the Java interfaces

The .jar file should be included in the CLASSPATH of the Java project, and the .dll (or .so file) should be included in the lib path.

To access the Rational Rhapsody application, you use the object RhapsodyAppServer. See the API javadoc output for details.
Java API Differences from COM API

The following are specific differences between the Rational Rhapsody Java API and the Rational Rhapsody COM API:

- Methods in the Java version of the API throw RhapsodyAPIException exceptions. You can use the toString method to get the description of the exception.
- IRPCollection provides a method called toList that returns a native Java list container populated with the elements of the collection. This is the recommended method of iterating over collections with the Java version of the API. (In Java 1.5, you can cast the list to a types list and thus benefit from the for-each iterator.)
- Unlike the COM version of the API, where you have to use the IDispatch::QueryInterface method, in the Java version, you can use the native Java operator instanceof.
- To check whether two interfaces point to the same model element, you should use the native boolean Object.equals(Object) method.

Rational Rhapsody Environment Initialization before Using Rational Rhapsody API on Linux

An initialization script called rhp_env (located in the root of the Rational Rhapsody installation directory) must be run before using Rational Rhapsody on Linux.

This is done automatically when Linux users launch Rational Rhapsody as described in the documentation. However, this script must also be run by Linux users who run Java applications that include use of the Rational Rhapsody API.

When you write a Java application that includes use of the Rational Rhapsody API, make sure to inform the users of the application that they must run the initialization script prior to running the Java application.

Alternatively, you can try to automate this process for the users of your application, for example, by having the script run as part of each users Linux startup process, or by including a call to this script in the script file you provide for launching your Java application (provided, of course, that Rational Rhapsody is installed in the same location on each users computer).
The Rational Rhapsody Repository API consists of a set of COM interfaces that supports dual interfaces (COM and automation). This allows access from Visual Basic and any language implemented with COM bindings. COM interfaces allow access from either Visual Basic® or VBScript, even when type information is not available (for example, OLE automation).

**Note**

Each interface represents a class in the Rational Rhapsody repository, and the set of interfaces forms the Rational Rhapsody object model. Each instance in the Rational Rhapsody repository returns a reference to a particular COM interface based on its metaclass. For example, access to an event in the Rational Rhapsody repository is via the IRPEvent interface.

### COM API Tools and Languages

The following sections describe how to use the Rhapsody COM API tools and languages

#### COM API with Visual Basic

Like all COM-based APIs, two components are required to create Rational Rhapsody automation scripts:

- The Rational Rhapsody COM type library, rhapsody.tlb. COM type libraries are self-documenting and easy to browse using COM object viewers.
- A Rational Rhapsody executable providing COM server functionality.

In Visual Basic, attach the rhapsody.tlb library to the project by selecting **Project > References**. This familiarizes the VB environment with the Rational Rhapsody API interfaces. No further action is required. VB implicitly connects to the Rhapsody server (rhapsody.exe) once the VB application is executed.
Example

The following VB program shows an example of how to traverse all the classes and add a serial number property (initialized to 0) to each one.

```vbnet
Public Sub SetClassesInPackage(p As IRPPackage)
    ' Routine to add recursively a property to all classes in a package
    Dim allClassifiers As RPCollection
    Set allClassifiers = p.nestedClassifiers
    Dim c As RPClassifier
    For each c in allClassifiers
        isClass = c.isOfMetaClass 'Class'
        If isClass Then
            On error resume next
            If not c.addProperty('general:class:serialNo', 'int', '0') then
                If not err.Number then
                    Print 'class can’t be assigned a property', c.name
                end if
            Else ' Check for nested packages
                isPackage = c.isOfMetaClass 'Package'
                If isPackage Then ' nested package case
                    Dim nestedP as Package
                    Set nestedP = c ' cast classifier to package
                    SetClassesInPackage nestedP
                End If
            End If
        Next
    End Sub

    ' The main program
    Dim Rph As Object
    Dim ProjName As String
    Dim Prj As RPProject
    Dim Packages As RPCollection

    Set Rph = CreateObject("Rhapsody.Application")
    ProjName = 'D:\Rhapsody\Examples\PingPong.rpy'
    Rph.OpenProject projName
    Set Prj = Rph.activeProject
    Packages = Prj.packages
    Dim p As RPPackage
    For each p in allProjectClassifiers
        SetClassesInPackage p
    Next
```
COM API with VBScript

Most Rational Rhapsody users on Windows platforms can use the Visual Basic IDE programming environment or VBA, which are not available on a Solaris platform. However, Rational Rhapsody users on Solaris platforms can access the Rational Rhapsody API using VBScript (Visual Basic Scripting edition), a cross-platform development language.

Running VBScript

The setup for running VBScript scripts is done during installation. Note the following:

1. Before running a VBScript script, you must run Rational Rhapsody at least once for registration of the COM interfaces in the registry.
2. Run the VBS script located in the Rational Rhapsody home directory.
3. Use the vbtest program by Mainsoft™ to run vbs programs.

VBScript samples are available in the Samples/Vbs directory of the Rational Rhapsody installation.

Writing Files from VBScript

Some of the elements of Visual Basic are not included in VBScript, such as file input/output functions. Rhapsody compensates for this with the addition of a File object to the Rational Rhapsody COM library to facilitate reading and writing to files. To write to files, use code similar to the following in your script:

```
rem Create a rhapsody object.
.
.
rem Create and open a file object.
Set F = CreateObject("Rhapsody.RPASCIIFile")

rem Use it to open a file.
F.open "/tmp/show.txt"

rem Use is to write to the file with VB script commands.
F.write "Succeeded in opening project " + vbCrLf

rem Close the file when finished with it.
F.close
```
**Example VBScript**

The following VBScript script dumps packages, classes, and events. It is included in the Rhapsody installation.

```vb
Dim rappl
Dim appl
Dim p
Dim s
Dim c
Dim pack
Dim F
Set rappl = CreateObject("Rhapsody.Application")
Set F = CreateObject("Rhapsody.RPASCIIFile")
F.open "/tmp/show.txt"
F.write "Succeeded in opening project "+ vbLf
MsgBox "Started Rhapsody" + rappl.version
s = "/disk1/RP/Samples/Pingpong/pingpong.rpy"
rappl.openProject s
Set p = rappl.activeProject
Set c = p.components
For Each pack In c
    MsgBox pack.Name
Next
dim NextPack, NextOperation
rem Lets send them to a file
level=1
ShowPackages p, level
F.close
MsgBox "Done listing the Project"

sub ShowPackages(p, levelPack)
    CallLevelPack = levelPack + 1
    Set Pk = p.packages
    For Each Pack In Pk
        PrintSpace levelPack
        F.write"Package:"+pack.Name+vbLf
        ShowClasses Pack, CallLevelPack
        CallLevelClass = levelClass + 1
        ShowEvents Pack, CallLevelPack
    Next
End Sub

sub ShowClasses(Pack, levelClass)
    CallLevelClass = levelClass + 1
    set NextPack = Pack.Classes
    PrintSpace levelClass
    F.write"Classes:"+vbLf
    For Each Class In NextPack
        PrintSpace levelClass
        F.writeClass.Name+"":"+vbLf
    Next
End Sub
```
ShowOperations Class, CallLevelClass
CallLevelClass = levelClass + 1
ShowAttributes Class, CallLevelClass
Next
End Sub

sub ShowOperations(Class, levelOperation)
CallLevelOperation = levelOperation + 1
set NextOperation = Class.Operations
PrintSpace levelOperation
F.write"Operations::"+vbLf
for Each Operation in NextOperation
    PrintSpace CallLevelOperation
    F.write":"+Operation.name+vbLf
Next
CallLevelClass = levelOperation + 1
End Sub

sub ShowAttributes(Class, levelAttribute)
CallLevelAttribute = levelAttribute + 1
set NextAttribute = Class.Attributes
PrintSpace levelAttribute
F.write"Attributes::"+vbLf
for Each Attribute in NextAttribute
    PrintSpace CallLevelAttribute
    F.write":"+Attribute.name+vbLf
Next
End Sub

sub ShowEvents(Pack, levelEvent)
CallLevelEvent = levelEvent + 1
set NextEvent = Pack.Events
PrintSpace levelEvent
F.write"Events::"+vbLf
for Each RHPEvent in NextEvent
    PrintSpace CallLevelEvent
    F.write":"+RHPEvent.name+vbLf
Next
End Sub

sub PrintSpace (levelPrint)
For x = 1 to levelPrint
    F.write ""
Next
End sub
Setting Up the COM Interface for Visual C++

Like all COM-based APIs, two components are required to create Rational Rhapsody automation scripts:

- The Rational Rhapsody COM type library, rhapsody.tlb. COM type libraries are self-documenting and easy to browse using COM object viewers. One such viewer is provided in the Share directory of the installation.
- A Rhapsody executable providing COM server functionality.

The class wizard can create Rhapsody proxy objects by attaching to the rhapsody.tlb library. This requires the VC++ project also to be COM-enabled.

The important steps in setting up the COM interface are as follows:

1. **Include an #import statement.** For example:

   ```
   #import "C:\Rhapsody\rhapsody.tlb" no_namespace
   named_guids
   ```

   This statement makes C++ recognize the various interfaces as C++ classes.

2. **Invoke the rhapsody.application object.** For example:

   ```
   IRPApplication apl = NULL;
   hr = CoCreateInstance(CLSID_RPApplication,
   NULL,CLSCTX_ALL, IID_IRPApplication, (void**)&apl);
   ```

3. **Access elements of the rhapsody.application object through API methods.** For example:

   ```
   // Get project file name
   IRPProjectPtr proj = NULL;
   hr = apl->openProject (projectFileName, &proj);

   // Get count of packages in project
   IRPCollectionPtr collection;
   hr = proj->get_packages(&collection);
   long elementsCollectionCount;
   hr = collection->get_Count(&elementsCollectionCount);
   ```

The following two examples demonstrate how to invoke Rhapsody from a C++ client using direct COM calls to the Rational Rhapsody API interface.
Sample: Reading from the API

The following example is the primary file in a Visual C++ workspace application that reads from a Rational Rhapsody project using the COM API interface.

```cpp
//
// ReadAPI.cpp : Defines the entry point for the console
//application.
//
#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include <atlbase.h>
// The following depends on the place that Rhapsody is installed
#import "F:\Documents\RiCPP_2.3MR1\Rhapsody\rhapsody.tlb"
raw_interfaces_only, no_namespace, named_guids

void printBSTR(BSTR errorMessage)
{
    LPCWSTR tmpName = errorMessage;
    char buf[1000];
    int tmpNameLen = tmpName != NULL ? wcslen(tmpName) : 0;
    if (tmpNameLen != 0)
    {
        wcstombs(buf, tmpName, (tmpNameLen*2)+1);
        printf(buf);
        printf(\"\n\n\");  
    }
}

void printErrorMessageIfError(HRESULT hr,
IRPModelElement* modelElement)
{
    if (FAILED(hr))
    {
        BSTR errorMessage;
        HRESULT tmpHr;
        tmpHr = modelElement
>getErrorMessage(&errorMessage);
        printBSTR(errorMessage);
    }
}

int loadProject(const char* rpyFileName)
{
    HRESULT hr;
    CLSID clsid;
    hr = CLSIDFromProgID(OLESTR("Rhapsody.Application"),
&clsid);
    if (FAILED(hr))
    {
        printf(_T("Failed to resolve CLSID. HR = 0x%8x"),hr);
        return 0;
    }
    // Create CoClass instance from ClassId, using dispatch iid
    IRPApplicationPtr apl;
    hr = ::CoCreateInstance( CLSID_RPApplication, NULL,
CLSCTX_ALL, IID_IRPApplication, (void**)&apl );
```
if (FAILED(hr))
{
    printf(_T("Failed to create instance. HR = 0x%8x"),hr);
    return 0;
}
int len = MultiByteToWideChar(CP_ACP, 0, rpyFileName,
strlen(rpyFileName), NULL, NULL);
BSTR projectFileName = SysAllocStringLen(NULL, len);
MultiByteToWideChar(CP_ACP, 0, rpyFileName,
strlen(rpyFileName), projectFileName, len);
IRPProjectPtr proj = NULL;
hr = apl->openProject (projectFileName,&proj);
SysFreeString(projectFileName);
IRPCollectionPtr collection;
hr = proj->get_packages(&collection);
long elementsCollectionCount;
hr = collection->get_Count(&elementsCollectionCount);
BSTRpackageName;
VARIANT r;
for ( int i = 1; i <= elementsCollectionCount; i++)
{
    IRPPackagePtr p;
    hr = collection->get_Item(i, &r);
    hr = r.pdispVal->QueryInterface(IID_IRPPackage,
        (void**)&p);
    hr = p->get_name(&packageName);
    printBSTR(packageName);
}
hr = apl->quit();
return 0;
}
void Usage()
{
    printf("Usage: ReadAPI rpyFile\n");
}
// General remark: In the following, in most cases there
// is no check on the returned hr for readability.
int main(int argc, char* argv[])
{
    HRESULT hr;
    hr = CoInitialize(0);
    if (FAILED(hr))
    {
        printf(_T("Failed to initialize COM"));
        return 0;
    }
    if (argc == 2)
        loadProject(argv[2]);
    else
        Usage();
    // loadProject("D:\Temp\Project.rpy");
    CoUninitialize();
    return 0;
}
Sample: Writing to the API

The following example is the primary file in a Visual C++ workspace application that writes to a Rational Rhapsody project using the COM API interface.

Note

Change the #import line to match your own project.

```cpp
// WriteAPI.cpp : Defines the entry point for the console application.

#include "stdafx.h"
#include <stdio.h>
#include <stdlib.h>
#include <atlbase.h>

// The following depends on the place that Rhapsody is installed
#import "D:\Rhapsody\rhapsody.tlb" raw_interfaces_only, no_namespace, named_guids

void printBSTR(BSTR errorMessage)
{
    LPCWSTR tmpName = errorMessage;
    char buf[1000];
    int tmpNameLen = tmpName != NULL ? wcslen(tmpName) : 0;
    if (tmpNameLen != 0)
    {
        wcstombs(buf, tmpName, (tmpNameLen*2)+1);
        printf(buf);
        printf("
");}
}

void printErrorMessageIfError(HRESULT hr, IRPModelElement* modelElement)
{
    if (FAILED(hr))
    {
        BSTR errorMessage;
        HRESULT tmpHr;
        tmpHr = modelElement->getErrorMessage(&errorMessage);
        printBSTR(errorMessage);
    }
}

int createNewProject(const char* userDirectoryName, const char* userProjectName)
{
    HRESULT hr;
    hr = CoInitialize(0);
    if (FAILED(hr))
    {
        printf(_T("Failed to initialize COM"));
        return 0;
    }
    CLSID clsid;
    hr = CLSIDFromProgID(OLESTR("Rhapsody.Application"), &clsid);
    if (FAILED(hr))
    {
        printf(_T("Failed to resolve CLSID. HR =
```
void Usage()
{
    printf("Usage: WriteAPI directoryName projectName\n");
}

0x%8x"),hr);
return 0;
}
void Usage()
{
    printf("Usage: WriteAPI directoryName projectName\n");
}

// Create CoClass instance from ClassId, using
// dispatch iid
IRPApplicationPtr apl;
hr = ::CoCreateInstance(CLSID_RPApplication, NULL,
    CLSCTX_ALL, IID_IRPApplication, (void**)&apl);
if (FAILED(hr))
    printf(_T("Failed to create instance. HR = 0x%8x"),hr);
return 0;
}

IRPProjectPtr proj = NULL;
inT len = MultiByteToWideChar(CP_ACP, 0,
    userDirectoryName, strlen(userDirectoryName), NULL,
    NULL);
BSTR projectLocation = SysAllocStringLen(NULL, len);
MultiByteToWideChar(CP_ACP, 0, userDirectoryName,
    strlen(userDirectoryName), projectLocation, len);
len = MultiByteToWideChar(CP_ACP, 0, userProjectName,
    strlen(userProjectName), NULL, NULL);
BSTR projectName = SysAllocStringLen(NULL, len);
MultiByteToWideChar(CP_ACP, 0, userProjectName,
    strlen(userProjectName), projectName, len);
hr = apl->createNewProject(projectLocation, projectName);
hr = apl->activeProject(&proj);
SysFreeString(projectLocation);
SysFreeString(projectName);
IRPPackagePtr package;
BSTR packageName = SysAllocString(L"myPackage");
hr = proj->addPackage(packageName, &package);
SysFreeString(packageName);
IRPClassPtr newClass;
BSTR className = SysAllocString(L"myClass");
hr = package->addClass(className, &newClass);
SysFreeString(className);
IRPOperationPtr operation;
BSTR operationName = SysAllocString(L"myOperation");
hr = newClass->addOperation(operationName,
    &operation);
SysFreeString(operationName);
IRPAttributePtr attribute;
BSTR attributeName = SysAllocString(L"myAttribute");
hr = newClass->addAttribute(attributeName,
    &attribute);
SysFreeString(attributeName);
hr = proj->save();
hr = apl->quit();
CoUninitialize();
return 0;
}
// General remark: In the following, in most cases there
// is no check on the returned hr for readability.
int main(int argc, char* argv[])
{
    HRESULT hr;
    hr = CoInitialize(0);
    if (FAILED(hr))
    {
        printf(_T("Failed to initialize COM"));
        return 0;
    }
    if (argc == 3)
        createNewProject(argv[2], argv[3]);
    else
        Usage();
    // createNewProject("D:\temp\Project", "Project");
    CoUninitialize();
    return 0;
}
Manipulating Project Elements

The following sections describe how to create, modify, and delete Rhapsody project elements.

Creating a Project Element

There are two ways to add a new Rhapsody element:

- Add a new object while the project is still open in Rational Rhapsody using the method `addNewAggr` on an owner object, supplying the metatype, name, and receiving the newly created object.

  The syntax for the call is as follows:

  ```
  owner.addNewAggr(metaType, name);
  ```

  In this call, `metaType` and `name` are String expressions for the type and name of an object with which to form an aggregation relation with an owner object.

  For example, if a package `p` is present in your open model, you can execute the following code in Visual Basic:

  ```
  Dim c as RPClass
  c = p.AddNewAggr("Class","C");
  ```

  When finished, the new class `c` is added to package `p`.

- There are also `addObject` methods available for every object. For example:

  ```
  Dim cl as RPClass
  Dim attr as RPAttribute
  Set cl = Package.AddClass("C");
  Set attr = Class.AddAttribute("att");
  ```

  The objects created are connected to their owner. Even a new project can be created using a special method.

**Note:** Do not use the VB methods `createObject` or `createInstance` to create new elements. The only correct way to create new elements is with the `addNewAggr` method or the specific `addObject` methods.
Modifying an Element

When you attempt to modify an object through an API method, you call the appropriate method, such as `setName(newName)`. Rhapsody checks the permissions, and returns one of the values listed in the following table.

<table>
<thead>
<tr>
<th>Return State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>The operation is performed and returns without error. For example, you want to name a class “A”.</td>
</tr>
<tr>
<td>NOOP</td>
<td>The operation is not performed and returns without error. For example, you want to name a class “A”, but it already has that name.</td>
</tr>
<tr>
<td>NO</td>
<td>The operation is not performed and returns with an error. For example, you want to name a class “A”, but it is read-only, or there is already a class named “A” present. The error message <code>RP_CANT_MODIFY</code> is returned as the error message for this method.</td>
</tr>
<tr>
<td>WARNING</td>
<td>You can choose from two working modes: • Force mode on? WARNING is regarded as YES. • Force mode off? WARNING is regarded as NO.</td>
</tr>
<tr>
<td>MERGE</td>
<td>The operation is not performed as if a NO is returned. Merge routines are available.</td>
</tr>
</tbody>
</table>

Deleting an Element

The method `deleteFromProject` deletes an object from its package. In addition, there are `DeleteXXXX` methods that delete elements of a core object.

In the following examples, `cl` and `att` are wrappers to their core objects.

```vbnet
Package.DeleteClass(cl);
Class.DeleteAttribute(att);
```

Only `cl = NULL` and `att = NULL` in a Visual Basic application will delete the wrapper itself.
Handling Properties Using the API

Rhapsody model elements can have name/value pairs, known as properties, that extend the model in some way. They provide, for example, instructions for code generation, additional application-dependent properties, and so on.

The name (or key) part of the name/value pair is a string that must consist of three qualifying fields separated by a period. For example:

```
<lang>_CG.Configuration.Environment
```

The first of the three fields designates a subject, such as code generation, reverse engineering, and so on. The second field designates the metaclass (or stereotype) to which the property applies. The third field designates the name of the property.

The value part of the name/value pair is a string that can be interpreted as either a string value, an integer, a Boolean, or an enumerated type. For example, “Microsoft” is one of the enumerated values “Microsoft, MicrosoftDLL, VxWorks, Solaris2, Borland, MSStandardLibrary, PsosX86, PsosPPC, MicrosoftWinCE, OseSfk, Linux, Solaris2GNU, QNXNeutrinoGCC, QNXNeutrinoCW, OsePPCDiab” for the key `<lang>_CG.Configuration.Environment`.

For a given property name, a Rhapsody model element can have either a specific value (a value given to it by either a user or Rhapsody), or a default value, which it finds by searching a predefined search path. For some keys, it is possible to have no value at all.

Propagation of Default Property Values

To facilitate assignment of values to groups of model elements rather than a single model element each time, Rhapsody implements a propagation mechanism where property values propagate along the containment hierarchy. The propagation originates at the `factory.prp` file, continues to the project through the `site.prp` file, and then on to the configuration and model containment hierarchy.

For example, consider a class `C1` that is nested in a package `P11` that is nested in a package `P1`. Class `C1` is denoted by the expression `P1::P11::C1`. Assume that for all the classes in `P11` the statecharts should not be implemented (generated). To do this, the property `CG.Class.ImplementStatechart` should be set to `False` for package `P11`. By default, all classes within `P11` (recursively) “inherit” this value, unless overridden. If this behavior is required for the entire project, this property should be set to `False` at the project level.

**Note**

The propagation mechanism referred to resembles inheritance, although the word “inheritance” is intentionally not used to avoid confusion.
Handling Properties Using the API

Methods for Manipulating Properties

The API provides a number of functions that enable you to add or modify Rhapsody properties. These methods belong to the IRPModelElement interface and include the following:

- addProperty
- getPropertyValue
- getPropertyValueExplicit
- removeProperty
- setPropertyValue

You can use properties set in the site.prp file to create customized documentation. These properties can also be accessed by the API and changed as required.
Error Handling

All COM methods return a status of `HRESULT` indicating the success status of the method. In Visual Basic (VB), `HRESULT` is not visible and a failure status raises a VB error condition that, if not handled, aborts the calling program.

Most of the API functions do not create side effects, and therefore there is no reason for them to flag an error. However, the API might flag errors if permission on an update is not given.

The following table lists the methods that flag errors and might require error handling.

<table>
<thead>
<tr>
<th>Method</th>
<th>Member Of</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>addProperty</code></td>
<td>IRPModelElement</td>
</tr>
<tr>
<td><code>getPropertyValue</code></td>
<td>IRPModelElement</td>
</tr>
<tr>
<td><code>getPropertyValueExplicit</code></td>
<td>IRPModelElement</td>
</tr>
<tr>
<td><code>removeProperty</code></td>
<td>IRPModelElement</td>
</tr>
<tr>
<td><code>setPropertyValue</code></td>
<td>IRPModelElement</td>
</tr>
<tr>
<td><code>save</code></td>
<td>IRPProject</td>
</tr>
<tr>
<td><code>saveAs</code></td>
<td>IRPProject</td>
</tr>
</tbody>
</table>

Catching an Error Condition in VB

Catching an error condition in VB is performed using an On Error statement. A practical way to handle errors flagged by method calls is demonstrated by the following example:

```vbnet
On Error Resume Next
getSelectedElement.getPropertyValue("no.property.exists")
Dim s As String
getSelectedElement.getErrorMessage s
MsgBox s
```

In this example:

- `Resume Next` makes the program continue to execute at the statement immediately following the one that caused the error.
- The method `getErrorMessage`, defined for every model element, fetches a message of the most recent error occurrence. This message can be displayed to diagnose the error, as shown in the example.
Error Handling

Error Codes

A return value of zero indicates success. The following table lists the non-zero values that represent Rhapsody API error codes.

<table>
<thead>
<tr>
<th>Error</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RP_CANT_ADD_AGGREGATE</td>
<td>Could not add the element.</td>
</tr>
<tr>
<td>RP_CANT_MODIFY</td>
<td>The item cannot be modified.</td>
</tr>
<tr>
<td>RP_CANT_DELETE</td>
<td>The item cannot be deleted.</td>
</tr>
<tr>
<td>RP_NO_OPEN_PROJECT</td>
<td>There is no open project with which to interface.</td>
</tr>
<tr>
<td>RP_DELETED_OBJECT_ERROR</td>
<td>Indicates a reference to a deleted object.</td>
</tr>
<tr>
<td>RP_BAD_ENUMERATED_VALUE</td>
<td>The enumerated type used does not exist.</td>
</tr>
<tr>
<td>RP_BAD_PROPERTY_KEY_ERROR</td>
<td>Illegal property key syntax (not in &lt;subject&gt;.&lt;metaclass&gt;.&lt;name&gt; format).</td>
</tr>
<tr>
<td>RP_MISSING_PROPERTY_ERROR</td>
<td>The property requested does not exist.</td>
</tr>
<tr>
<td>RP_PROPERTY_EXISTS_ERROR</td>
<td>Attempt to add a property that already exists.</td>
</tr>
<tr>
<td>RP_CONFIGURATION_NOT_IN_COMPONENT_ERROR</td>
<td>Attempt to set an active configuration a nonexistent one.</td>
</tr>
<tr>
<td>RP_OPERATION_FAILED_ERROR</td>
<td>Applying an operation that cannot be handled by certain objects, although defined by its base interface. An example is addProperty, which is defined for all model elements, but currently generalization and reception cannot apply it.</td>
</tr>
<tr>
<td>RP_SAVE_FAILED_ERROR</td>
<td>The save or save as operation failed, probably because of lack of file writing privileges.</td>
</tr>
<tr>
<td>RP_CANNOT_WRITE_TO_FILE_ERROR</td>
<td>The provided file name cannot be opened for writing. Currently, this applies to the getPicture method of IRPDiagram.</td>
</tr>
</tbody>
</table>
Installing Custom Helpers

*Helpers* are custom programs that can be attached to Rhapsody to extend it. Helpers can be either external programs (executables) or VBA macros:

- An external program helper is typically either a VB or a C++ program that uses the COM API and connects to the Rhapsody instance via the `GetObject` COM service.
  
  **Note:** Currently, `GetObject` is not supported on Linux systems.

- A VBA macro helper is a VBA macro defined in a VBA module promoted to be a helper.

Helpers are attached to the Tools menu of Rational Rhapsody using the **Customize** option.

Adding Helpers to Rational Rhapsody

To add a helper, select **Tools > Customize** in Rational Rhapsody. The Helpers dialog box is displayed. This dialog box is similar to the Visual Studio external tools menu. You manipulate the menu and create new entries using the toolbar at the top of the dialog box, which includes the following tools:

- New
- Delete
- Move Up
- Move Down
Rational Rhapsody API Interface

Rational Rhapsody includes a interface tool for users who want to programmatically interact with their Rational Rhapsody projects for useful applications such as the preparation of custom reports. This interface is referred to as the Rational Rhapsody application programming interface (Rational Rhapsody API or simply API).

Without going into excessive detail, this lesson describes how to use the Visual Basic® API examples that come with Rational Rhapsody to make your own Visual Basic API applications.

This chapter describes how to perform the following tasks:

- Generate a report using RPYReporter.
- Generate a model tree using RPYExplorer.
- View the Visual Basic source code for RPYReporter and RPYExplorer.

The Rational Rhapsody API functions through a set of methods and attributes that act as a set of Microsoft COM interfaces. Using these methods and attributes, users of languages with COM bindings such as C++, Java, and Visual Basic (VB) can programmatically access a Rational Rhapsody project and all its model elements. Currently, access is restricted to read-only access for model elements and write access for model properties.
Rational Rhapsody API Examples

The Rational Rhapsody distribution includes two example applications prepared in Visual Basic that access Rational Rhapsody projects through the Rational Rhapsody API. The following sections describe these examples in detail.

Running the RPYReporter Example

to run the RPYReporter example:

1. Double-click on the executable file RPYReporter.exe in the Samples\CppSamples\Api\RPYReporter directory under your Rational Rhapsody installation directory. The RPY Project Reporter dialog box is displayed.

2. Click Load Project and browse for the Dishwasher project you completed in the tutorial.

3. Select your Dishwasher project, then click OK. Rational Rhapsody displays a wait screen while the project is being loaded.


   After preparing the report, the application displays the name and location of the text file containing the report so you can access it at any time.

5. Click OK to display the report in Notepad.

   The report contains detailed information about your model, including data types used, stereotypes, names of events, classes, operations, and so on.

6. Click File > Exit to close Notepad.

7. Click Exit to exit the application.

8. Click Yes when asked if you really want to quit.
Running the RPYExplorer Example

To run the RPYExplorer example, follow these steps:

1. Double-click the executable file RPYExplorer.exe in the Samples\CppSamples\API\RPYExplorer directory under your Rational Rhapsody installation directory. Rational Rhapsody displays the RPY Explorer window, as shown in the following figure.

![RHY Explorer window](image)

2. In the window, select **File > Load RPY Project**.

3. In the resultant dialog box, browse for your **Dishwasher** project, then click **Open**. The root of an expandable **Dishwasher** tree is displayed, with a plus sign in front of it.

4. Click the plus sign to expand the **Dishwasher** project.

   At the categories level, expandable segments appear for Packages, Object Diagrams, Sequence Diagrams, and so on.

5. Expand each category to reveal its contents.
6. To expand individual elements of a category, simply select them.

The RPYExplorer example has a browser similar to the Rational Rhapsody browser. Information for each highlighted model element is displayed on the right-hand side of the dialog box.

**Tools Menu Options**

The Tools menu options provide the following capabilities:

- Get, set, add and remove project properties using property dot notation (Subject.MetaClass.Property).
- Get nested elements recursively for a selected element. For example, if you highlight a component and select **Get Nested Elements Recursive** from the Tools menu, you receive a small report on all configurations and files in the component.
- Save a report of an element’s properties and methods to a text file.
- Report on a model.
- View diagrams. You can view a diagram only after storing a diagram as an .emf file.

**Storing and Viewing Diagram Files**

To store and view diagram files, follow these steps:

1. Highlight an individual diagram in the tree. The properties and methods for the diagram are displayed in the right-hand pane.

   When you highlight a diagram in the VB browser, VB automatically creates an .emf file of the diagram in your system’s temporary directory (for example, C:\TEMP). VB displays the message “getPicture: see metaFile in your TMP folder” in the right-hand panel.

2. To save the file to a different location (in addition to the one in your temporary directory), select **Tools > Create EMetaFile from the RPDiagram**. You are prompted for the name and location of a file in which to store the diagram.

3. To view a stored diagram file, select **Tools > RPDiagram Viewer**.

4. In the resultant dialog box, highlight the appropriate .emf file, then click **View Selected RPDiagrams**. The diagram is displayed.
Running RPYReporter in Visual Basic

The intent of this lesson is to describe how the examples were prepared so you can create your own applications.

The RPYReporter and RPYExplorer examples were created in the Microsoft Visual Basic 6.0 IDE (Interface Development Environment). Although the intent of this lesson is not to instruct you in Visual Basic, the features are explained as encountered in order to see how the examples were prepared. Note that although this tutorial uses Visual Basic version 6.0, version 5.0 is also compatible.

Do the following:

1. Start Microsoft Visual Basic 6.0 IDE using the Windows Start menu or from within Rational Rhapsody by selecting Tools > VBA > Visual Basic Editor.

2. In the New Project dialog box, select Standard EXE and click Open. The Microsoft Visual Basic design window is displayed with an empty, default project.

3. Select File > Open Project and browse for the RPYReporter project file, Project1.vbp, located in the subdirectory Samples\CppSamples\API\RPYReporter of the Rational Rhapsody installation directory. This is the same directory with the executable RPYReporter.exe.

4. Select Project1.vbp, then click OK to load it.

When the RPYReporter project is loaded, you should see several open windows in the VB IDE. The Project Explorer window has a browser-like appearance with the window title Project - RPYReporter, as shown in the following figure.

![Project Explorer Window](image)

If this window is not displayed, select View > Project Explorer on the VB desktop.
5. In the Project Explorer, double-click on the form **RPYReportDumpForm**. A window containing this form is displayed, as shown in the following figure.

This form is similar to the dialog box in the RPY Report executable.

Another window that should be present on the VB IDE is the Properties window, shown in the following figure.

If this window is not open, select **View > Properties Window**.
VB Forms

*Forms* are the basis for writing programs in Visual Basic. Each form consists of elements such as buttons, text fields, and pull-downs.

The form and its elements each have properties that are listed in the Properties window. Currently, the Properties window displays the properties for the entire form. You can show the properties of each form element by clicking on an individual element, then examining the Properties window.

**Placing Elements on Forms**

To place elements on a form, follow these steps:

1. Click the appropriate type of form element in the Form toolbox on the left.
2. Double-click a location for the element, or click and drag to establish its outline.

**Viewing the Element Properties and Code**

Each element has many properties, such as Appearance, BackColor, Caption, and Label. For example, if you click the *Load Project* button, you can see its properties consist of a name (cmdLoad), a type (CommandButton), and others such as Caption (“Load Project”), which labels the button. Note that the name cmdLoad begins with the three character prefix “cmd” which, by denotes a command button. Note the different prefixes used for the other elements.

Each form element automatically has code associated with it that reacts to different events on the element. The most common of these is the “Click” event. For each element that you can click, there is a Visual Basic subprogram that services that click, whose name is the same as the element’s name with the “_Click” suffix.

To view the properties and code associated with an element, follow these steps:

1. Click on each form element and observe the element type and name. These appear in the pull-down box at the top of the Properties window.
2. On the form, double-click the *Load Project* button to see the subprogram cmdLoad_Click() in the VB desktop.

A window appears with all of the code for the RPYReportDumpForm form that has been scrolled so the start of the cmdLoad_Click() subprogram is at the top, as shown in the following figure.
Note that the subprogram `cmdLoad_Click()` calls the subprogram `mnuFileLoad_Click()`. You can scroll through the entire contents of this code window to find `mnuFileLoad()`, or select it directly using the left pull-down at the top of the code window. The `mnuFileLoad_Click()` calls the subprogram `loadRPYProject()`, with the argument `projectNameText.Text`.

The RPYReporter example was originally built with menu commands instead of button commands, which is why `cmdLoad_Click()` calls `mnuFileLoad_Click()`. Currently, the menu command elements are invisible and therefore unusable.

To enable them, follow these steps:

1. Select **Tools > Menu File Editor**.
2. Check the **Visible** check box for the rows &File, &Tools, and &Help.
3. Uncheck these boxes for now because you do not want to use menus for the application.

### Running RPYReporter Step-by-Step

To step through the code of the RPYReporter example, follow these steps:

1. Press the F8 key to begin the RPYReporter example.

   In the RPYReportDumpForm, the first line of the `Form_Load()` subprogram is highlighted. This subprogram loads the form and sets the variable `doc` to the special value of `Nothing`.

   If you scroll to the very top of the code window, you can see the variable `doc` declared as an `Object`. VB enables you to create an object so it can be subsequently used to refer to an actual object. That object will eventually be the Rhapsody API Application object, which you will see later. For now, `doc` is assigned the value of `Nothing`, which keeps it from referencing anything.
Note: The keyword Private is used to indicate that a variable or subprogram is available only within the module in which it is declared. Therefore, the variable doc is relevant only to this code module, the one accompanying the form RPYReportDumpForm.

2. Press F8 three times until the Form_Load() subprogram is ended and the RPYReportDumpForm form is displayed.

3. Click Load Project to continue program execution.

   Selecting Load Project calls the local subprogram cmdLoad_Click(), which is now displayed and highlighted in the code window.

4. Continue pressing F8 to verify that cmdLoad_Click() calls the subprogram mnuFileLoad(), which calls mnuFileLoad_Click(), which calls the subprogram loadRPYProject() with the argument projectNameText.Text.

5. Press F8 to proceed to the first line of the subprogram loadRPYProject().

   The projectNameText element is the name of the long text box at the top of the RPYReportDumpForm form. This element has a property called Text, which is the actual text contents of that text box. The program can designate the contents of the text property using the expression projectNameText.Text. Thus, if you typed the project name in the projectNameText field, the subprogram loadRPYProject() would now have it as an argument. As it is, its value is currently an empty, or blank, string.

   Note: The following steps assume that you have clicked F8 to move to next section of code to be described.

6. The On Error GoTo CancelHandler line enables the Cancel button on the dialog box. If you click Cancel, execution continues at the code line following the line labeled CancelHandler:, located at the bottom of the loadRPYProject() subprogram that exits the subprogram.

7. rpyModelName is a string variable that will hold the name of the project you are loading. Its value is initialized to an empty string.

8. The next few lines involve properties and an operation of the object RPYModelDlg. This element, a common dialog box, does not appear on the form during execution until its operation ShowOpen is executed.

   The first three RPYModelDlg lines change the properties of the dialog for its initial directory, default file search pattern, and the name of the project (which was passed as an argument). Finally, the ShowOpen operation of the RPYModelDlg object is executed and the Open dialog box is displayed with the appropriate property changes.

9. Browse for your Dishwasher project, then click OK.
The step `rpyModelName = RPYModelDlg.FileName` is ready for execution. This step sets the string variable `rpyModelName` to the name of the project you selected in the Open dialog box.


In the following line, the variable `rpyModelName` is checked to see if it is empty. If so, the `loadRPYProject` subprogram exits. Otherwise, it loads the `waitForm` object, followed by the execution of the `waitForm` object’s `Show` operation (`waitForm.show`), which displays the `waitForm` form to tell the user the project is loading.

Before continuing program execution, you need to learn more about the Rational Rhapsody API.
The Rational Rhapsody API: A Closer Look

The Rational Rhapsody API is a set of classes consisting of operations and attributes that enable you to programmatically interact with a Rational Rhapsody project (repository) using a programming environment that supports Microsoft COM (Component Object Model). This allows an application to interface programs using COM, such as Rational Rhapsody. In this way, standard interfaces to obtain system services or provide functionality to other programs can be established.

You can make the Rational Rhapsody API classes available for the RPYReporter project file (Project1.vbp) using references, which allow the use of objects from other applications.

To see the list of references in this project, follow these steps:

1. Stop execution of the RPYReporter application by selecting Run > End in the VB integrated development environment (IDE).
2. Select Project > References in the VB IDE. VB displays the References dialog box, as shown in the following figure.

Access to the Rational Rhapsody API classes is made possible by referencing the RHAPSODY.tlb library file included in the Rational Rhapsody distribution. Without it, the Rational Rhapsody API is not available. Be sure to check this part of your project if this becomes questionable. When you create a new project to access a Rational Rhapsody model, the very first step is to make sure that your project references RHAPSODY.tlb.
The Rational Rhapsody API classes that come from the RHAPSODY.tlb reference, along with their operations and attributes, are visible in the VB design area. In Visual Basic, interface classes are implemented with names that begin with the letter “I.” However, when the interfaces are seen in the VB IDE, they appear without the “I.” For example, the IRPModelElement class appears as RPModelElement.

To display the Rational Rhapsody API classes and their methods and properties, follow these steps:

1. Select View > Object Browser. The Object Browser dialog box is displayed, as shown in the following figure.
2. Select the **rhapsody** library from the pull-down field. VB displays the Rational Rhapsody API classes, as shown in the following figure.

![Object Browser](image)

3. Click on one of the API classes to see its attributes and operations.

4. Click on an attribute or operation of the selected API class to view a small report on it at the bottom of the display area.

5. Click the “X” in the upper, right-hand corner to dismiss the dialog box.

**Continuing the Step-by-Step Execution of RPYReporter**

Now that you have seen how the Rational Rhapsody API is made available to the RPYReporter project, you can continue step-by-step execution of the RPYReporter application to see how it is used.

Continue executing each step of the program, as follows:

1. If you halted execution earlier, press F8 to begin step-by-step execution again.

2. The next execution step in the **loadRPYProject** subprogram calls the **disableAllButtons** subprogram, which sets all the enabled properties of all **RPYReportDumpForm** buttons to **False**, rendering the buttons unusable (grayed-out). Press F8 to move through the subprogram.

3. The next step compares **Not doc** (recall that **doc** is an object of type **Object**) against the value **Nothing**. Because **doc** was created a few steps ago and was initialized to **Nothing**, execution steps into the **Else** part of the **If-Then-Else** statement that follows it.
4. Because the module-level variable `THE_APPLICATION` has been set to the string “rhapsody.Application” (scroll to the top of the window to see the declaration) the line `Set doc = CreateObject(THE_APPLICATION)` makes `doc` a reference to the Rhapsody Application object and a stepping stone for upcoming use of the Rhapsody API.

   **Note:** Rhapsody is started as an application during the execution of the line `CreateObject(THE_APPLICATION)`.

5. Because `doc` is now a reference to the Application object, you can use API class operations and attributes through it. Therefore, the line `doc.openProject rpyModelName` actually calls the openProject subprogram of the Application object referenced by `doc`, and opens the project file you selected.

6. The next line, `Set theProject = doc.activeProject`, calls the activeProject method of the Application object referenced by `doc` and sets the project you loaded as the active project in Rational Rhapsody.

7. The `unload waitForm` line unloads wait dialog box.

8. The next line, `projectNameText.text` is set to the name and path of the Dishwasher model (rpy file) you selected.

9. Now that the project is loaded, the program calls `EnableAllButtons` to re-enable all the buttons on the main form. Press F8 to step through each button.

10. Now that a project has been loaded, the property `Enabled` of the `mnuToolsReport` object is set to True. The function of this menu item is equivalent to that of the Report on Project button.

11. Press F8 to step through the exiting of all subprograms that have been entered as part of project loading. These include, in order:
   a. loadRPYProject()
   b. mnuFileLoad_Click()
   c. cmdLoad_Click()

   The program now waits in stasis for the next event to occur through other button clicks on the RPY Project Reporter window.

**Code Summary of Loading a Project**

The following is a code summary of the project-loading process in VB:

```vbnet
Private doc As Object
Private ProjectName As String
Private theProject As RPModelElement

' Get project name and store as ProjectName
```
'Open the Rhapsody API Application Object
Set doc = CreateObject("rhapsody.Application")
doc.openProject ProjectName
Set theProject = doc.activeProject

Reporting on a Project

Reporting requires the execution of several important API operations. The following instructions assume that you have performed the previous project loading example and are continuing uninterrupted. However, if you have stopped the program, press F8 to enter the program in step-by-step mode and repeat all steps from the previous section. Otherwise, continue stepping through the program, as follows:

1. In the RPYReportDumpForm form, click **Report on Project**.

2. The subprogram cmdReport_Click() is called, which calls the subprogram mnuToolsReport_Click.

   Within the mnuToolsReport_Click subprogram, the waitForm form is loaded and displayed, and the buttons of the RPYReportDumpForm form are disabled.

3. Because the report will be written to a file, the function getDefaultLogFileName generates a name for the file using the project name string rpyModelName as a base.

4. After the name of the report output file is generated in the variable logfile, it is opened by a call to the VB subprogram Open, which opens it for output and assigns it the reference number of FILE_NUMBER (set to 1 at the top of the code file) for future calls on this file.

5. Finally, the subprogram Report_on_Model is called with the arguments theProject and FILE_NUMBER. The variable theProject has been typed to be an API object type RPModelElement.

6. In the Report_on_Model subprogram, the calling arguments are passed by value using the keyword ByVal, which makes a local copy of them.

   Note that in the diagram for the Rational Rhapsody API hierarchy (see *The Rational Rhapsody API: A Closer Look*) that all the remaining classes, except for the Application class, inherit from RPModelElement. By using an object of type RPModelElement, you can access objects of subclasses corresponding to hierarchical project elements in a generic fashion. Many of the properties of an RPModelElement have been developed to make its identification and consequent action possible.

   Before proceeding to other steps in Report_on_Model, note the typing of local variables col as RPCollection, and e as RPModelElement. An RPCollection is a collection of RPModelElement objects used for holding and accessing the result of a “get” that obtains multiple or numerous objects satisfying the requirements of the get.
7. After setting the variable `tb` to an empty string, the second line performs the following get:

```vba
Set col = aProject.getNestedElementsRecursive()
```

The `getNestedElementsRecursive()` method, a member of object class `RPModelElements`, is called for the current project, `aProject`, and returns a collection of `RPModelElements` that is accessed through the variable `col`. The method `getNestedElementsRecursive()` retrieves all owned elements of the calling object and places the results in a collection. Because the calling object in this case is a project, `getNestedElementsRecursive()` returns all packages, classes, diagrams, and so on that belong to the project.

The remaining code opens the report file and writes a header to it, followed by a large `for` loop over each element in `col` (for `e in col`). Within the loop, each element is analyzed for its type and is reported accordingly. As previously mentioned, a variety of properties of the element identify it (the element’s `metaClass(e.metaClass)`), making this computed action possible.

**Code Summary of Reporting a Project**

The following is a code summary of the project-reporting process in VB:

```vba
Dim col As RPCollection
Dim e As RPModelElement
Private logFileName As String
Private Const FILE_NUMBER As Integer = 1

' Open file logFileName: FILE_NUMBER'
' Set col = theProject.getNestedElementsRecursive()
' Write header to file=FILE_NUMBER
for e In col
    ' Identify model element e based on e.xxxx properties
    ' Write report of e based on e.xxxx properties
Next
' Close file=FILE_NUMBER
```
Starting and Saving Your Own VB IDE Work

If you want to use the API, spend some time studying the RPYReporter example and the more complex RPYExplorer example. In conjunction with the examples, you can use the online help, which contains the methods and properties of each API class along with descriptions of required arguments.

If you want to use these Rational Rhapsody API examples as a starting point for your own applications, the following sections describe how to perform some common tasks.

Saving the Examples as New Projects

If you want to create your own applications by modifying one of the supplied examples, a good starting point is to save the appropriate example as a new project in its own directory. Note that VB projects consist of a project file (.vbp), a form file for each form (.frm), and module files (.bas). Use the File > Save As options for projects, forms, and modules, and save to a new directory.

Making Your Own New Projects

You might decide to start from scratch and build your own project. When you open Visual Basic, VB displays a default new project environment, complete with a blank form. Alternatively, you can create a new project environment by selecting File > New Project > Standard EXE in the VB IDE.

Once you have started a new project or begun working with an existing one, you can add new forms or modules to a project by right-clicking on the forms folder in the VB Explorer window, then select either Add > Form or Add > Module.

Compiling and Making Your Executables

To create your own applications, you must compile and make your projects into executable files.

In Visual Basic 6.0, compiling is seen as part of making so when you make, you compile. Compiling appears as a separate step only when you test run your project in the Visual Basic IDE by selecting Run > Start With Full Compile.

To make your application’s executable, select File > Make [Project].exe.
Creating Applications with Microsoft Word VB IDE

In addition to the Visual Basic IDE, you can use the Visual Basic editor of Microsoft Word to create applications that use the Rational Rhapsody API.

Follow these steps:

1. Start Microsoft Word.
2. Select File > New to start a new document.
3. In the New dialog box, select the template labeled Blank Document, then click OK.
4. Select File > Save As and save the new, blank document as Word_API.doc.
5. Start a new Word macro by selecting Tools > Macro > Record New Macro.
6. In the Record Macro dialog box, follow these steps:
   a. For the Name field, type “CountPackages.”
   b. For the Store macro in field, select Word_API.doc from the pull-down list.
   c. Click the Keyboard icon.
7. In the Customize Keyboard dialog box, follow these steps:
   a. If it is not there already, move the cursor to the Press new shortcut key field. While holding down the Alt key, type the characters “CP”. When finished, you should see the following entry:

      Alt+C,P

   b. In the Save changes in field, select Word_API.doc.
   c. Click Assign and Close, in that order.

   A small dialog box (shown below) appears to stop and pause the recording of the macro that you are currently recording.

8. Click the small square to stop recording the macro.

You now have a macro named CountPackages saved in the file Word_API.doc that you can trigger at any time within this document with the keyboard sequence Alt+C,P. Currently, the macro has no content.
Specifying the Macro Content

To alter the content of the CountPackages macro, follow these steps:

1. With the file Word_API.doc still loaded in Word, select Tools > Macro > Macros.

2. In the Macros dialog box, follow these steps:
   a. In the Macros in field, select Word_API.doc.
   b. In the list of available macros, select CountPackages.
   c. Click Edit.

   The Microsoft Word Visual Basic IDE opens, so you can edit the contents of the macro CountPackages.


4. In the Reference - Project dialog box, scroll down until you find the reference rhapsody.

5. Mark the rhapsody check box, then click OK. Its location is reported in a small area at the bottom of the dialog box, referencing the RHAPSODY.tlb file located in the Rhapsody installation directory.

6. Insert the following code between the lines Sub CountPackages() and End Sub, but after the comments that appear identifying the macro, date, and author.
Make sure the `projName` path is correct for your Rational Rhapsody installation.

```vba
' Start Rhapsody
Dim rhapApp As Object
Set rhapApp = CreateObject("rhapsody.Application")
' Set Project Name String
Dim projName As String
projName = "C:\Rhapsody40\Samples\CppSamples\Radio\Radio.rpy"
' Open Project
Dim theProject As RModelElement
rhapApp.openProject projName
Set theProject = rhapApp.activeProject
' Get Packages
Dim packages As rhapsody.RPCollection
Set packages = theProject.packages
' Report Packages to Current Word Doc (ThisDocument)
Dim package As rhapsody.RPPackage
For Each package In packages
    ThisDocument.Range.InsertAfter package.name & vbCrLf
Next
' Close Application When Finished
If Not rhapApp Is Nothing Then rhapApp.Quit
```

7. Run the macro by selecting **Run > Run Sub/UserForm**.

If you encounter an error, click **Debug** on the error dialog window to see the offending line of code highlighted.

If the macro works, you will see the packages of the Rhapsody project radio displayed in the document screen of Word. There are three packages:

- guiPkg
- hardwarePkg
- radioPkg

Once you are sure that the macro works, you can execute it in the Word document area by simply typing the macro key sequence (Alt+C,P).
Comments on the Code

The following sequence loads the project:

```vba
Dim rhapApp As Object
Set rhapApp = CreateObject("rhapsody.Application")
Dim projName As String
projName = "C:\Rhapsody\some_project.rpy"
Dim theProject As RPModelElement
rhapApp.openProject projName
Set theProject = rhapApp.activeProject
```

An alternative sequence is as follows:

```vba
Dim rhapApp As rhapsody.Application
Set rhapApp = CreateObject("rhapsody.Application")
Dim projName As String
projName = "C:\Rhapsody\some_project.rpy"
Dim theProject As RPProject
rhapApp.openProject projName
Set theProject = rhapApp.activeProject
```

Note the use of `RPCollection` in the following sequence:

```vba
Dim packages As rhapsody.RPCollection
Set packages = theProject.packages
```

Unlike the RPYReporter example, a “get” method was not used to obtain the elements (in this case, packages). You can use this method for obtaining model elements on one level.

Finally, note the following `For` loop over the packages:

```vba
For Each package In packages
    ThisDocument.Range.InsertAfter package.name & vbCrLf
Next
```

Printing to the Word document is accomplished through the second line of code. The object called `ThisDocument` is the highest level object of Word, representing the document itself. You can see it in the explorer window in the upper, left-hand corner of the VB desktop. Highlight it to examine some its properties.
Modifying the Example to Print Classes

Suppose that instead of printing the names of all the classes in the radio model, you want to print the names of all the classes for a particular package, such as radioPkg. To modify the previous code and save it to another macro, follow these steps:

1. Start a new Word macro by selecting **Tools > Macro > Record New Macro**.
2. In the Record Macro dialog box, follow these steps:
   a. For the **Name** field, type “CountClassesForPackage.”
   b. For the **Store macro in** field, select `Word_API.doc` from the pull-down list.
   c. Click the **Keyboard** icon.
3. In the Customize Keyboard dialog box, follow these steps:
   a. If it is not there already, move the cursor to the **Press new shortcut key** field. While holding down the Alt key, type the “CC” characters. When finished, you should see the following entry:
      
      Alt+C,C
   b. In the **Save changes in** field, select `Word_API.doc`.
   c. Click **Assign** and **Close**, in that order.

   A small dialog box appears to stop and pause the recording of the current macro.
4. Click the small square to stop recording the macro.
5. With the file `Word_API.doc` still loaded in Word, select **Tools > Macro > Macros**.
6. In the Macros dialog box, follow these steps:
   a. In the **Macros in** field, select `Word_API.doc`.
   b. In the list of available macros, select CountClassesForPackage.
   c. Click **Edit**. The focus switches to the VB editor.

   Note the presence of the new, empty CountClassesForPackage subprogram. If you scroll up, you can see the code you created for the CountPackages macro.
7. Cut and paste the code between the lines `Sub CountPackages()` and `End Sub` in the CountPackages macro, but after the comments that appear identifying the macro, date, and author.
8. Replace this section:

```
' Report Packages to Current Word Doc (ThisDocument)
Dim package As rhapsody.RPPackage
For Each package In packages
    ThisDocument.Range.InsertAfter package.name & vbCrLf
Next
' Close Application When Finished
If Not rhapApp Is Nothing Then rhapApp.Quit
```

With this:

```
' Report Classes of Package "radioPkg" to Current Document
Dim package As rhapsody.RPPackage
For Each package In packages
    If (package.name = "radioPkg") Then
        Dim classes As rhapsody.RPCollection
        Dim class As rhapsody.RPClass
        Set classes = package.classes
        For Each class In classes
            ThisDocument.Range.InsertAfter class.name & vbCrLf
        Next
    End If
Next
' Close Application When Finished
If Not rhapApp Is Nothing Then rhapApp.Quit
```

9. Run the macro by selecting Run > Run Sub/UserForm.

If you encounter an error, click Debug on the error dialog window to see the offending line of code highlighted.

If the macro works, you will see the classes of the radioPkg package displayed in the document screen of Word, as follows:

- Frequency
- IDisplay
- ITuner
- Radio
- Waveband
Rhapsody API Interfaces

This section contains reference information describing the classes and methods that comprise the abstract factory interface. For ease of use, the interfaces are presented in alphabetical order.

**Note**

Only the public and protected methods are documented.

The reference material for each of the Rhapsody API interfaces is shown in VB-compliant form (except for the interface class names). This means the following:

- Each COM interface has attributes and methods. In Visual Basic, the attributes are identified as properties.

- The actual identity of the interface classes used in the Rhapsody API varies with the language platform of the client application attempting to interface with the Rhapsody repository. In COM, all interface names start with “I”, such as IRPModelElement. Visual C++ connects directly with the COM tables, which are C++ (or C++-related), and sees the “I”. However, Visual Basic (VB) tries to be user-friendly by avoiding the use of the “I” so, for example, the IRPModelElement interface is RPModelElement in VB. If you open the object browser in the Microsoft Visual Basic IDE, you can see which classes are there and what they are called. Nevertheless, in the reference material, interface objects are identified with the “IRP” prefix and not the “RP” prefix seen in VB.

- Void returns are not shown as `void`—they are simply not shown.

- Pointers are not displayed. In C++, interfaces and collections of interfaces are handled with pointers. VB has no pointers.

- Each method has an implied argument: an instance of its interface referred to as “this.” Thus, the reference on a method of IRPClass will refer to something done to “this Class.”

- String returns and arguments are shown as `String`. For C++, this type is `BSTR`.

- There is only one collection object type: IRPCollection. In the reference material, however, collections are displayed as “xxxxs” where xxxx refers to the object type of the collection and the “s” indicates it is a collection.
Access to VB Properties

The COM API interface consists of data and methods. In Visual Basic, the data is identified as properties. These properties are implemented with invisible operations that enable some properties to be read/write (RW). In other words, the property can be used to set a value in a Rhapsody 6.1 model or retrieve it. Thus, if A is a read/write property, you can set the model value it points to through an "A=..." statement or retrieve it through a "...=A" statement.

**Note**

Not all properties are implemented with write ability. These are identified as read-only (RO).

API Conventions

The Rhapsody Repository API is a set of COM interfaces specified in terms of COM properties and methods, using COM types. The API listings have two syntaxes to describe the various attributes and methods provided by each interface:

- The VB syntax that follows indicates that the function takes a string argument for the property key and then returns a string:
  ```vbnet
  getProperty (propertyKey As String) As String
  ```
  The C/C++ prototype for the same function is:
  ```c
  HRESULT getProperty (String propertyKey, String*** retVal);
  ```
- All interfaces are prefixed with “IRP” (“I” for interface, “RP” for Rhapsody 6.1). For example, the interface for a package is IRPPackage.
- Calls returning multiple objects return the equivalent of a VBA “collection.” To enhance readability, collections are treated as “typed,” for example, “Collection of IRPClasses.” However, in the API, all collections are implemented as “Collection of IRPModelElements.”
- Enumerated types are treated as strings. For example, the getVisibility method of an attribute returns the string “Public,” “Protected,” or “Private.”
The Rhapsody API interfaces are as follows:

- IRPAction Interface
- IRPActor Interface
- IRPAnnotation Interface
- IRPApplication Interface
- IRPArgument Interface
- IRPASCIIFile Interface
- IRPAssociationClass Interface
- IRPAssociationRole Interface
- IRPAttribute Interface
- IRPBlock Interface
- IRPClass Interface
- IRPClassifier Interface
- IRPClassifierRole Interface
- IRCollaboration Interface
- IRCollaborationDiagram Interface
- IRCollection Interface
- IRPComment Interface
- IRPComponent Interface
- IRPComponentDiagram Interface
- IRPComponentInstance Interface
- IRPConfiguration Interface
- IRPConnector Interface
- IRPConstraint Interface
- IRPControlledFile
- IRPDependency Interface
- IRPDeploymentDiagram Interface
- IRPDiagram Interface
- IRPEnumerationLiteral Interface
- IRPEvent Interface
Rhapsody API Interfaces

- IRPEventReceptionInterface
- IRPExecutionOccurrenceInterface
- IRPExternalCodeGeneratorInterface
- IRPExternalCodeGeneratorInvokerInterface
- IRPFileInterface
- IRPFlowInterface
- IRPFlowchartInterface
- IRPFlowItemInterface
- IRPGeneralizationInterface
- IRPGraphEdgeInterface
- IRPGraphElementInterface
- IRPGraphicalPropertyInterface
- IRPGraphNodeInterface
- IRPGuardInterface
- IRPHyperLinkInterface
- IRPImageMap
- IRPInstanceInterface
- IRPInteractionOccurrenceInterface
- IRPInterfaceItemInterface
- IRPLinkInterface
- IRPMessageInterface
- IRPMessagePointInterface
- IRPModelElementInterface
- IRPModuleInterface
- IRPNodeInterface
- IRPObjectModelDiagramInterface
- IRPOperationInterface
- IRPPackageInterface
- IRPPortInterface
- IRPProfileInterface
- IRPProjectInterface
- IRPRelationInterface
IRPAAction Interface

The IRPAAction interface represents the action of a transition in a statechart. It inherits from IRPModelElement.

VB Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>String</td>
<td>RW</td>
<td>The entered body of this action</td>
</tr>
</tbody>
</table>
**IRPActor Interface**

The `IRPActor` interface represents Rhapsody actors. It inherits from `IRPClassifier`.

**IRPAnnotation Interface**

The `IRPAnnotation` interface represents Rhapsody annotations—notes, comments, constraints, and requirements. It inherits from `IRPModelElement`.

**VB Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>anchoredByMe</td>
<td>Rpcollection</td>
<td>RO</td>
<td>The list of model elements that are anchored to the annotation</td>
</tr>
<tr>
<td>body</td>
<td></td>
<td></td>
<td>Deprecated</td>
</tr>
<tr>
<td>body</td>
<td>String</td>
<td>RW</td>
<td>The body text of the remark</td>
</tr>
<tr>
<td>specification</td>
<td>String</td>
<td>RW</td>
<td>The body text for the annotation</td>
</tr>
</tbody>
</table>

**Method Summary**

| addAnchor     | Adds an anchor from the annotation to the specified model element. |
**addAnchor**

**Read method**

**Description**

The `addAnchor` method adds an anchor from the annotation to the specified model element.

**Visual Basic**

**Syntax**

```visualbasic
addAnchor(target As RPModelElement)
```

**Arguments**

- `target`
  
  The model element to which to anchor the annotation

**C/C++ Prototype**

```c
HRESULT addAnchor (IRPModelElement* target)
```

**Return Value**

- HRESULT (0 for success, or a signed integer error code)
IRPApplication Interface

The application is the top-level object of the Rhapsody object model, which represents the Rhapsody application shell. It conceptually provides the functionality available through the Rhapsody menu bars. Initially, the application object exposes the minimal set of functionality required to open a project.

When you use VB or VC++ to interface to the COM API, the IRPApplication object needs to be created before any other API interface objects are used. However, if you use the Rhapsody VBA interface, you are automatically connected to the IRPApplication object.
RB Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BuildNo</td>
<td>CString</td>
<td>RO</td>
<td>The current build number</td>
</tr>
<tr>
<td>Language</td>
<td>String</td>
<td>RW</td>
<td>The current language setting</td>
</tr>
<tr>
<td>OMROOT</td>
<td>String</td>
<td>RO</td>
<td>The value for OMROOT</td>
</tr>
<tr>
<td>SerialNo</td>
<td>CString</td>
<td>RO</td>
<td>The serial number</td>
</tr>
<tr>
<td>ToolSet</td>
<td>CString</td>
<td>RO</td>
<td>The current tool setting (demo, Designer, and so on)</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>activeProject</td>
<td>Returns a pointer to the active (open) project</td>
</tr>
<tr>
<td>addToModel</td>
<td>Adds a Rhapsody unit located in the specified file to the current model with or without descendant elements</td>
</tr>
<tr>
<td>addToModelByReference</td>
<td>Adds the Rhapsody unit you specify to your model as a reference.</td>
</tr>
<tr>
<td>addToModelFromURL</td>
<td>Adds a Rhapsody unit located at the specified URL to the current model</td>
</tr>
<tr>
<td>arcCheckOut</td>
<td>Checks out files from the CM archive into the model</td>
</tr>
<tr>
<td>build</td>
<td>Builds the application</td>
</tr>
<tr>
<td>checkIn</td>
<td>Checks in the specified unit within the model into the CM archive you have already connected to (using connectToArchive)</td>
</tr>
<tr>
<td>checkModel</td>
<td>Checks the current model</td>
</tr>
<tr>
<td>checkOut</td>
<td>Refreshes a unit in the model by checking it out from the CM archive</td>
</tr>
<tr>
<td>connectToArchive</td>
<td>Connects the Rhapsody 6.1 project to the specified CM archive</td>
</tr>
<tr>
<td>createNewProject</td>
<td>Creates a new project named &lt;projectName&gt; in &lt;projectLocation&gt;</td>
</tr>
<tr>
<td>enterAnimationCommand</td>
<td>Specifies the command to begin animation</td>
</tr>
<tr>
<td>errorMessage</td>
<td>Returns the most recent error message</td>
</tr>
<tr>
<td>forceRoundtrip</td>
<td>Forces a roundtrip of the code back into the Rhapsody 6.1 model, and vice versa</td>
</tr>
<tr>
<td>generate</td>
<td>Generates code for the active configuration of the active component</td>
</tr>
<tr>
<td>getDiagramOfSelectedElement</td>
<td>Retrieves the diagram of the current element</td>
</tr>
</tbody>
</table>
### Rhapsody API Interfaces

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getErrorMessage</code></td>
<td>Returns the most recent error message</td>
</tr>
<tr>
<td><code>getListOfFactoryProperties</code></td>
<td>Retrieves the list of properties in the <code>&lt;lang&gt;_factory.prp</code> file</td>
</tr>
<tr>
<td><code>getListOfSelectedElements</code></td>
<td>Returns the collection of model elements</td>
</tr>
<tr>
<td><code>getListOfSiteProperties</code></td>
<td>Retrieves the list of properties in the <code>&lt;lang&gt;_site.prp</code> file</td>
</tr>
<tr>
<td><code>getSelectedElement</code></td>
<td>Retrieves the current model element</td>
</tr>
<tr>
<td><code>getTheExternalCodeGeneratorInvoker</code></td>
<td>Retrieves the invoker for the external code generator</td>
</tr>
<tr>
<td><code>highlightByHandle</code></td>
<td>Highlights an element, given its handle</td>
</tr>
<tr>
<td><code>highlightElement</code></td>
<td>Highlights the specified element</td>
</tr>
<tr>
<td><code>importClasses</code></td>
<td>Imports classes according to the reverse engineering setting stored in the current configuration</td>
</tr>
<tr>
<td><code>make</code></td>
<td>Builds the current component following the current configuration</td>
</tr>
<tr>
<td><code>openProject</code></td>
<td>Opens a Rhapsody 6.1 project</td>
</tr>
<tr>
<td><code>openProjectFromURL</code></td>
<td>Opens the Rhapsody 6.1 product at the specified URL</td>
</tr>
<tr>
<td><code>openProjectWithLastSession</code></td>
<td>Opens the project using the settings from the previous Rhapsody 6.1 session</td>
</tr>
<tr>
<td><code>openProjectWithoutSubUnits</code></td>
<td>Opens the Rhapsody 6.1 project without subunits</td>
</tr>
<tr>
<td><code>quit</code></td>
<td>Closes the active Rhapsody 6.1 project</td>
</tr>
<tr>
<td><code>rebuild</code></td>
<td>Rebuilds the application</td>
</tr>
<tr>
<td><code>refreshAllViews</code></td>
<td>Refreshes all the views</td>
</tr>
<tr>
<td><code>regenerate</code></td>
<td>Regenerates the active configuration of the active component</td>
</tr>
<tr>
<td><code>report</code></td>
<td>Generates a report in ASCII or RTF into the specified file</td>
</tr>
<tr>
<td><code>roundtrip</code></td>
<td>Roundtrips code changes back into the open model</td>
</tr>
<tr>
<td><code>setComponent</code></td>
<td>Sets the current component for the open project</td>
</tr>
<tr>
<td><code>setConfiguration</code></td>
<td>Sets the current configuration for the open project</td>
</tr>
<tr>
<td><code>setLog</code></td>
<td>Creates a log file that records all the information that is normally displayed in the Rhapsody 6.1 output window</td>
</tr>
<tr>
<td><code>version</code></td>
<td>Returns the version of Rhapsody 6.1 that corresponds to the current COM API version</td>
</tr>
</tbody>
</table>
activeProject

Read method

Description

The `activeProject` method returns a pointer to the active (open) project.

Visual Basic

Syntax

```
activeProject() As RPProject
```

Return Value

A pointer to the current open project (an `RPProject`)

C/C++ Prototype

```
HRESULT activeProject (IRPProject** pVal)
```

Return Value

`HRESULT` (0 for success, or a signed integer error code)

addToModel

Write method

Description

The `addToModel` method adds a Rhapsody unit located in the specified file to the current model with or without descendant elements.

Note: When adding a file with descendants, all the file subunits must be in the unit directory of the project before you issue the command.

Visual Basic

Syntax

```
addToModel (filename As String, withDescendant As Long)
```

Arguments

filename
The full file name of the file that contains the unit to be added

withDescendants
Specifies whether to bring in descendants of the unit to be added to the model
C/C++ Prototype

HRESULT addToModel (BSTR filename, long withDescendant)

Return Value

HRESULT (0 for success, or a signed integer error code)
addToModelByReference

The method addToModelByReference adds the Rhapsody unit you specify to your model as a reference.

Syntax

addToModelByReference (filename As String)

Arguments

filename

The name of the file that contains the unit to be added. The full path to the file must be specified.

Return Value

HRESULT (0 for success, or a signed integer error code)

Example

Sub addJavadocProfile()
    Dim app As Object
    Set app = GetObject(, "Rhapsody.Application")
    On Error GoTo aa
    app.addToModelByReference ("C:\temp\JavaDocProfile.sbs")
    Exit Sub
aa:
    MsgBox errorMessage
End Sub
**addToModelFromURL**

*Write method*

**Description**

The `addToModelFromURL` method adds a Rhapsody unit located at the specified URL to the current model. This method is used to support the Webify Toolkit.

**Visual Basic**

**Syntax**

```vbnet
addToModelFromURL (url As String)
```

**Arguments**

- `url`
  - The URL that contains the unit to be added

**C/C++ Prototype**

```c
HRESULT addToModelFromURL (BSTR url)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)

---

**arcCheckOut**

*Write method*

**Description**

The `arcCheckOut` method checks out files from the configuration management (CM) archive into the model.

**Note:** The difference between `arcCheckOut` and `checkOut` is that `arcCheckOut` refers to files in the archive, whereas `checkOut` refers to units in the model. To add new units to the model, use `arcCheckOut`. The method `checkOut` is intended to refresh elements already existing in the model.

**Visual Basic**

**Syntax**

```vbnet
arcCheckOut (filename As String, label As String, isLocked As Long, isRecursive As Long)
```

**Arguments**

- `filename`
  -
Specifies the name of the file.

label

Specifies the revision or label to be checked out. If this is set to NULL, the last revision on the main trunk (the default) will be checked out.

isLocked

Specifies whether the file is locked. The possible values are as follows:

1--Designates that a writable file be checked out and the archive locked from other checkouts of the file.

0--The file is checked out as read-only and the archive not locked to other checkouts.

isRecursive (1 or 0)

If this is set to 1, the file and all the other elements that it contains are checked out.

C/C++ Prototype

HRESULT arcCheckOut (BSTR filename, BSTR label, long isLocked, long isRecursive)

Return Value

HRESULT (0 for success, or a signed integer error code)
build

**Note**
Currently, this method has not been implemented.

**Read method**

**Description**
The `build` method builds the application.

**Visual Basic**

**Syntax**

```vbnet
build()
```

**C/C++ Prototype**

```c
HRESULT build()
```

**Return Value**

```c
HRESULT (0 for success, or a signed integer error code)
```

checkIn

**Read method**

**Description**
The `checkIn` method checks in the specified unit within the model into the configuration management (CM) archive you have already connected to (using `connectToArchive`).

**Visual Basic**

**Syntax**

```vbnet
checkIn (unitName As String, label As String, isLocked As Long, isRecursive As Long, description As String)
```
Arguments

unitName
The name of the unit.
label
The label to apply when you check in the file to the archive. If it is not needed, set this argument to NULL.
isLocked (1 or 0)
Specifies whether to lock the archive after checkin.
isRecursive
If set to 1, check in the unit and all the elements contained in it.
description
The description to add to the unit when you check it in to the archive.

C/C++ Prototype

HRESULT checkIn (BSTR unitName, BSTR label,
                   long isLocked, long isRecursive, BSTR description)

Return Value

HRESULT (0 for success, or a signed integer error code)
checkModel

Read method

Description

The checkModel method checks the current model. This is equivalent to the Rhapsody 6.1 command Tools > Check Model for the current configuration.

Visual Basic

Syntax

checkModel()

C/C++ Prototype

HRESULT checkModel()

Return Value

HRESULT (0 for success, or a signed integer error code)
checkOut

Write method

Description

The **checkOut** method refreshes a unit in the model by checking it out from the CM archive.

Visual Basic

Syntax

```visual-basic
checkOut (unitName As String, label As String, isLocked As Long, isRecursive As Long)
```

Arguments

- **unitName**
  The name of the unit.

- **label**
  The revision or label to be checked out. If you set this to NULL, the last revision on the main trunk (the default) will be checked out.

- **isLocked**
  Specifies whether to lock the archive after checkout. The possible values are as follows:
  - 1--Designates that a writable unit is to be checked out and the archive locked from other checkouts of the unit.
  - 0--The unit is checked out as read-only and the archive not locked to other checkouts.

- **isRecursive**
  If this is set to 1, check out the unit and all the elements contained in it.

C/C++ Prototype

```c
HRESULT checkOut (BSTR unitName, BSTR label, long isLocked, long isRecursive)
```

Return Value

- **HRESULT** (0 for success, or a signed integer error code)
connectToArchive

**Read method**

**Description**

The `connectToArchive` method connects the Rhapsody 6.1 project to the specified CM archive. This operation is necessary only for the following cases:

- There is no current association in the project.
- The association needs to be modified.

**Visual Basic**

**Syntax**

```vbnet
connectToArchive (archivePath As String)
```

**Arguments**

- `archivePath`
  The path to location of archive

**C/C++ Prototype**

```c
HRESULT connectToArchive (BSTR archivePath)
```

**Return Value**

- `HRESULT (0 for success, or a signed integer error code)`
createNewProject

Write method

Description

The `createNewProject` method creates a new project named `<projectName>` in `<projectLocation>`. You should call this operation before a project has been opened, or after a project has been saved.

Note that helper applications might not close the current document. This means that the `createNewProject` method should not be used in a VBA macro that you specify as a helper.

Visual Basic

Syntax

```vbnet
createNewProject (projectLocation As String, projectName As String)
```

Arguments

- `projectLocation`
The location of the project
- `projectName`
The name of the project

C/C++ Prototype

```c
HRESULT createNewProject (BSTR projectLocation, BSTR projectName)
```

Return Value

```c
HRESULT (0 for success, or a signed integer error code)
```
**deferredAddToModel**

**Write method**

**Description**

The `deferredAddToModel` method TBS.

**Visual Basic**

**Syntax**

```visual-basic
deferedAddToModel(filename As String, 
                    withDescendants As Long, orijPrjId As String, 
                    eraseDir As Long)
```

**Arguments**

- `filename`:
  The full name of the file that contains the unit to be added
- `withDescendants`:
  Specifies whether to bring in descendants of the unit to be added to the model
- `orijPrjId`:
  The project ID
- `eraseDir`:
  Specifies whether to delete the directory after the unit has been added to the model

**C/C++ Prototype**

```c
HRESULT deferredAddToModel (BSTR filename, 
                            long withDescendants, BSTR orijPrjId, long eraseDir);
```

**Return Value**

- `HRESULT` (0 for success, or a signed integer error code)
**enterAnimationCommand**

**Read method**

**Description**

The `enterAnimationCommand` method specifies the command to begin animation.

**Visual Basic**

**Syntax**

`enterAnimationCommand (command As String)`

**Arguments**

- `command`
  
  The animation command

**C/C++ Prototype**

`HRESULT enterAnimationCommand (BSTR command)`

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
errorMessage

Read method

Description

The `errorMessage` method returns the most recent error message.

Visual Basic

Syntax

```vbnet
errorMessage() As String
```

Return Value

A pointer to the most recent error message (a string)

C/C++ Prototype

```c
HRESULT errorMessage (BSTR* __MIDL_0016)
```

Arguments

- `BSTR*`
  - A pointer to most recent error message

Return Value

- `HRESULT` (0 for success, or a signed integer error code)
**forceRoundtrip**

**Read method**

**Description**

The `forceRoundtrip` method forces a roundtrip of the code back into the Rhapsody 6.1 model, and vice versa.

**Visual Basic**

**Syntax**

```vbnet
forceRoundtrip()
```

**C/C++ Prototype**

```c
HRESULT forceRoundtrip()
```

**Return Value**

HRESULT (0 for success, or a signed integer error code)

---

**generate**

**Read method**

**Description**

The `generate` method generates code for the active configuration of the active component.

**Visual Basic**

**Syntax**

```vbnet
generate()
```

**C/C++ Prototype**

```c
HRESULT generate()
```

**Return Value**

HRESULT (0 for success, or a signed integer error code)
getDiagramOfSelectedElement

Read method

Description

The `getDiagramOfSelectedElement` method retrieves the diagram of the current element.

Visual Basic

Syntax

```visualbasic
getDiagramOfSelectedElement() As RPDiagram
```

Return Value

The `RPDiagram`

C/C++ Prototype

```c
HRESULT getDiagramOfSelectedElement (IRPDiagram** pVal)
```

Return Value

`HRESULT (0 for success, or a signed integer error code)`
**getErrorMessage**

**Read method**

**Description**

The `getErrorMessage` method returns the most recent error message.

**Visual Basic**

**Syntax**

```
getErrorMessage(__MIDL_0014 As String) As String
```

**Return Value**

A pointer to the most recent error message (a string)

**C/C++ Prototype**

```
HRESULT getErrorMessage (BSTR* __MIDL_0014)
```

**Arguments**

- `BSTR*`
  - A pointer to most recent error message

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
getListOfFactoryProperties

Note
Currently, this method has not been implemented.

Read method

Description
The `getListOfFactoryProperties` method returns the list of properties in the `<lang>_factory.prp` file.

Visual Basic

Syntax
```
getListOfFactoryProperties() As RPCollection
```

Return Value
The list of properties defined in the `<lang>_factory.prp` file

C/C++ Prototype
```
HRESULT getListOfFactoryProperties (IRPCollection** pVal)
```

Return Value
`HRESULT (0 for success, or a signed integer error code)`
**getListOfSelectedElements**

**Read method**

**Description**

The `getListOfSelectedElements` method returns a collection of model elements.

In Version 4.1, this method was modified as follows:

- If the instance is selected in the context of an OMD, the method returns an `IRPInstance` instead of `IRPClass` or `IRPActor`. See “IRPInstance Interface” for more information on this interface.

- If a link is selected in the context of an OMD, the method returns an `IRPLink` instead of `IRPRelation`. See “IRPLink Interface” for more information on this interface.

- If an instance is selected in the context of a sequence diagram, the method returns an `IRPClassifierRole` instead of `IRPClass`. See “IRPClassifierRole Interface” for more information on this interface.

**Visual Basic**

**Syntax**

```vbnet
getListOfSelectedElements () As RPCollection
```

**Return Value**

The collection of elements

**C/C++ Prototype**

```c
HRESULT getListOfSelectedElements (IRPCollection** pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
getListOfSiteProperties

Note
Currently, this method has not been implemented.

Read method

Description
The `getListOfSiteProperties` method returns the list of properties in the `<lang>_site.prp` file.

Visual Basic

Syntax

```vbnet
getListOfSiteProperties() As RPCollection
```

Return Value
The list of properties defined in the `<lang>_site.prp` file

C/C++ Prototype

```c
HRESULT getListOfSiteProperties (IRPCollection** pVal)
```

Return Value

```c
HRESULT (0 for success, or a signed integer error code)
```
getSelectedElement

Read method

Description

The `getSelectedElement` method retrieves the current model element.

In Version 4.1, this method was modified as follows:

- If the instance is selected in the context of an OMD, the method returns an `IRPInstance` instead of `IRPClass` or `IRPActor`. See “IRPInstance Interface” for more information on this interface.
- If a link is selected in the context of an OMD, the method returns an `IRPLink` instead of `IRPRelation`. See “IRPLink Interface” for more information on this interface.
- If an instance is selected in the context of a sequence diagram, the method returns an `IRPClassifierRole` instead of `IRPClass`. See “IRPClassifierRole Interface” for more information on this interface.

Visual Basic

Syntax

```vbscript
getSelectedElement() As RPModelElement
```

Return Value

The current model element

C/C++ Prototype

```c
HRESULT getSelectedElement (IRPModelElement** pVal)
```

Return Value

`HRESULT (0 for success, or a signed integer error code)
```

VBA Example

The following example assumes that a link is selected.

```vbscript
Dim m As RPModelElement
Dim link as RPLink
Dim fromCls as RPClass
Dim toCls as RPClass
Dim from as RPInstance
Dim to as RPInstance
Dim rel as RPRelation
Set m = getSelectedElement
If m.metaClass = "Link" then
```
link = m
from = link.from
   to = link.to
fromCls = from.otherClass
toCls = to.otherClass
rel = link.instantiates

'Variable content:
'link points to the selected link.
'from points to the "source" instance.
'to points to the "target" instance.
'fromCls points to the class of the "source" instance.
'toCls points to the class of the "target" instance.
'rel points to the relation instantiated by the link.

MsgBox m.name & " is a link from instance " & from.name & " of class " + clsFrom.name & " to instance " & to.name & " of class " + toCls.name & " which instantiates the " & rel.name & " relation."

End If
getTheExternalCodeGeneratorInvoker

Read method

Description

The `getTheExternalCodeGeneratorInvoker` method returns the invoker for the external code generator.

Visual Basic

Syntax

```visualbasic
getTheExternalCodeGeneratorInvoker() As RPExternalCodeGeneratorInvoker
```

Return Value

The `RPExternalCodeGeneratorInvoker` singleton. The external code generator queries the application for this interface.

C/C++ Prototype

```c
HRESULT getTheExternalCodeGeneratorInvoker (IRPExternalCodeGeneratorInvoker** pVal)
```

Return Value

`HRESULT` (0 for success, or a signed integer error code)

highlightByHandle

Read method

Description

The `highlightByHandle` method highlights the specified model element, given its handle.

The rules for developing the handle for each element type are as follows:

1. The metaclass in the beginning is the value of the metaClass property.
2. The GUID at the end of the name is the value of the GUID property.
3. The structure of the name is as follows:

   `<Package name>::<Class name>.<Element name>`

In this syntax:

   - `<Package name>` is the full path of the package of the element (for example, P1::P2).
– `<Class full name>` is the full path of the class of the element (for example, C1::C2).
– `<Element name>` is the name of the element.

See the section “Example” for a code example that uses this method.

**Visual Basic**

**Syntax**

highlightByHandle (strHandle As String)

**Arguments**

strHandle

The handle to the element to highlight. Call the method with this argument using the following string:

"(<metaclass>)<FullPathName>(<GUID>)"

**C/C++ Prototype**

HRESULT highlightByHandle (BSTR strHandle)

**Return Value**

HRESULT (0 for success, or a signed integer error code)

**Example**

Dim proj As RProject
Dim m As RModelElement
Dim str As String

Dim app As Object
set app = GetObject(, "Rhapsody.Application")

On Error GoTo aa

Set proj = getProject
Set m = proj.findNestedElementRecursive("state_0", "State")
str = "(" & m.metaClass & ")" & m.getFullPathName & "(" & m.GUID & ")"
app.highlightByHandle (str)

Exit Sub

aa:

MsgBox errorMessage
**highLightElement**  
Read method  

**Description**  
Highlights the specified element.

**Visual Basic**  

**Syntax**  

```vbnet
highLightElement (val As RPModelElement)
```

**Arguments**  

`val`  
The element to highlight

**C/C++ Prototype**  

```c
HRESULT highLightElement (IRPModelElement* val)
```

**Return Value**  

`HRESULT (0 for success, or a signed integer error code)`
importClasses

Write method

Description

The `importClasses` method imports classes according to the reverse engineering setting stored in the current configuration. This is equivalent to selecting the Rhapsody 6.1 command **Tools > Reverse Engineering**.

Visual Basic

Syntax

```vbnet
importClasses()
```

C/C++ Prototype

```c
HRESULT importClasses ()
```

Return Value

`HRESULT` (0 for success, or a signed integer error code)

make

Read method

Description

The `make` method builds the current component following the current configuration.

Visual Basic

Syntax

```vbnet
make()
```

C/C++ Prototype

```c
HRESULT make ()
```

Return Value

`HRESULT` (0 for success, or a signed integer error code)
openProject

Read method

Description

The openProject method opens a Rhapsody 6.1 project.

Note that helper applications might not close the current document. This means that you should not use the openProject method in a VBA macro that you specify as a helper:

Visual Basic

Syntax

openProject (filename As String) As RPProject

Arguments

filename

The name of the file that contains the project

Return Value

A pointer to the opened project (an RPProject)

C/C++ Prototype

HRESULT openProject (BSTR filename, IRPProject** pVal)

Return Value

HRESULT (0 for success, or a signed integer error code)
openProjectFromURL

Write method

Description

The openProjectFromURL method opens the Rhapsody 6.1 product at the specified URL. This method is used to support the Webify Toolkit.

Visual Basic

Syntax

openProjectFromURL (url As String)

Arguments

url

The URL of the project to open

C/C++ Prototype

HRESULT openProjectFromURL (BSTR url)

Return Value

HRESULT (0 for success, or a signed integer error code)
**openProjectWithLastSession**

**Write method**

**Description**

The `openProjectWithLastSession` method opens the project using the settings from the previous Rhapsody 6.1 session.

**Visual Basic**

**Syntax**

```vb
openProjectWithLastSession (filename As String) As RPProject
```

**Arguments**

- `filename`  
  The name of the project to open

**Return Value**

The RPProject that was opened

**C/C++ Prototype**

```c
HRESULT openProjectWithLastSession (BSTR filename, IRPProject** pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
openProjectWithoutSubUnits

Write method

Description

The **openProjectWithoutSubUnits** method opens the Rhapsody 6.1 project without subunits.

Visual Basic

Syntax

```vbnet
openProjectWithoutSubUnits (filename As String)
    As RPProject
```

Arguments

- `filename` - The name of the project to open

C/C++ Prototype

```c
HRESULT openProjectWithoutSubUnits (BSTR filename,
    IRPProject** pVal)
```

Return Value

- `HRESULT` (0 for success, or a signed integer error code)
**quit**

**Read method**

**Description**

The `quit` method closes the active Rhapsody 6.1 project.

Note that helper applications might not close the current document. This means that you should not use the `quit` method in a VBA macro that you specify as a helper:

**Visual Basic**

**Syntax**

`quit()`

**C/C++ Prototype**

`HRESULT quit()`

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
rebuild

Note
Currently, this method has not been implemented.

Read method

Description
The rebuild method rebuilds the application.

Visual Basic

Syntax
rebuild()

C/C++ Prototype
HRESULT rebuild()

Return Value
HRESULT (0 for success, or a signed integer error code)
refreshAllViews

Read method

Description

The refreshAllViews method refreshes the views.

Visual Basic

Syntax

refreshAllViews()

C/C++ Prototype

HRESULT refreshAllViews()

Return Value

HRESULT (0 for success, or a signed integer error code)

regenerate

Read method

Description

The regenerate method regenerates the active configuration of the active component.

Visual Basic

Syntax

regenerate()

C/C++ Prototype

HRESULT regenerate()

Return Value

HRESULT (0 for success, or a signed integer error code)
**report**

**Read method**

**Description**

The `report` method generates a report in ASCII or RTF into the specified file. The report is generated for the elements found in the scope of the current component.

**Visual Basic**

**Syntax**

```
report (format As String, outputFileName As String)
```

**Arguments**

- `format`  
  The file format. The possible values are as follows:  
  ASCII  
  RTF

- `outputFileName`  
  The name of the output file, including the path.

**C/C++ Prototype**

```
HRESULT report (BSTR format, BSTR outputFileName)
```

**Return Value**

- `HRESULT` (0 for success, or a signed integer error code)
**roundtrip**

**Write method**

**Description**

The `roundtrip` method roundtrips code changes back into the open model.

**Visual Basic**

**Syntax**

```vbnet
roundtrip()
```

**C/C++ Prototype**

```c
HRESULT roundtrip()
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
**setComponent**

*Write method*

**Description**

The `setComponent` method sets the current component for the open project.

**Visual Basic**

**Syntax**

```
setComponent (component As String)
```

**Arguments**

- `component`
  
  The name of component in the project

**C/C++ Prototype**

```
HRESULT setComponent (BSTR component)
```

**Return Value**

- `HRESULT (0 for success, or a signed integer error code)`
**setConfiguration**

**Write method**

**Description**

The `setConfiguration` method sets the current configuration for the open project.

**Note:** This method fails if the configuration is not found within the current component. Therefore, you should call `setComponent` before `setConfiguration`.

**Visual Basic**

**Syntax**

```vbnet
setConfiguration (configuration As String)
```

**Arguments**

`configuration`

The name of the configuration in the project. This refers to the simple name of the configuration, not the full name, i.e., not `packageA::componentB::configC`.

**C/C++ Prototype**

```c
HRESULT setConfiguration (BSTR configuration)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
**setLog**

*Write method*

**Description**

The `setLog` method creates a log file that records all the information that is normally displayed in the Rhapsody 6.1 output window.

**Visual Basic**

**Syntax**

```vbnet
setLog (logFile As String)
```

**Arguments**

- **LogFile**
  - The name of the log file, including the path

**C/C++ Prototype**

```c
HRESULT setLog (BSTR logFile)
```

**Return Value**

- `HRESULT (0 for success, or a signed integer error code)`
version

Read method

Description

The version method returns the version of Rhapsody 6.1 that corresponds to the current COM API version.

Visual Basic

Syntax

version() As String

Return Value

The version of Rhapsody that corresponds to the COM API version

C/C++ Prototype

HRESULT version (BSTR* __MIDL_0015)

Return Value

HRESULT (0 for success, or a signed integer error code)
Rhapsody API Interfaces

IRPArgument Interface

The IRPArgument interface represents an argument of an operation or an event. It inherits from IRPVariable.

VB Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>argumentDirection</td>
<td>String</td>
<td>RW</td>
<td>The direction of the argument (In, Out, or InOut)</td>
</tr>
<tr>
<td>declaration</td>
<td>String</td>
<td>RW</td>
<td>A string that represents an inline declaration of this argument</td>
</tr>
<tr>
<td>defaultValue</td>
<td>String</td>
<td>RW</td>
<td>The default value of this argument</td>
</tr>
<tr>
<td>type</td>
<td>RPType</td>
<td>RW</td>
<td>The type of this argument</td>
</tr>
</tbody>
</table>

Method Summary

| setNameDeclaration | Sets the C++ type declaration for this argument |
**setTypeDeclaration**

*Write method*

**Description**

The `setTypeDeclaration` method sets the C++ type declaration for this argument.

**Visual Basic**

**Syntax**

```vbnet
setTypeDeclaration (newVal As String)
```

**Arguments**

- **NewVal**
  
  The C++ type declaration for this argument

**C/C++ Prototype**

```c
HRESULT setTypeDeclaration (BSTR newVal)
```

**Return Value**

```c
HRESULT (0 for success, or a signed integer error code)
```
IRPASCIIFile Interface

The IRPASCIIFile interface represents a disk file that you can open, close, and write to. It is a top-level interface in the Rhapsody 6.1 object model.

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>close</td>
<td>Closes a file</td>
</tr>
<tr>
<td>open</td>
<td>Opens a file</td>
</tr>
<tr>
<td>write</td>
<td>Writes to the specified file</td>
</tr>
</tbody>
</table>

close

Write method

Description

The close method closes the file.

Visual Basic

Syntax

close()

C/C++ Prototype

HRESULT close

Return Value

HRESULT (0 for success, or a signed integer error code)
open

Write method

Description

The open method opens a file.

Visual Basic

Syntax

open (filename As String)

Arguments

filename

The name of file to open

C/C++ Prototype

HRESULT open (BSTR filename)

Return Value

HRESULT (0 for success, or a signed integer error code)
write

Write method

Description

The write method writes to the specified file.

Visual Basic

Syntax

write (data As String)

Arguments

Data

The ASCII string data to write to the disk file

C/C++ Prototype

HRESULT write (BSTR data)

Return Value
Rhapsody Interfaces

HRESULT (0 for success, or a signed integer error code)

**IRPAssociationClass Interface**

The **IRPAssociationClass** interface represents a Rhapsody 6.1 association (bi-directional, directed, composition, or aggregation). **IRPAssociationClass** inherits from the **IRPClass**.

**VB Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>end1</td>
<td>RPRelation</td>
<td>RO</td>
<td>The first end of the association line</td>
</tr>
<tr>
<td>end2</td>
<td>RPRelation</td>
<td>RO</td>
<td>The second end of the association line</td>
</tr>
</tbody>
</table>

**IRPAssociationRole Interface**

The **IRPAssociationRole** interface represents a channel or relation through which objects in a collaboration communicate. This object is meaningful only for collaborations displayed in collaboration diagrams. **IRPAssociationRole** inherits from the **IRPModelElement**.

**VB Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>roleType</td>
<td>String</td>
<td>RO</td>
<td>The role type (specified or unspecified)</td>
</tr>
</tbody>
</table>

**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getClassifierRoles</td>
<td>Returns a collection of <strong>IRPClassifierRoles</strong> linked by the current association role</td>
</tr>
<tr>
<td>getFormalRelations</td>
<td>Returns a collection of <strong>IRPRelations</strong> for the current association role</td>
</tr>
</tbody>
</table>
**getClassifierRoles**

Read method

**Description**

The `getClassifierRoles` method returns a collection of `IRPClassifierRoles` linked by the current association role.

Note that an association role in a collaboration diagram is always bidirectional.

**Visual Basic**

**Syntax**

```vbnet
getClassifierRoles () As RPCollection
```

**Return Value**

A collection of classifier roles

**C/C++ Prototype**

```c
HRESULT getClassifierRoles ( 
   IRPCollection** classifierRoles)
```

**Return Value**

```c
HRESULT (0 for success, or a signed integer error code)
```
**getFormalRelations**

**Read method**

**Description**

The `getFormalRelations` method returns a collection of `IRPRelations` for the current association role. Pass one of the following values to the method:

- 0—Get the unspecified relations.
- 1—Get the directional relations.
- 2—Get the bidirectional relations.

**Visual Basic**

**Syntax**

```vbnet
getFormalRelations() As RPCollection
```

**Return Value**

A collection of `RPRelations`

**C/C++ Prototype**

```c
HRESULT getFormalRelations (  
  IRPCollection** classifierRoles)
```

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
IRPAttribute Interface

The IRPAttribute interface represents a class attribute. It inherits from IRPVariable.

**VB Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>declaration</td>
<td>String</td>
<td>RW</td>
<td>The declaration of this attribute. For an inline declaration, this is an uninterpreted string.</td>
</tr>
<tr>
<td>defaultValue</td>
<td>String</td>
<td>RW</td>
<td>The default value of this attribute, if one has been defined.</td>
</tr>
<tr>
<td>isConstant</td>
<td>Long</td>
<td>RW</td>
<td>A flag that indicates whether the attribute is read-only or modifiable.</td>
</tr>
<tr>
<td>isOrdered</td>
<td>Long</td>
<td>RW</td>
<td>A flag that specifies whether the order of the reference type items is significant.</td>
</tr>
<tr>
<td>isReference</td>
<td>Long</td>
<td>RW</td>
<td>A flag that specifies whether the attribute is referenced as a reference (such as a pointer (*) or an address (&amp;) in C++).</td>
</tr>
<tr>
<td>isStatic</td>
<td>Long</td>
<td>RW</td>
<td>A flag that indicates whether this attribute is a static class attribute. Static status implies that the attribute belongs to the class as a whole rather than to an individual instance.</td>
</tr>
<tr>
<td>multiplicity</td>
<td>String</td>
<td>RW</td>
<td>The multiplicity of the attribute. If this is greater than 1, use the isOrdered property to specify whether the order of the reference type items is significant.</td>
</tr>
<tr>
<td>type</td>
<td>RPType</td>
<td>RW</td>
<td>The type of this attribute. For Rhapsody predefined types, this is a reference to that type.</td>
</tr>
<tr>
<td>visibility</td>
<td>String</td>
<td>RW</td>
<td>The visibility of this attribute (public, protected, or private).</td>
</tr>
</tbody>
</table>
IRPBlock Interface

The IRPBlock interface was removed in version 7.2 of Rhapsody.

Use IRPInstance instead.
IRPClass Interface

The IRPClass interface represents Rhapsody 6.1 classes. It inherits from IRPClassifier.

VB Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isActive</td>
<td>Long</td>
<td>RW</td>
<td>Indicates whether this class is an active class.</td>
</tr>
<tr>
<td>isBehaviorOverridden</td>
<td>Long</td>
<td>RW</td>
<td>Indicates whether the statechart of the subclass overrides the statechart of this class. A statechart is <em>not</em> inherited.</td>
</tr>
<tr>
<td>isComposite</td>
<td>Long</td>
<td>RO</td>
<td>Indicates whether this class is a composite class.</td>
</tr>
<tr>
<td>isReactive</td>
<td>Long</td>
<td>RO</td>
<td>Indicates whether this class has a statechart that is, it's a reactive class).</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addClass</td>
<td>Adds a class to the current class</td>
</tr>
<tr>
<td>addConstructor</td>
<td>Adds a constructor to the current class</td>
</tr>
<tr>
<td>addDestructor</td>
<td>Adds a destructor to the current class</td>
</tr>
<tr>
<td>addEventReception</td>
<td>Adds an event reception to the current class</td>
</tr>
<tr>
<td>addLink</td>
<td>Adds a link between two objects to the current class</td>
</tr>
<tr>
<td>addReception</td>
<td>Adds a reception to the current class</td>
</tr>
<tr>
<td>addSuperclass</td>
<td>Adds a superclass to the current class</td>
</tr>
<tr>
<td>addTriggeredOperation</td>
<td>Adds a triggered operation to the current class</td>
</tr>
<tr>
<td>addType</td>
<td>Adds a type to the current class</td>
</tr>
<tr>
<td>deleteClass</td>
<td>Deletes a class from the current class</td>
</tr>
<tr>
<td>deleteConstructor</td>
<td>Deletes a constructor from the current class</td>
</tr>
<tr>
<td>deleteDestructor</td>
<td>Deletes a destructor from the current class</td>
</tr>
<tr>
<td>deleteEventReception</td>
<td>Deletes the specified event reception from the current class</td>
</tr>
<tr>
<td>deleteReception</td>
<td>Deletes the specified reception from the current class</td>
</tr>
<tr>
<td>deleteSuperclass</td>
<td>Deletes a superclass from the current class</td>
</tr>
<tr>
<td>deleteType</td>
<td>Deletes a type from the current class</td>
</tr>
</tbody>
</table>
addClass

Write method

Description

The `addClass` method adds a class to the current class.

Visual Basic

Syntax

```vba
addClass (name As String) As RPClass
```

Arguments

- `name`:
  - The name of the new class

Return Value

The new class

C/C++ Prototype

```c
HRESULT addClass (BSTR name, IRPClass** pVal)
```

Return Value

- `HRESULT (0 for success, or a signed integer error code)`
addConstructor

Write method

Description

The addConstructor method adds a constructor to the current class.

Visual Basic

Syntax

addConstructor (argumentsData As String) As RPOperation

Arguments

argumentsData

The arguments for the constructor

Return Value

The new constructor for this class

C/C++ Prototype

HRESULT addConstructor (BSTR argumentsData,
IRPOperation** pVal)

Return Value

HRESULT (0 for success, or a signed integer error code)

Example

Sub addNetwork(c As RPClass)
Dim o As RPOperation
  c.addOperation ("serialize")
  c.addOperation ("unserialize")
  c.addConstructor ("")
  On Error Resume Next
  c.addDestructor ("")
  x = c.addStereotype("G3Network", "Class")
End Sub
addDestructor

Write method

Description

The **addDestructor** method adds a destructor to the current class.

Visual Basic

Syntax

```visualbasic
addDestructor() As RPOperation
```

Return Value

The new destructor for this class

C/C++ Prototype

```c
HRESULT addDestructor (IRPOperation** pVal)
```

Return Value

`HRESULT (0 for success, or a signed integer error code)`

Example

```plaintext
Sub addNetwork(c As RPClass)
  Dim o As RPOperation
  c.addOperation ("serialize")
  c.addOperation ("unserialize")
  c.addConstructor ("")
  On Error Resume Next
  c.addDestructor ("")
  x = c.addStereotype("G3Network", "Class")
End Sub
```
**addEventReception**

Write method

**Description**

The `addEventReception` method adds an event reception to the current class.

**Visual Basic**

**Syntax**

```vbnet
addEventReception (name As String) As RPEventReception
```

**Arguments**

- `name`  
  The name of the new event reception for this class

**Return Value**

The new event reception

**C/C++ Prototype**

```c
HRESULT addEventReception (BSTR name,
IRPEventReception** pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
addLink

The addLink method adds a link between two objects to the current class.

Syntax

```
addLink(fromPart As RInstance, toPart As RInstance, assoc As RRelation, fromPort As RPort, toPort As RPort) As RLink
```

Arguments

fromPart, toPart

The objects that are being linked.

assoc

Association that is being instantiated (optional).

fromPort, toPort

Ports that are being linked (optional).
addReception

Write method

Description

The `addReception` method adds a reception to the current class.

Visual Basic

Syntax

    addReception (name As String) As RPEventReception

Arguments

    name
    The name of the new reception for this class

Return Value

    The new reception

C/C++ Prototype

    HRESULT addReception (BSTR name,
                          IRPEventReception** pVal)

Return Value

    HRESULT (0 for success, or a signed integer error code)
addSuperclass

Write method

Description

The `addSuperclass` method inherits this class from a new superclass.

Visual Basic

Syntax

```vbnet
addSuperclass (superClass As RPClass)
```

Arguments

- `superClass`
  Specifies the `RPClass` from which this class will inherit

C/C++ Prototype

```c
HRESULT addSuperclass (IRPClass* superClass)
```

Return Value

- `HRESULT` (0 for success, or a signed integer error code)
addTriggeredOperation

Write method

Description

The addTriggeredOperation method adds a new triggered operation to the current class.

Visual Basic

Syntax

addTriggeredOperation (name As String) As RPOperation

Arguments

name

A string that specifies the name of the new trigger

Return Value

The new trigger for this class

C/C++ Prototype

HRESULT addTriggeredOperation (BSTR name,
IRPOperation** pVal)

Return Value

HRESULT (0 for success, or a signed integer error code)
addType
Write method

Description

The `addType` method adds a type to the current class.

Visual Basic

Syntax

```VisualBasic
addType (name As String) As RPType
```

Arguments

- `name`
  The name of the new type

Return Value

The new type for this class

C/C++ Prototype

```c
HRESULT addType (BSTR name, IRPTYPE** pVal)
```

Return Value

RESULT (0 for success, or a signed integer error code)
**deleteClass**

*Write method*

**Description**

The `deleteClass` method deletes a class from the current class.

**Visual Basic**

**Syntax**

```
deleteClass (name As String)
```

**Arguments**

- `name`
  
The name of the class to delete

**C/C++ Prototype**

```
HRESULT deleteClass (BSTR name)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
deleteConstructor

Write method

Description

The `deleteConstructor` method deletes a constructor from the current class.

Visual Basic

Syntax

```
    deleteConstructor (constructor As RPOperation)
```

Arguments

- `constructor`  
The constructor to delete

C/C++ Prototype

```
    HRESULT deleteConstructor (IRPOperation* constructor)
```

Return Value

- `HRESULT` (0 for success, or a signed integer error code)
**deleteDestructor**

**Write method**

**Description**

The **deleteDestructor** method deletes a destructor from the current class.

**Visual Basic**

**Syntax**

```vbscript
deleteDestructor()
```

**C/C++ Prototype**

```c
HRESULT deleteDestructor()
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
deleteEventReception

Write method

Description
The deleteEventReception method deletes the specified event reception.

Visual Basic

Syntax

deleteEventReception (pVal As RPEventReception)

Arguments

pVal
The event reception to delete

C/C++ Prototype

HRESULT deleteEventReception (IRPEventReception* pVal)

Return Value

HRESULT (0 for success, or a signed integer error code)
**deleteReception**

**Write method**

**Description**

The `deleteReception` method deletes the specified reception.

**Visual Basic**

**Syntax**

```
deleteReception (pVal As RPEventReception)
```

**Arguments**

- `pVal`
  
  The event reception to delete

**C/C++ Prototype**

```
HRESULT deleteReception (IRPEventReception* pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
deleteSuperclass

Write method

Description

The `deleteSuperclass` method deletes the superclass for the current class.

Visual Basic

Syntax

```vbnet
deleteSuperclass (superClass As RPClass)
```

Arguments

`superClass`
The superclass (base class) to delete

C/C++ Prototype

```c
HRESULT deleteSuperclass (IRPClass* superClass)
```

Return Value

`HRESULT` (0 for success, or a signed integer error code)
**deleteType**

*Write method*

**Description**

The *deleteType* method deletes a type from the current class.

**Visual Basic**

**Syntax**

```
deleteType (name As String)
```

**Arguments**

- `name`
  
  The type to delete

**C/C++ Prototype**

```
HRESULT deleteType (BSTR name)
```

**Return Value**

- HRESULT (0 for success, or a signed integer error code)
**IRPClassifier Interface**

The `IRPClassifier` interface is an abstract interface consisting of all the shared features of classes, actors, use cases, and (data) types. It inherits from `IRPUnit`.

**VB Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>activityDiagram</td>
<td><code>RPFlowchart</code></td>
<td>RO</td>
<td>The activity diagram</td>
</tr>
<tr>
<td>attributes</td>
<td>Collection of <code>RPAttributes</code></td>
<td>RO</td>
<td>A collection of attributes belonging to this classifier</td>
</tr>
<tr>
<td>baseClassifiers</td>
<td>Collection of <code>RPClassifiers</code></td>
<td>RO</td>
<td>A collection of classifiers from which this classifier is derived (inherits)</td>
</tr>
<tr>
<td>derivedClassifiers</td>
<td>Collection of <code>RPClassifiers</code></td>
<td>RO</td>
<td>A collection of classifiers that derive (inherit) from this classifier</td>
</tr>
<tr>
<td>flows</td>
<td>Collection of <code>RPInformationFlows</code></td>
<td>RO</td>
<td>A collection of flows belonging to this classifier</td>
</tr>
<tr>
<td>flowItems</td>
<td>Collection of <code>RPInformationItems</code></td>
<td></td>
<td>A collection of flowItems belonging to this classifier</td>
</tr>
<tr>
<td>generalizations</td>
<td>Collection of <code>RPGeneralizations</code></td>
<td>RO</td>
<td>A collection of generalizations that generalize this classifier (for which this classifier is a specialization)</td>
</tr>
<tr>
<td>interfaceItems</td>
<td>Collection of <code>RPInterfaceItems</code></td>
<td>RO</td>
<td>A collection of operations, events, and event receptions belonging to this classifier</td>
</tr>
<tr>
<td>nestedClassifiers</td>
<td>Collection of <code>RPClassifiers</code></td>
<td>RO</td>
<td>A collection of classifiers defined in this classifier</td>
</tr>
<tr>
<td>operations</td>
<td>Collection of <code>RPOperations</code></td>
<td>RO</td>
<td>A collection of operations belonging to this classifier</td>
</tr>
<tr>
<td>ports</td>
<td><code>RPCollection</code></td>
<td>RO</td>
<td>A collection of ports belonging to this classifier</td>
</tr>
<tr>
<td>relations</td>
<td>Collection of <code>RPRelations</code></td>
<td>RO</td>
<td>A collection of all relations belonging to this classifier</td>
</tr>
<tr>
<td>statechart</td>
<td><code>RPStatechart</code>*</td>
<td>RO</td>
<td>The handle to the statechart of this class, if it has one</td>
</tr>
</tbody>
</table>
# Method Summary

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>addActivityDiagram</code></td>
<td>Adds an activity diagram to the current class</td>
</tr>
<tr>
<td><code>addAttribute</code></td>
<td>Adds an attribute to the current class</td>
</tr>
<tr>
<td><code>addFlowItems</code></td>
<td>Adds the specified flowItem to the collection of flowItems</td>
</tr>
<tr>
<td><code>addFlows</code></td>
<td>Adds the specified flow to the collection of flows</td>
</tr>
<tr>
<td><code>addGeneralization</code></td>
<td>Adds a generalization to the current class</td>
</tr>
<tr>
<td><code>addOperation</code></td>
<td>Adds an operation to the current class</td>
</tr>
<tr>
<td><code>addRelation</code></td>
<td>Adds a symmetric relation between the current class and another one</td>
</tr>
<tr>
<td><code>addStatechart</code></td>
<td>Adds a statechart to the current class</td>
</tr>
<tr>
<td><code>addUnidirectionalRelation</code></td>
<td>Adds a directional relation from the current class to another class</td>
</tr>
<tr>
<td><code>deleteActivityDiagram</code></td>
<td>Deletes the specified activity diagram from the current class</td>
</tr>
<tr>
<td><code>deleteAttribute</code></td>
<td>Deletes the specified attribute from the current class</td>
</tr>
<tr>
<td><code>deleteFlowItems</code></td>
<td>Deletes the specified flowItem from the collection of flowItems</td>
</tr>
<tr>
<td><code>deleteFlows</code></td>
<td>Deletes the specified flow from the collection of flows</td>
</tr>
<tr>
<td><code>deleteGeneralization</code></td>
<td>Deletes the specified generalization from the current class</td>
</tr>
<tr>
<td><code>deleteOperation</code></td>
<td>Deletes the specified operation from the current class</td>
</tr>
<tr>
<td><code>deleteRelation</code></td>
<td>Deletes the specified relation from the current class</td>
</tr>
<tr>
<td><code>deleteStatechart</code></td>
<td>Deletes the specified statechart from the current class</td>
</tr>
<tr>
<td><code>findAttribute</code></td>
<td>Retrieves the specified attribute of the classifier</td>
</tr>
<tr>
<td><code>findBaseClassifier</code></td>
<td>Retrieves a base (parent) classifier of a classifier</td>
</tr>
<tr>
<td><code>findDerivedClassifier</code></td>
<td>Retrieves the specified derived classifier of a classifier</td>
</tr>
<tr>
<td><code>findGeneralization</code></td>
<td>Retrieves the specified generalization of a classifier</td>
</tr>
<tr>
<td><code>findInterfaceItem</code></td>
<td>Retrieves an operation or event reception of the given signature that belongs to a classifier</td>
</tr>
</tbody>
</table>
### Rhapsody Interfaces

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>findNestedClassifier</strong></td>
<td>Retrieves the specified classifier defined within this object</td>
</tr>
<tr>
<td><strong>findNestedClassifierRecursive</strong></td>
<td>Retrieves the specified classifier defined in this object and in objects defined within this object</td>
</tr>
<tr>
<td><strong>findRelation</strong></td>
<td>Retrieves the specified relation that belongs to the current classifier</td>
</tr>
<tr>
<td><strong>findTrigger</strong></td>
<td>Retrieves the specified trigger in the statechart of the current class</td>
</tr>
<tr>
<td><strong>getAttributesIncludingBases</strong></td>
<td>Retrieves the attributes defined for this class and the ones inherited from its superclasses</td>
</tr>
<tr>
<td><strong>getInterfaceItemsIncludingBases</strong></td>
<td>Retrieves the operations and event receptions defined for this class and the ones it inherited from its superclasses</td>
</tr>
<tr>
<td><strong>getRelationsIncludingBases</strong></td>
<td>Retrieves the relations defined for this class and the ones it inherited from its superclasses</td>
</tr>
</tbody>
</table>

**Note**

Some of the properties and methods are meaningful only for some of the derived interfaces. When meaningless, the call will return nothing (NULL) or an empty collection.
addActivityDiagram
Write method

Description
The `addActivityDiagram` method adds an activity diagram to the current class.

Visual Basic
Syntax
addActivityDiagram () As RFlowchart

Return Value
The new activity diagram

C/C++ Prototype

HRESULT addActivityDiagram (IRPFlowchart** pVal)

Return Value
HRESULT (0 for success, or a signed integer error code)
**addAttribute**

**Write method**

**Description**

The *addAttribute* method adds an attribute to the current class.

**Visual Basic**

**Syntax**

```
addAttribute (name As String) As RPAttribute
```

**Arguments**

- *name*  
The name of the new attribute

**C/C++ Prototype**

```
HRESULT addAttribute (BSTR name, IRPAttribute** pVal)
```

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
addFlowItems

Write method

Description

The addFlowItems method adds the specified flowItem to the collection of flowItems.

Visual Basic

Syntax

addFlowItems (name As String) As RPFlowItem

Arguments

name
The name of the new flowItem

C/C++ Prototype

HRESULT addFlowItems (BSTR name, IRPFlowItem** ppItem)

Return Value

HRESULT (0 for success, or a signed integer error code)
addFlows

Write method

Description

The `addFlows` method adds the specified flow to the collection of flows.

Visual Basic

Syntax

    addFlows (name As String) As RPFlow

Arguments

    name

    The name of the new flow

C/C++ Prototype

    HRESULT addFlows (BSTR name, IRPFlow** ppFlow)

Return Value

    HRESULT (0 for success, or a signed integer error code)
**addGeneralization**

**Write method**

**Description**

The `addGeneralization` method adds a generalization to the current class.

**Visual Basic**

**Syntax**

`addGeneralization (pVal As RPClassifier)`

**Arguments**

- `pVal`  
  The generalization to add to this class

**C/C++ Prototype**

`HRESULT addGeneralization (IRPClassifier *pVal)`

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)

**Example**

```visualbasic
Sub addUi(c As RPClass)
Dim x As Object
Dim p As RPPackage
Dim theClass As RPClass
' all gui objects are derived from GUI.UIBase
c.Description = "gui class"
On Error Resume Next
Set p = pr.findNestedElement("GUI", "Package")
Set theClass = p.findNestedElement("UIBase", "Class")
c.addGeneralization theClass
If Not Err.Number = 0 Then
    MsgBox (errorMessage)
End If

c.addStereotype "G3UI", "Class"
End Sub
```
addOperation

Write method

Description

The addOperation method adds an operation to the current class.

Visual Basic

Syntax

addOperation (name As String) As RPOperation

Arguments

name

The name of the new operation

Return Value

The operation added to this class

C/C++ Prototype

HRESULT addOperation (BSTR name, IRPOperation** pVal)

Return Value

HRESULT (0 for success, or a signed integer error code)

Example

Sub addNetwork(c As RPClass)
Dim o As RPOperation
    c.addOperation ("serialize")
    c.addOperation ("unserialize")
    c.addConstructor ("")
On Error Resume Next
    c.addDestructor ("")
    x = c.addStereotype("G3Network", "Class")
End Sub
**addRelation**

**Write method**

**Description**

The `addRelation` method adds a symmetric relation between the current class and another one.

**Visual Basic**

**Syntax**

```vbnet
addRelation (otherClassName As String,
             otherClassPackageName As String,
             roleName1 As String, linkType1 As String,
             multiplicity1 As String, roleName2 As String,
             linkType2 As String, multiplicity2 As String,
             linkName As String) As RPRelation
```

**Arguments**

- **OtherClassName**
  The name of the other class involved in the new relation with the current class.

- **OtherClassPackageName**
  The name of the package containing the other class.

- **roleName1**
  The role name of the other class, from the point of view of the current class.

- **roleName2**
  The role name of the current class, from the point of view of the other class.

- **linkType1**
  The relation type. The possible values are as follows:
  - Aggregation
  - Association
  - Composition

- **linkType2**
  The second relation type. The possible values are as follows:
  - Aggregation
  - Association
  - Composition

- **multiplicity1**
  The multiplicity of instances for the other class.

- **multiplicity2**
  The multiplicity of instances for the other class.
The multiplicity of instances for the current class.

linkName

The name of the link. This is a descriptive and explanatory field that plays no part in code generation.

Notes

The valid combinations of linkType1 and linkType2 are as follows:
Association/Association—I know you; you know me.
Aggregation/Association—I belong to you; you know me.
Composition/Association—I strongly belong to you; you know me.
Association/Aggregation—I know you; you belong to me.
Association/Composition—I know you; you strongly belong me.

Return Value

The new relation

C/C++ Prototype

HRESULT addRelation (BSTR otherClassName,
BSTR otherClassPackageName, BSTR roleName1,
BSTR linkType1, BSTR multiplicity1, BSTR roleName2,
BSTR linkType2, BSTR multiplicity2, BSTR linkName,
IRPRelation** pVal)

Return Value

HRESULT (0 for success, or a signed integer error code)
addStatechart

Write method

Description

The addStatechart method adds a statechart to the current class.

Visual Basic

Syntax

    addStatechart() As RPStatechart

Return Value

The new statechart

C/C++ Prototype

    HRESULT addStatechart (IRPStatechart** pVal)

Return Value

    HRESULT (0 for success, or a signed integer error code)
**addUnidirectionalRelation**

**Write method**

**Description**

The *addUnidirectionalRelation* method adds a directional relation from the current class to another class.

**Visual Basic**

**Syntax**

```vba
addUnidirectionalRelation (otherClassName As String,
 otherClassPackageName As String,
 roleName As String, linkType As String,
 multiplicity As String,
 linkName As String) As RPRelation
```

**Arguments**

- **OtherClassName**
  The name of the other class involved in the new relation with the current class.

- **OtherClassPackageName**
  The name of the package containing the other class.

- **roleName**
  The role name of the other class, from the point of view of the current class.

- **linkType**
  The relation type. The possible values are as follows:
  - Aggregation
  - Association
  - Composition

- **multiplicity**
  The multiplicity of instances for the other class.

- **linkName**
  The name of the link. This is a descriptive and explanatory field that plays no part in code generation.

**Return Value**

The new relation
C/C++ Prototype

HRESULT addUnidirectionalRelation (BSTR otherClassName, 
BSTR otherClassPackageName, BSTR roleName, 
BSTR linkType, BSTR multiplicity, BSTR linkName, 
IRPRelation** pVal)

Return Value

HRESULT (0 for success, or a signed integer error code)

deleteActivityDiagram

Write method

Description

The deleteActivityDiagram method deletes the specified activity diagram from the current class.

Visual Basic

Syntax

deleteActivityDiagram ()

C/C++ Prototype

HRESULT deleteActivityDiagram()
**deleteAttribute**

**Write method**

**Description**

The **deleteAttribute** method deletes the specified attribute from the current class.

**Visual Basic**

**Syntax**

```visual-basic
deleteAttribute (attribute As RPAttribute)
```

**Arguments**

- `attribute`
  The attribute to delete

**C/C++ Prototype**

```c
HRESULT deleteAttribute (IRPAttribute* attribute)
```

**Return Value**

- **HRESULT** (0 for success, or a signed integer error code)
**deleteFlowItems**

**Write method**

**Description**

The `deleteFlowItems` method deletes the specified `flowItem` from the collection of `flowItems`.

**Visual Basic**

**Syntax**

```vbscript
deleteFlowItems (pItem As RPFlowItem)
```

**Arguments**

- `pFlowItem`
  - The `flowItem` to delete

**C/C++ Prototype**

```c
HRESULT deleteFlowItems (IRPFlowItem* pItem)
```

**Return Value**

- `HRESULT (0 for success, or a signed integer error code)`
**deleteFlows**

*Write method*

**Description**

The `deleteFlows` method deletes the specified flow from the collection of flows.

**Visual Basic**

**Syntax**

```vbnet
deleteFlows (pFlow As RPFlow)
```

**Arguments**

- `pFlow`  
The flow to delete

**C/C++ Prototype**

```c
HRESULT deleteFlows (IRPFlow* pFlow)
```

**Return Value**

- `HRESULT` (0 for success, or a signed integer error code)
**deleteGeneralization**

**Write method**

**Description**

The `deleteGeneralization` method deletes the specified generalization from the current class.

**Visual Basic**

**Syntax**

`deleteGeneralization (superClass As RPClassifier)`

**Arguments**

- `superClass`:
  The superclass of the current class to be deleted.

**C/C++ Prototype**

`HRESULT deleteGeneralization (IRPClassifier* superClass)`

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
**deleteOperation**

**Write method**

**Description**

The `deleteOperation` method deletes the specified operation from the current class.

**Visual Basic**

**Syntax**

`deleteOperation (operation As RPOperation)`

**Arguments**

operation

The operation to delete

**C/C++ Prototype**

`HRESULT deleteOperation (IRPOperation* operation)`

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
deleteRelation

Write method

Description

The deleteRelation method deletes the specified relation from the current class.

Visual Basic

Syntax

deleteRelation (relation As RPRelation)

Arguments

relation

The relation to delete

C/C++ Prototype

HRESULT deleteRelation (IRPRelation* relation)

Return Value

HRESULT (0 for success, or a signed integer error code)
**deleteStatechart**

**Write method**

**Description**

The `deleteStatechart` method deletes the specified statechart from this class.

**Visual Basic**

**Syntax**

```vbnet
deleteStatechart()
```

**C/C++ Prototype**

```c
HRESULT deleteStatechart()
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
**findAttribute**

**Read method**

**Description**

The `findAttribute` method retrieves the specified attribute of the classifier.

**Visual Basic**

**Syntax**

```vbscript
findAttribute (name As String) As RPAttribute
```

**Arguments**

- `name`  
  The name of the attribute to find

**Return Value**

The named attribute of the classifier

**C/C++ Prototype**

```c
HRESULT findAttribute (BSTR newVal, IRPAttribute** pVal)
```

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
**Rhapsody Interfaces**

**findBaseClassifier**

**Read method**

**Description**

The `findBaseClassifier` method retrieves a base (parent) classifier of a classifier.

**Visual Basic**

**Syntax**

`findBaseClassifier (newVal As String) As RPClassifier`

**Arguments**

- `newVal`
  The name of the base classifier

**Return Value**

The base classifier of this classifier

**C/C++ Prototype**

```c
HRESULT findBaseClassifier (BSTR newVal,
IRPClassifier** pVal)
```

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
findDerivedClassifier

Read method

Description

The `findDerivedClassifier` method retrieves the specified derived classifier of a classifier.

Visual Basic

Syntax

```vbnet
findDerivedClassifier (newVal As String) As RPClassifier
```

Arguments

- `newVal`:
The name of the derived classifier of this classifier

Return Value

The derived classifier of this classifier

C/C++ Prototype

```c
HRESULT findDerivedClassifier (BSTR newVal, IRPClassifier** pVal)
```

Return Value

- `HRESULT (0` for success, or a signed integer error code)
findGeneralization

Read method

Description

The findGeneralization method retrieves the specified generalization that belongs to this classifier.

Visual Basic

Syntax

findGeneralization (newVal As String) As RPGeneralization

Arguments

newVal
The name of the generalization

Return Value

The RPGeneralization

C/C++ Prototype

HRESULT findGeneralization (BSTR newVal,
IRPGeneralization** pVal)

Return Value

HRESULT (0 for success, or a signed integer error code)
**findInterfaceItem**

**Read method**

**Description**

The `findInterfaceItem` method retrieves an operation or event reception of the given signature that belongs to a classifier.

**Visual Basic**

**Syntax**

```vbnet
findInterfaceItem (signature As String) As RPInterfaceItem
```

**Arguments**

- `signature`: The signature of the operation or event reception of this classifier

**Return Value**

The operation or event reception

**C/C++ Prototype**

```c
HRESULT findInterfaceItem (BSTR signature, 
IRPInterfaceItem** pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
**findNestedClassifier**

**Read method**

**Description**

The `findNestedClassifier` method retrieves the specified classifier defined within this object.

**Visual Basic**

**Syntax**

```vbnet
findNestedClassifier (newVal As String) As RPClassifier
```

**Arguments**

- `newVal`
  The name of the nested classifier

**Return Value**

- The nested classifier within this classifier

**C/C++ Prototype**

```c
HRESULT findNestedClassifier (BSTR newVal, IRPClassifier** pVal)
```

**Return Value**

- `HRESULT` (0 for success, or a signed integer error code)
**findNestedClassifierRecursive**

**Read method**

**Description**

The `findNestedClassifierRecursive` method recursively retrieves the specified classifier defined in this object and in objects defined within this object.

**Visual Basic**

**Syntax**

```vbnet
findNestedClassifierRecursive (newVal As String) As RPModelElement
```

**Arguments**

- `newVal`:
  - The name of the nested classifier (at any level of ownership)

**Return Value**

The nested classifier

**C/C++ Prototype**

```c
HRESULT findNestedClassifierRecursive (BSTR newVal, IRPModelElement** pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
findNestedGeneralization

Read method

Description

The findNestedGeneralization method retrieves the specified generalization relation.

Visual Basic

Syntax

    findGeneralization (name As String) As IRPGeneralization

Arguments

    name

    A string that specifies the name of the generalization to find

Return Value

    The generalization for this classifier (an IRPGeneralization)

C++ Prototype

    HRESULT findGeneralization(BSTR newVal,
        IRPGeneralization** pVal)

Return Value

    HRESULT (0 for success, or a signed integer error code)
**findRelation**

**Read method**

**Description**

The `findRelation` method retrieves the specified relation that belongs to the current classifier.

**Visual Basic**

**Syntax**

```vbnet
findRelation (newVal As String) As RPRelation
```

**Arguments**

- `newVal`  
  The name of the relation to find

**Return Value**

The classifier’s relation

**C/C++ Prototype**

```c
HRESULT findRelation (BSTR newVal, IRPRelation** pVal)
```

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
findTrigger

Read method

Description

The `findTrigger` method retrieves the specified trigger in the statechart of the current class.

Visual Basic

Syntax

```visualbasic
findTrigger (name As String) As RPInterfaceItem
```

Arguments

```text
name
The name of the trigger to find
```

Return Value

The trigger

C/C++ Prototype

```c
HRESULT findTrigger (BSTR name, IRPInterfaceItem** pVal)
```

Return Value

```text
HRESULT (0 for success, or a signed integer error code)
```
**getAttributesIncludingBases**

*Read method*

**Description**

The `getAttributesIncludingBases` method retrieves the attributes defined for this class and the ones inherited from its superclasses.

**Visual Basic**

**Syntax**

```visual-basic
getAttributesIncludingBases() As RPCollection
```

**Return Value**

A collection of class attributes (`RPAttributes`)

**C/C++ Prototype**

```c
HRESULT getAttributesIncludingBases (
    IRPCollection** pVal)
```

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
getInterfaceItemsIncludingBases

Read method

Description

The getInterfaceItemsIncludingBases method retrieves the operations and event receptions defined for this class and the ones it inherited from its superclasses.

Visual Basic

Syntax

getInterfaceItemsIncludingBases() As RPCollection

Return Value

A collection of interface items

C/C++ Prototype

HRESULT getInterfaceItemsIncludingBases(
    IRPCollection** pVal)

Return Value

HRESULT (0 for success, or a signed integer error code)
getRelationsIncludingBases

Read method

Description

The `getRelationsIncludingBases` method retrieves the relations defined for this class and the ones it inherited from its superclasses.

Visual Basic

Syntax

```vbnet
getRelationsIncludingBases() As RPRelations
```

Return Value

A collection of relations

C/C++ Prototype

```c
HRESULT getRelationsIncludingBases (IRPCollection** pVal)
```

Return Value

`HRESULT` (0 for success, or a signed integer error code)
IRPClassifierRole Interface

The **IRPClassifierRole** interface represents an object participating in the collaboration. It usually corresponds to some object of a given class or actor. It inherits from **IRPModelElement**.

**VB Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>formalClassifier</td>
<td>RPClassifier</td>
<td>RO</td>
<td>The classifier (NULL (unspecified), systemBorder, or multipleObjects)</td>
</tr>
<tr>
<td>referencedSequenceDiagram</td>
<td>RPSequence Diagram</td>
<td>RW</td>
<td>The referenced sequence diagram</td>
</tr>
<tr>
<td>roleType</td>
<td>String</td>
<td>RO</td>
<td>The role type (unspecified, systemBorder, class, actor, or multipleObjects)</td>
</tr>
</tbody>
</table>
IRPCollaboration Interface

The IRPCollaboration interface represents the logical collaboration, devoid of any sequence diagram or collaboration diagram graphics. Note that the two diagrams give rise to similar but slightly different IRPCollaboration objects. This class inherits from IRFModelElement.

**VB Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>activationCondition</td>
<td>CString</td>
<td>RO</td>
<td>The activation condition. This can be empty.</td>
</tr>
<tr>
<td>activationMode</td>
<td>CString</td>
<td>RO</td>
<td>The activation mode (initial, invariant, or unspecified).</td>
</tr>
<tr>
<td>associations</td>
<td>RPAssociationRoles</td>
<td>RO</td>
<td>A collection of RPAssociationRoles in the collaboration diagram. This applies only to collaboration diagram-based IRPCollaborations.</td>
</tr>
<tr>
<td>classifier</td>
<td>RPClassifierRoles</td>
<td>RO</td>
<td>A collection of RPClassifierRoles in the collaboration diagram.</td>
</tr>
<tr>
<td>messagePoints</td>
<td>RPMessagePoints</td>
<td>RO</td>
<td>A collection of RPMessagePoints. For sequences, this is the way of obtaining full information about the order of messages in the sequence diagram. For collaborations, each send messagepoint is immediately followed by a receive messagepoint on the same message.</td>
</tr>
<tr>
<td>messages</td>
<td>RPMessages</td>
<td>RO</td>
<td>A collection of RPMessages. For collaborations, this list contains all information regarding the order of elements in the model. For sequences, some information is lost and the message list is ordered by the send time (as opposed to the receive time).</td>
</tr>
<tr>
<td>mode</td>
<td>Cstring</td>
<td>RO</td>
<td>The mode (existential, universal, or unspecified).</td>
</tr>
</tbody>
</table>
### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>addCancelledTimeout</strong></td>
<td>Adds a cancelled timeout to the diagram</td>
</tr>
<tr>
<td><strong>addClassifierRole</strong></td>
<td>Adds a classifier role</td>
</tr>
<tr>
<td><strong>addClassifierRoleByName</strong></td>
<td>Adds a classifier role, given its name</td>
</tr>
<tr>
<td><strong>addCtor</strong></td>
<td>Adds a constructor</td>
</tr>
<tr>
<td><strong>addDestructionEvent</strong></td>
<td>Adds a destruction event to a classifier role in a sequence diagram</td>
</tr>
<tr>
<td><strong>addDtor</strong></td>
<td>Adds a destructor</td>
</tr>
<tr>
<td><strong>addFoundMessage</strong></td>
<td>Adds a found message to a classifier role in a sequence diagram</td>
</tr>
<tr>
<td><strong>addInteractionOccurrence</strong></td>
<td>Adds an interaction occurrence (reference diagram) to the diagram</td>
</tr>
<tr>
<td><strong>addInteractionOperator</strong></td>
<td>Adds an interaction operator to a sequence diagram</td>
</tr>
<tr>
<td><strong>addLostMessage</strong></td>
<td>Adds a lost message to a classifier role in a sequence diagram</td>
</tr>
<tr>
<td><strong>addMessage</strong></td>
<td>Adds a message</td>
</tr>
<tr>
<td><strong>addSystemBorder</strong></td>
<td>Adds a system border</td>
</tr>
<tr>
<td><strong>addTimeInterval</strong></td>
<td>Adds a time interval to the diagram</td>
</tr>
<tr>
<td><strong>addTimeout</strong></td>
<td>Adds a timeout the diagram</td>
</tr>
<tr>
<td><strong>generateSequence</strong></td>
<td>Generates the specified sequence diagram</td>
</tr>
<tr>
<td><strong>getConcurrentGroup</strong></td>
<td>Retrieves the activation messages</td>
</tr>
<tr>
<td><strong>getConcurrentGroup</strong></td>
<td>Retrieves all the messages concurrent with the input message, including the input message itself</td>
</tr>
<tr>
<td><strong>getMessagePoints</strong></td>
<td>Returns an ordered collection of all messagepoints occurring on this classifier</td>
</tr>
<tr>
<td><strong>getPredecessor</strong></td>
<td>Retrieves the message that precedes the specified message</td>
</tr>
<tr>
<td><strong>getSuccessor</strong></td>
<td>Retrieves the message that follows the specified message</td>
</tr>
</tbody>
</table>
addCancelledTimeout

Write method

Description

The `addCancelledTimeout` method adds a cancelled timeout to a collaboration diagram.

Visual Basic

Syntax

```visualbasic
addCancelledTimeout (receiver As RPClassifierRole) As RPMessage
```

Arguments

- `receiver`
  The receiver object for the timeout

Return Value

The new cancelled timeout

C/C++ Prototype

```c
HRESULT addCancelledTimeout (IRPClassifierRole *receiver,
                               IRPMessage **pVal)
```

Return Value

- `HRESULT` (0 for success, or a signed integer error code)
**addClassifierRole**

**Write method**

**Description**

The `addClassifierRole` method adds a classifier role.

**Visual Basic**

**Syntax**

```
addClassifierRole (newVal As String, cls As RPClass) As RPClassifierRole
```

**Arguments**

- `newVal`
  The name of the new classifier role
- `cls`
  The name of the class

**Return Value**

The new `RPClassifierRole`

**C/C++ Prototype**

```
HRESULT addClassifierRole (BSTR newVal, IRPClass *cls,
                          IRPClassifierRole** pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
**addClassifierRoleByName**

**Write method**

**Description**

The `addClassifierRoleByName` method adds the specified classifier role.

**Visual Basic**

**Syntax**

```vba
addClassifierRoleByName (newVal As String, 
classFullPath As String) As RPClassifierRole
```

**Arguments**

- `newVal`
  The name of the classifier role to add
- `classFullPath`
  The full path to the class

**Return Value**

The new `RPClassifierRole`

**C/C++ Prototype**

```c
HRESULT addClassifierRoleByName (BSTR newVal, 
BSTR classFullPath, I RPClassifierRole** pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
**addCtor**

**Write method**

**Description**

The `addCtor` method adds a constructor.

**Visual Basic**

**Syntax**

```visualbasic
addCtor (interItem As RPInterfaceItem, actualParamList As String, sender As RPClassifierRole, receiver As RPClassifierRole) As RPMessage
```

**Arguments**

- `interItem`:
  The interface item
- `actualParamList`:
  The list of parameters for the constructor
- `sender`:
  The RPClassifierRole that acts as the sender
- `receiver`:
  The RPClassifierRole that acts as the receiver

**Return Value**

An RPMessage

**C/C++ Prototype**

```c
HRESULT addCtor (IRPInterfaceItem *interItem, BSTR actualParamList, IRPClassifierRole *sender, IRPClassifierRole *receiver, IRPMessage** pVal)
```

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
addDestructionEvent

Write method

Description

The `addDestructionEvent` method is used to add a destruction event to a classifier role in a sequence diagram.

Visual Basic

Syntax

```vbnet
addDestructionEvent (classifier As RPClassifierRole) As RPMessage
```

Arguments

classifier

The classifier role to which the destruction event should be added.

Return Value

An RPMessage

Return Value

HRESULT (0 for success, or a signed integer error code)
addDtor

Write method

Description

The \texttt{addDtor} method adds a destructor.

Visual Basic

Syntax

\begin{verbatim}
addDtor (interItem As RPInterfaceItem, 
actualParamList As String, sender As RPClassifierRole, 
receiver As RPClassifierRole) As RPMessage
\end{verbatim}

Arguments

\begin{itemize}
\item \texttt{interItem}\hspace{1em} The interface item
\item \texttt{actualParamList}\hspace{1em} The list of parameters for the constructor
\item \texttt{sender}\hspace{1em} The \texttt{RPClassifierRole} that acts as the sender
\item \texttt{receiver}\hspace{1em} The \texttt{RPClassifierRole} that acts as the receiver
\end{itemize}

Return Value

An RPMessage

C/C++ Prototype

\begin{verbatim}
HRESULT addDtor (IRPInterfaceItem *interItem, 
BSTR actualParamList, IRPClassifierRole *sender, 
IRPClassifierRole *receiver, IRPMessage** pVal)
\end{verbatim}

Return Value

\begin{verbatim}
HRESULT (0 for success, or a signed integer error code)
\end{verbatim}
addFoundMessage

Write method

Description

The `addFoundMessage` method is used to add a found message to a classifier role in a sequence diagram.

Visual Basic

Syntax

```
addFoundMessage (receiver As RPClassifierRole) As RPMessage
```

Arguments

- `receiver`
  
  The classifier role that receives the message from an unknown sender.

Return Value

An `RPMessage`

Return Value

- HRESULT (0 for success, or a signed integer error code)
addInteractionOccurrence
Write method

Description
The addInteractionOccurrence method adds a new interaction occurrence (reference diagram) to the collaboration diagram.

Visual Basic
Syntax
addInteractionOccurrence () As RPInteractionOccurrence

Return Value
The new interaction occurrence

C/C++ Prototype
HRESULT addInteractionOccurrence (IRPInteractionOccurrence** pVal)

Return Value
HRESULT (0 for success, or a signed integer error code)
addInteractionOperator

Write method

Description

The `addInteractionOperator` method is used to add an interaction operator to a sequence diagram.

Visual Basic

Syntax

```visualbasic
addInteractionOperator() As RPInteractionOperator
```

Arguments

None

Return Value

An `RPInteractionOperator`

Return Value

`HRESULT` (0 for success, or a signed integer error code)
**addLostMessage**

**Write method**

**Description**

The `addLostMessage` method is used to add a lost message to a classifier role in a sequence diagram.

**Visual Basic**

**Syntax**

```vbnet
addLostMessage (sender As RPClassifierRole) As RPMessage
```

**Arguments**

sender

The classifier role that sent the message that did not reach its target.

**Return Value**

An RPMessage

**Return Value**

HRESULT (0 for success, or a signed integer error code)
addMessage

Write method

Description

The `addMessage` method adds a message.

Visual Basic

Syntax

```vbnet
addMessage (interItem As RPInterfaceItem,
 actualParamList As String, sender As RPClassifierRole,
 receiver As RPClassifierRole) As RPMessage
```

Arguments

- `interItem`: The interface item
- `actualParamList`: The list of parameters for the constructor
- `sender`: The `RPClassifierRole` that acts as the sender
- `receiver`: The `RPClassifierRole` that acts as the receiver

Return Value

The new message

C/C++ Prototype

```c
HRESULT addMessage (IRPInterfaceItem *interItem,
 BSTR actualParamList, IRPClassifierRole *sender,
 IRPClassifierRole *receiver, IRPMessage** pVal)
```

Return Value

`HRESULT (0 for success, or a signed integer error code)`
addSystemBorder

Write method

Description

The `addSystemBorder` method adds a system border to the collaboration diagram.

Visual Basic

Syntax

```vba
addSystemBorder () As RPClassifierRole
```

Return Value

The new system border

C/C++ Prototype

```cpp
HRESULT addSystemBorder (IRPClassifierRole** pVal)
```

Return Value

`HRESULT (0 for success, or a signed integer error code)`
addTimeInterval

Write method

Description

The `addTimeInterval` method adds a time interval to the diagram.

Visual Basic

Syntax

```vbscript
addTimeInterval (receiver As RPClassifierRole)
As RPMessage
```

Arguments

`interItem`

The interface item

Return Value

The new time interval

C/C++ Prototype

```c
HRESULT addTimeInterval (IRPClassifierRole *receiver,
IRPMessage** pVal)
```

Return Value

`HRESULT (0 for success, or a signed integer error code)`
addTimeout

Write method

Description

The addTimeout method adds a timeout.

Visual Basic

Syntax

addTimeout (interItem As RPInterfaceItem,
    actualParamList As String, sender As RPClassifierRole,
    receiver As RPClassifierRole) As RPMessage

Arguments

    interItem
    The interface item

    actualParamList
    The list of parameters for the constructor

    sender
    The RPClassifierRole that acts as the sender

    receiver
    The RPClassifierRole that acts as the receiver

Return Value

    The new timeout

C/C++ Prototype

HRESULT addTimeout (IRPInterfaceItem *interItem,
    BSTR actualParamList, IRPClassifierRole *sender,
    IRPClassifierRole *receiver, IRPMessage** pVal)

Return Value

    HRESULT (0 for success, or a signed integer error code)
generateSequence

Write method

Description

The `generateSequence` method generates the specified sequence diagram.

Visual Basic

Syntax

```vbnet
generateSequence (newVal As String, owner As RPPackage) As RPSequenceDiagram
```

Arguments

- newVal
  The name of the sequence diagram to generate
- owner
  The owner package

Return Value

- The new RPSequenceDiagram

C/C++ Prototype

```c
HRESULT generateSequence (BSTR newVal, IRPPackage* owner, IRPSequenceDiagram** pVal)
```

Return Value

- HRESULT (0 for success, or a signed integer error code)
**getActivator**

**Read method**

**Description**

The `getActivator` method retrieves the activation messages.

**Visual Basic**

**Syntax**

```vbnet
getActivator (msg As RPMessage) As RPMessage
```

**Arguments**

- **msg**
  The message to retrieve

**Return Value**

A collection of `RPMessages`

**C/C++ Prototype**

```c
HRESULT getActivator (IRPMessage* msg,
                     IRPMessage** pVal)
```

**Return Value**

- `HRESULT` (0 for success, or a signed integer error code)
getConcurrentGroup

Read method

Description

The `getConcurrentGroup` method retrieves all the messages concurrent with the input message, including the input message itself. If the message does not have any concurrent messages because it is sequential, the method returns only the message itself.

Visual Basic

Syntax

```vbnet
getConcurrentGroup (message As RPMessage) As RPMessages
```

Arguments

- `message` - The group of messages to retrieve

Return Value

A collection of `RPMessages`

C/C++ Prototype

```c
HRESULT getConcurrentGroup (IRPMessage* message,
IRPCollection** pVal)
```

Return Value

`HRESULT` (0 for success, or a signed integer error code)
**getMessagePoints**  
Read method

**Description**

The `getMessagePoints` method returns an ordered collection of all messagepoints occurring on this classifier.

**Visual Basic**

**Syntax**

```visualbasic
getMessagePoints (classifier As RPClassifierRole)  
As RPCollection
```

**Arguments**

- `classifier`  
The RPClassifier whose messagepoints you want to retrieve

**Return Value**

A collection of RPMessagePoints

**C/C++ Prototype**

```c
HRESULT getMessagePoints (IRPClassifier* classifier,  
IRPCollection** pVal)
```

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
**getPredecessor**  
*Read method*

**Description**

The `getPredecessor` method retrieves the message that precedes the specified message.

**Visual Basic**

**Syntax**

```vbnet
getPredecessor (message As RPMessage) As RPMessage
```

**Arguments**

- `message`  
  The message whose predecessor you want

**Return Value**

The message that precedes the specified message

**C/C++ Prototype**

```c
HRESULT getPredecessor (IRPMessage *message,
                         IRPMessage** pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
**getSuccessor**

**Read method**

**Description**

The `getSuccessor` method retrieves the message that follows the specified message.

**Visual Basic**

**Syntax**

```vbnet
getSuccessor (message As RPMessage) As RPMessage
```

**Arguments**

- `message`:

  The message whose successor you want.

**Return Value**

The message that follows the specified message.

**C/C++ Prototype**

```c
HRESULT getSuccessor (IRPMessage *message, 
                     IRPMessage** pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
IRPCollaborationDiagram Interface

The IRPCollaborationDiagram interface represents a collaboration diagram. It inherits from IRPDiagram.

Method Summary

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getLogicalCollaboration</td>
<td>Retrieves the logic behind the collaboration diagram</td>
</tr>
</tbody>
</table>

getLogicalCollaboration

Read method

Description

The getLogicalCollaboration method retrieves the logic behind the collaboration diagram.

Visual Basic

Syntax

getLogicalCollaboration() As RPCollaboration

Return Value

The collaboration diagram

C/C++ Prototype

HRESULT getLogicalCollaboration (IRPCollaboration** pVal)

Return Value

HRESULT (0 for success, or a signed integer error code)
IRPCollection Interface

The IRPCollection interface is a utility class used to return collections of IRPModelElements. Methods and attributes that need to return more than one element always return a pointer to an IRPCollection.

IRPCollection also supports VB iteration via the following construct:

For Each obj in col

VB Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>Long</td>
<td>RO</td>
<td>The number of elements currently in the collection</td>
</tr>
<tr>
<td>Item(long i)</td>
<td>RModelElement</td>
<td>RO</td>
<td>The i\textsuperscript{th} element in the collection</td>
</tr>
</tbody>
</table>

Method Summary

| addItem | Adds an item to the collection |
**addItem**

**Write method**

**Description**

The `addItem` method adds an item to the collection.

**Visual Basic**

**Syntax**

```vbnet
addItem (newVal As RPModelElement)
```

**Arguments**

- `newVal`  
The new item to add

**C/C++ Prototype**

```c
HRESULT addItem (IRPModelElement* newVal)
```

**Return Value**

- `HRESULT (0 for success, or a signed integer error code)`
Rhapsody Interfaces

IRPComment Interface

The IRPComment interface represents Rhapsody comments. It inherits from IRPAnnotation.

IRPComponent Interface

The IRPComponent interface represents a code generation component. It inherits from IRPUnit.

VB Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>additionalSources</td>
<td>String</td>
<td>RW</td>
<td>The additional source files to be compiled with the component.</td>
</tr>
<tr>
<td>buildType</td>
<td>String</td>
<td>RW</td>
<td>The build type (library or executable).</td>
</tr>
<tr>
<td>configurations</td>
<td>Collection of RPConfiguration</td>
<td>RW</td>
<td>The configurations of this component.</td>
</tr>
<tr>
<td>files</td>
<td>Collection of RPFiles</td>
<td>RO</td>
<td>The files of this component.</td>
</tr>
<tr>
<td>includePath</td>
<td>String</td>
<td>RW</td>
<td>The path to standard headers to be linked with the component.</td>
</tr>
<tr>
<td>libraries</td>
<td>String</td>
<td>RW</td>
<td>The libraries to be linked with the component (for example, &quot;x.lib, y.lib, z.lib&quot;).</td>
</tr>
<tr>
<td>nestedComponents</td>
<td>Collection of RPComponent</td>
<td>RO</td>
<td>The components nested in this component.</td>
</tr>
<tr>
<td>path(fullPath)</td>
<td>String(path) Boolean(fullPath)</td>
<td>RO</td>
<td>The string containing the path to the component. If fullPath is True, the full path is returned:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&lt;drive&gt;:&lt;model dir&gt;&lt;component dir&gt;&lt;config dir&gt;)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If fullPath is False, the path relative to the project is returned:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&lt;component dir&gt;&lt;config dir&gt;</td>
</tr>
</tbody>
</table>
# Method Summary

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scopeElements</td>
<td>Collection of RPModelElement</td>
<td>RO</td>
<td>The logical elements allocated to this component.</td>
</tr>
<tr>
<td>standardHeaders</td>
<td>String</td>
<td>RW</td>
<td>The standard header files to be linked with the component.</td>
</tr>
<tr>
<td>addConfiguration</td>
<td></td>
<td></td>
<td>Adds a configuration to this component</td>
</tr>
<tr>
<td>addFile</td>
<td></td>
<td></td>
<td>Adds an empty file to the current component</td>
</tr>
<tr>
<td>addFolder</td>
<td></td>
<td></td>
<td>Adds an empty folder to the current component</td>
</tr>
<tr>
<td>addNestedComponent</td>
<td></td>
<td></td>
<td>Adds a component to the current component</td>
</tr>
<tr>
<td>addScopeElement</td>
<td></td>
<td></td>
<td>Places a model element within the scope of the current component</td>
</tr>
<tr>
<td>addToScope</td>
<td></td>
<td></td>
<td>Places the specified file, classes, and packages within the scope of the current component</td>
</tr>
<tr>
<td>allElementsInScope</td>
<td></td>
<td></td>
<td>Places all model elements within the scope of the current component</td>
</tr>
<tr>
<td>deleteConfiguration</td>
<td></td>
<td></td>
<td>Deletes the specified configuration from the current component</td>
</tr>
<tr>
<td>deleteFile</td>
<td></td>
<td></td>
<td>Deletes the specified file from the current component</td>
</tr>
<tr>
<td>findConfiguration</td>
<td></td>
<td></td>
<td>Retrieves the specified configuration in the current component</td>
</tr>
<tr>
<td>getConfigByDependency</td>
<td></td>
<td></td>
<td>Retrieves the appropriate configuration to use in the component on which the current component depends</td>
</tr>
<tr>
<td>getFile</td>
<td></td>
<td></td>
<td>Returns the file in which the specified classifier will be generated</td>
</tr>
<tr>
<td>getFileName</td>
<td></td>
<td></td>
<td>Retrieves the name of the file to which the specified classifier will be generated in this component</td>
</tr>
<tr>
<td>getModelElementFileName</td>
<td></td>
<td></td>
<td>Gets the file name of the specified model element</td>
</tr>
<tr>
<td>getPackageFile</td>
<td></td>
<td></td>
<td>Returns the package file</td>
</tr>
<tr>
<td>removeScopeElement</td>
<td></td>
<td></td>
<td>Deletes a scope element</td>
</tr>
<tr>
<td>setPath</td>
<td></td>
<td></td>
<td>Sets the path of the application built for this component</td>
</tr>
</tbody>
</table>
addConfiguration

Write method

Description

The `addConfiguration` method adds a configuration to the current component.

Visual Basic

Syntax

```vbnet
addConfiguration (name As String) As RPConfiguration
```

Arguments

name
The name of the new configuration

Return Value

The new configuration

C/C++ Prototype

```c
HRESULT addConfiguration (BSTR name,
IRPConfiguration** configuration)
```

Return Value

`HRESULT` (0 for success, or a signed integer error code)
addFile

Write method

Description

The addFile method adds an empty file to the current component.

Visual Basic

Syntax

addFile (name As String) As RPFile

Arguments

name

The name of the new file

Return Value

The file added to the component

C/C++ Prototype

HRESULT addFile (BSTR name, IRPFile** file)

Return Value

HRESULT (0 for success, or a signed integer error code)
addFolder

Write method

Description

The addFolder method adds an empty folder to the current component.

Visual Basic

Syntax

addFolder (name As String) As RPFile

Arguments

name

The name of the new folder

Return Value

The folder added to the component

C/C++ Prototype

HRESULT addFolder (BSTR name, IRPFile** file)

Return Value

HRESULT (0 for success, or a signed integer error code)
addNestedComponent

Write method

Description

The `addNestedComponent` method adds a component to the current component.

Visual Basic

Syntax

```plaintext
addNestedComponent (name As String) As RPComponent
```

Arguments

- `name`
  
  The name of the component to add

Return Value

The component added to the current component

C/C++ Prototype

```plaintext
HRESULT addNestedComponent (BSTR name,
  IRPComponent** pval)
```

Return Value

`HRESULT` (0 for success, or a signed integer error code)
addScopeElement

Write method

Description

The `addScopeElement` method places a model element within the scope of the current component.

Visual Basic

Syntax

`addScopeElement (pVal As RPModelElement)`

Arguments

pVal

The RPModelElement to place in the scope of the current component

C/C++ Prototype

`HRESULT addScopeElement (IRPModelElement* pVal)`

Return Value

`HRESULT (0 for success, or a signed integer error code)`
**addToScope**

Write method

**Description**

The `addToScope` method places the specified file, classes, and packages within the scope of the current component.

**Visual Basic**

**Syntax**

```visualbasic
addToScope (file As RPFile, 
classes As RPCollection, packages As RPCollection)
```

**Arguments**

- `file`  
The file to place in scope of the current component
- `classes`  
The classes to place in scope of the current component
- `packages`  
The packages to place in scope of the current component

**C/C++ Prototype**

```c
HRESULT addToScope (IRPFile* file, 
IRPCollection* classes, IRPCollection* packages)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
**allElementsInScope**

**Write method**

**Description**

The `allElementsInScope` method places all model elements within the scope of the current component.

**Visual Basic**

**Syntax**

`allElementsInScope()`

**C/C++ Prototype**

`HRESULT allElementsInScope()`

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
**deleteConfiguration**

**Write method**

**Description**

The `deleteConfiguration` method deletes the specified configuration from the current component.

**Visual Basic**

**Syntax**

```vbnet
deleteConfiguration (configuration As RPConfiguration)
```

**Arguments**

configuration

The configuration to delete

**C/C++ Prototype**

```c
HRESULT deleteConfiguration (IRPConfiguration* configuration)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
**deleteFile**

**Write method**

**Description**

The `deleteFile` method deletes the specified file from the current component.

**Visual Basic**

**Syntax**

```vbscript
deleteFile (file As RFile)
```

**Arguments**

- `file`
  
  The file to delete

**C/C++ Prototype**

```c
HRESULT deleteFile (IRPFile* file)
```

**Return Value**

- `HRESULT` (0 for success, or a signed integer error code)
**findConfiguration**

**Read method**

**Description**

The `findConfiguration` method retrieves the specified configuration in the current component.

**Visual Basic**

**Syntax**

```vbnet
findConfiguration (name As String) As RPConfiguration
```

**Arguments**

- `name` - The name of the configuration to retrieve

**Return Value**

The Rhapsody configuration

**C/C++ Prototype**

```c
HRESULT findConfiguration (BSTR name, IRPConfiguration** pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
**getConfigByDependency**

Used in cases where there are dependencies between components, this method retrieves the appropriate configuration to use in the component on which the current component depends. The argument required is the name of the dependency between the components.

**getFile**

*Read method*

**Description**

The `getFile` method returns the file in which the specified classifier will be generated.

**Visual Basic**

**Syntax**

```vbnet
getFile (c As RPClassifier, spec As Long) As RPFile
```

**Arguments**

- `c`  
  The classifier.  
- `spec` (1 or 0)  
  If this is set to 1, the file is a specification file.

**Return Value**

The file in which the specified classifier is generated.

**C/C++ Prototype**

```c
HRESULT getFile (IRPClassifier* c, long spec,  
IRPFile** pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
**getFileName**

Read method

**Description**

The `getFileName` method retrieves the name of the file to which the specified classifier will be generated in this component.

**Visual Basic**

**Syntax**

```vbnet
getFileName (c As RPClassifier, spec As Long, withExt As Long) As String
```

**Arguments**

- `c`  
  The classifier.
- `spec` (1 or 0)  
  If this is set to 1, the file is a specification file.
- `withExt` (1 or 0)  
  If this is set to 1, the file extension is included in the retrieval.

**Return Value**

The name of the file that contains the generated classifier

**C/C++ Prototype**

```c
HRESULT getFileName (IRPClassifier* c, long spec, long withExt, BSTR* pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
**getModelElementFileName**

**Read method**

**Description**

The **getModelElementFileName** method gets the file name of the specified model element.

**Visual Basic**

**Syntax**

```vba
getModelElementFileName (c As RPModelElement, long spec As Long, withExt As Long) As String
```

**Arguments**

- **c**
  The model element.
- **spec** (1 or 0)
  If this is set to 1, this is a specification file.
- **withExt** (1 or 0)
  If this is set to 1, the extension is included in the returned file name.

**Return Value**

The file name

**C/C++ Prototype**

```c
HRESULT getModelElementFileName (IRPModelElement *c, long spec, long withExt, BSTR *pVal)
```

**Return Value**

 HRESULT (0 for success, or a signed integer error code)
**getPackageFile**

**Read method**

**Description**

The `getPackageFile` method returns the package file.

**Visual Basic**

**Syntax**

```visualbasic
getPackageFile (c as RPPackage, spec As Long spec) As RPFile
```

**Arguments**

- `c`
  - The model element.
- `spec (1 or 0)`
  - If this is set to 1, this is a specification file.

**Return Value**

The file name

**C/C++ Prototype**

```c
HRESULT getPackageFile (IRPPackage* c, long spec, IRPFile** pVal)
```

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
**removeScopeElement**

*Write method*

**Description**

The `removeScopeElement` method deletes the scope element.

**Visual Basic**

**Syntax**

```vbs
removeScopeElement (pVal As RPModelElement)
```

**Arguments**

- `pVal`
  - The element to delete

**C/C++ Prototype**

```c
HRESULT removeScopeElement (IRPModelElement* pVal)
```

**Return Value**

- `HRESULT (0 for success, or a signed integer error code)`
**setPath**

**Write method**

**Description**

The `setPath` method sets the path of the application built for this component.

**Visual Basic**

**Syntax**

```
setPath (path As String)
```

**Arguments**

- `path`

  The path to which this component is built

**C/C++ Prototype**

```
HRESULT setPath (BSTR path)
```

**Return Value**

- `HRESULT (0 for success, or a signed integer error code)`
IRPComponentDiagram Interface

The IRPComponentDiagram interface represents a component diagram. It inherits from the IRPDiagram.

Currently, IRPComponentDiagram does not expose additional functionality to the diagram.

IRPComponentInstance Interface

The IRPComponentInstance interface represents a component instance. It inherits from the IRPComponent.

VB Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>componentType</td>
<td>RPComponent</td>
<td>RW</td>
<td>The component type</td>
</tr>
<tr>
<td>node</td>
<td>RPNode</td>
<td>RO</td>
<td>The node</td>
</tr>
</tbody>
</table>
The IRPConfiguration interface represents a code generation configuration within a given IRPComponent. It inherits from IRPModelElement.

**VB Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>additionalSources</td>
<td>String</td>
<td>RW</td>
<td>The additional source files to be compiled with this configuration.</td>
</tr>
</tbody>
</table>
| allElementsInInstrumentationScope | Long        | RW     | A Boolean value that reflects the All Elements and Selected Elements options of the instrumentation scope. The property defines the following accessor and mutator:  
  propget, HRESULT allElementsIn-Instrumentation-Scope ([out, retval] BOOL *pVal);  
  propput, HRESULT allElementsIn-Instrumentation-Scope ([in] BOOL newVal); |
<p>| buildSet                 | String        | RW     | The build set of this configuration (debug or release).                    |
| compilerSwitches         | String        | RW     | The compiler switches to be applied to this configuration in addition to those already specified in property <code>&lt;lang&gt;_CG::&lt;env&gt;::CPPCompileSwitches</code>. |
| generateCodeForActors     | Boolean       | RW     | If this is TRUE, code is generated for actors when this configuration is generated. |
| includePath              | String        | RW     | The path to standard headers to be linked with the configuration.           |
| initialInstances         | RPCollection  | RO     | The initial instances.                                                     |</p>
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>initializationCode</td>
<td>String</td>
<td>RW</td>
<td>The string containing the initialization code to be added to the main program after any initialization done by Rhapsody and before the main program loop.</td>
</tr>
<tr>
<td>instrumentationScope</td>
<td>RPCollection</td>
<td>RW</td>
<td>A container of elements in the selected instrumentation scope, if the All Elements option is selected.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The property defines the following accessor: propget, HRESULT instrumentationScope([out], retval] IRPCollection** pVal);</td>
</tr>
<tr>
<td>instrumentationType</td>
<td>String</td>
<td>RW</td>
<td>The type of instrumentation in this configuration (None, Trace, or Animate).</td>
</tr>
<tr>
<td>libraries</td>
<td>String</td>
<td>RW</td>
<td>The libraries to be linked with the component (for example, &quot;x.lib, y.lib, z.lib&quot;).</td>
</tr>
<tr>
<td>linkSwitches</td>
<td>String</td>
<td>RW</td>
<td>The link switches to be applied to the configuration in addition to those already specified in the property &lt;lang&gt;_CG::&lt;env&gt;::LinkSwitches.</td>
</tr>
<tr>
<td>path(fullPath)</td>
<td>String(path)</td>
<td>RO</td>
<td>The string containing the path to the component. If fullPath is true, the full path is returned: &lt;drive&gt;:&lt;model dir&gt;&lt;component dir&gt;&lt;config dir&gt;</td>
</tr>
<tr>
<td></td>
<td>Boolean(fullPath)</td>
<td></td>
<td>If fullPath is false, the path relative to the project is returned: &lt;component dir&gt;&lt;config dir&gt;</td>
</tr>
<tr>
<td>scopeType</td>
<td>String</td>
<td>RW</td>
<td>The scope type of the configuration (explicit or derived).</td>
</tr>
</tbody>
</table>
Rhapsody API Interfaces

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>standardHeaders</td>
<td>String</td>
<td>RW</td>
<td>The standard header files to be linked with the configuration.</td>
</tr>
<tr>
<td>statechartImplementation</td>
<td>String</td>
<td>RW</td>
<td>The statechart implementation of the configuration (flat or reusable).</td>
</tr>
<tr>
<td>timeModel</td>
<td>String</td>
<td>RW</td>
<td>The time model of the configuration (real or simulated).</td>
</tr>
</tbody>
</table>

**Method Summary**

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addInitialInstance</td>
<td>Adds an instance to the list of initial instances for the current configuration</td>
</tr>
<tr>
<td>addPackageToInstrumentationScope</td>
<td>Adds a classifier to the instrumentation scope</td>
</tr>
<tr>
<td>addToInstrumentationScope</td>
<td>Adds explicit initial instances to the instrumentation scope</td>
</tr>
<tr>
<td>deleteInitialInstance</td>
<td>Deletes an instance from the list of build instances for the current configuration</td>
</tr>
<tr>
<td>getDirectory</td>
<td>Retrieves the build directory specified for the current configuration</td>
</tr>
<tr>
<td>getItsComponent</td>
<td>Retrieves the component to which the current configuration belongs</td>
</tr>
<tr>
<td>getMainName</td>
<td>Retrieves the name of the file where the main() routine for the current configuration resides</td>
</tr>
<tr>
<td>getMakefileName</td>
<td>Retrieves the name of the makefile generated for the current configuration</td>
</tr>
<tr>
<td>getTargetName</td>
<td>Retrieves the build name of the file to be generated for the current configuration</td>
</tr>
<tr>
<td>removeFromInstrumentationScope</td>
<td>Removes the classifier from the instrumentation scope</td>
</tr>
<tr>
<td>removePackageFromInstrumentationScope</td>
<td>Removes the specified package from the instrumentation scope, including all its aggregated classes, actors, and nested packages</td>
</tr>
<tr>
<td>setDirectory</td>
<td>Sets the directory for the current configuration</td>
</tr>
<tr>
<td>setItsComponent</td>
<td>Sets the owning component for the current configuration</td>
</tr>
</tbody>
</table>
addInitialInstance

Write method

Description

The addInitialInstance method adds an instance to the list of initial instances for the current configuration.

Visual Basic

Syntax

addInitialInstance (newVal As RPModelElement)

Arguments

newVal

The new instance to add to list of initial instances for this configuration

C/C++ Prototype

HRESULT addInitialInstance (IRPModelElement* newVal)

Return Value

HRESULT (0 for success, or a signed integer error code)
addPackageToInstrumentationScope

Write method

Description

The `addPackageToInstrumentationScope` method adds the specified package to the instrumentation scope, including all its aggregated classes, actors, and nested packages.

Visual Basic

Syntax

```visualbasic
addPackageToInstrumentationScope (pVal As RPPackage)
```

Arguments

`pVal`

The package to add to the instrumentation scope

C/C++ Prototype

```c
HRESULT addPackageToInstrumentationScope (IRPPackage* pVal)
```

Return Value

`HRESULT (0 for success, or a signed integer error code)`
**addToInstrumentationScope**

**Write method**

**Description**

The `addToInstrumentationScope` method adds explicit initial instances to the instrumentation scope.

Beginning with Version 5.0, Rhapsody 6.1 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 6.1 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).

**Visual Basic**

**Syntax**

```vbnet
addToInstrumentationScope (pVal As RPClassifier)
```

**Arguments**

- **pVal**
  - The initial instance to add to the instrumentation scope

**C/C++ Prototype**

```c
HRESULT addToInstrumentationScope (IRPClassifier* pVal)
```

**Return Value**

- `HRESULT` (0 for success, or a signed integer error code)
deleteInitialInstance

Write method

Description

The deleteInitialInstance method deletes an instance from the list of build instances for the current configuration.

Visual Basic

Syntax

deleteInitialInstance (newVal As RPModelElement)

Arguments

NewVal

The initial instance to delete from list of initial instances for this configuration

C/C++ Prototype

HRESULT deleteInitialInstance (IRPModelElement* newVal)

Return Value

HRESULT (0 for success, or a signed integer error code)
getDirectory

Read method

Description

The getDirectory method retrieves the build directory specified for the current configuration.

Visual Basic

Syntax

gendir (fullPath As Long, newName As String) As String

Arguments

    fullPath
    If this is 1, the returned directory contains the full path.
    newName
    Reserved for future use.

Return Value

The build directory for the current configuration

C/C++ Prototype

HRESULT getDirectory (long fullPath, BSTR newName, BSTR* retVal)

Return Value

HRESULT (0 for success, or a signed integer error code)
getItsComponent
   Read method

Description
   The getItsComponent method retrieves the component to which the current configuration belongs.

Visual Basic

Syntax
   getItsComponent() As RPComponent

Return Value
   The component to which this configuration belongs

C/C++ Prototype

HRESULT getItsComponent (IRPComponent** pVal)

Return Value
   HRESULT (0 for success, or a signed integer error code)
getMainName

Read method

Description

The getMainName method retrieves the name of the file where the main() routine for the current configuration resides.

Visual Basic

Syntax

    getMainName() As String

Return Value

The location of the file that contains the main()

C/C++ Prototype

    HRESULT getMainName (BSTR* pVal)

Return Value

    HRESULT (0 for success, or a signed integer error code)
getMakefileName

Read method

Description

The `getMakefileName` method retrieves the name of the makefile generated for the current configuration.

Visual Basic

Syntax

```vbnet
getMakefileName (fullPath As Long) As String
```

Arguments

`fullPath`

Set this to one of the following values:

1--Return the full path.

0--Return the path relative to the project directory.

Return Value

The name of the makefile

C/C++ Prototype

```c
HRESULT getMakefileName (long fullPath, BSTR* pVal)
```

Return Value

`HRESULT (0 for success, or a signed integer error code)`
**getTargetName**

*Read method*

**Description**

The `getTargetName` method retrieves the build name of the file to be generated for the current configuration.

**Visual Basic**

**Syntax**

```visualbasic
getTargetName (fullPath As Long) As String
```

**Arguments**

`fullPath`  
Set this to one of the following values:  
1--Return the full path.  
0--Return the path relative to the project directory.

**Return Value**

The name of the build file (for example, BuildName.exe or BuildName.lib)

**C/C++ Prototype**

```c
HRESULT getTargetName (long fullPath, BSTR* pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
**removeFromInstrumentationScope**

**Write method**

**Description**

The `removeFromInstrumentationScope` method removes the classifier from the instrumentation scope.

**Visual Basic**

**Syntax**

```visualbasic
removeFromInstrumentationScope (pVal As RPClassifier)
```

**Arguments**

- `pVal`  
  The classifier to remove from the instrumentation scope

**C/C++ Prototype**

```c
HRESULT removeFromInstrumentationScope (
    IRPClassifier *pVal)
```

**Return Value**

- `HRESULT (0 for success, or a signed integer error code)`
removePackageFromInstrumentationScope

Write method

Description

The `removePackageFromInstrumentationScope` method removes the specified package from the instrumentation scope, including all its aggregated classes, actors, and nested packages.

Visual Basic

Syntax

`removePackageFromInstrumentationScope (pVal As RPPackage)`

Arguments

- `pVal`
  
  The package to remove from the instrumentation scope

C/C++ Prototype

```c
HRESULT removePackageFromInstrumentationScope (IRPPackage* pVal)
```

Return Value

- `HRESULT` (0 for success, or a signed integer error code)
**setDirectory**

**Note**

Currently, this method has not been implemented.

**Write method**

**Description**

The `setDirectory` method sets the directory for the current configuration.

**Visual Basic**

**Syntax**

```visualbasic
setDirectory (fullpath As Long, newName As String)
```

**Arguments**

- `fullpath` (1 or 0)
  
  Set this to 1 to include the full directory path.

- `newName`
  
  The new name for the directory.

**C/C++ Prototype**

```c
HRESULT setDirectory (long fullpath, BSTR newName)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
**setItsComponent**

**Write method**

**Description**

The `setItsComponent` method sets the owning component for the current configuration.

**Visual Basic**

**Syntax**

```
setItsComponent (newVal As RPComponent)
```

**Arguments**

- **newVal**
  - The new owner component for this configuration

**C/C++ Prototype**

```
HRESULT setItsComponent (IRPComponent* newVal)
```

**Return Value**

- **HRESULT (0 for success, or a signed integer error code)**
IRPConnector Interface

The IRPConnector interface represents a connector in a statechart diagram. It inherits from IRPStateVertex.

VB Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connectorType</td>
<td>String</td>
<td>RW</td>
<td>The connector type (Termination, History, Condition, Fork, Join, or Unknown)</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getDerivedInEdges</td>
<td>Retrieves the incoming transitions for the connector</td>
</tr>
<tr>
<td>getDerivedOutEdge</td>
<td>Retrieves the outgoing transition for the connector</td>
</tr>
<tr>
<td>getOfState</td>
<td>Returns the state connected to the current connector if it is a history connector</td>
</tr>
<tr>
<td>isConditionConnector</td>
<td>Determines whether the current connector is a condition connector</td>
</tr>
<tr>
<td>isDiagramConnector</td>
<td>Determines whether the current connector is a diagram connector</td>
</tr>
<tr>
<td>isForkConnector</td>
<td>Determines whether the current connector is a fork synch bar connector</td>
</tr>
<tr>
<td>isHistoryConnector</td>
<td>Determines whether the current connector is a history connector</td>
</tr>
<tr>
<td>isJoinConnector</td>
<td>Determines whether the current connector is a join synch bar connector</td>
</tr>
<tr>
<td>isJunctionConnector</td>
<td>Determines whether the current connector is a junction connector</td>
</tr>
<tr>
<td>isStubConnector</td>
<td>Determines whether the current connector is a stub connector</td>
</tr>
<tr>
<td>isTerminationConnector</td>
<td>Determines whether the current connector is a termination connector</td>
</tr>
<tr>
<td>setState</td>
<td>Updates the source state of the current connector with a new state</td>
</tr>
</tbody>
</table>
**getDerivedInEdges**

Read method

Description

The `getDerivedInEdges` method retrieves the incoming transitions for the connector.

Visual Basic

Syntax

```visual-basic
getDerivedInEdges() As RPCollection
```

Return Value

The incoming transitions

C/C++ Prototype

```c
HRESULT getDerivedInEdges (IRPCollection** pVal)
```

Return Value

`HRESULT` (0 for success, or a signed integer error code)
getDerivedOutEdge

Read method

Description

The `getDerivedOutEdge` method retrieves the outgoing transition for the connector.

Visual Basic

Syntax

```
getDerivedOutEdge() As Transition
```

Return Value

The outgoing transition

C/C++ Prototype

```
HRESULT getDerivedOutEdge (IRPTransition** pVal)
```

Return Value

`HRESULT` (0 for success, or a signed integer error code)
getOfState

Read method

Description

The `getOfState` method returns the state connected to the current connector if it is a history connector. This is the state for which the history connector maintains historical state information.

Visual Basic

Syntax

```vbnet
getOfState() As RPState
```

Return Value

The state for which the history connector maintains state information

C/C++ Prototype

```c
HRESULT getOfState (IRPState** pVal)
```

Return Value

`HRESULT (0 for success, or a signed integer error code)`
isConditionConnector

Read method

Description

The isConditionConnector method determines whether the current connector is a condition connector.

Visual Basic

Syntax

isConditionConnector() As Long

Return Value

1 if the connector is a condition connector; 0 otherwise

C/C++ Prototype

HRESULT isConditionConnector (long* pVal)

Return Value

HRESULT (0 for success, or a signed integer error code)
isDiagramConnector

Read method

Description

The isDiagramConnector method determines whether the current connector is a diagram connector.

Visual Basic

Syntax

isDiagramConnector() As Long

Return Value

1 if the connector is a diagram connector; 0 otherwise

C/C++ Prototype

HRESULT isDiagramConnector (long* pVal)

Return Value

HRESULT (0 for success, or a signed integer error code)
**isForkConnector**

**Read method**

**Description**

The `isForkConnector` method determines whether the current connector is a fork synch bar connector.

**Visual Basic**

**Syntax**

```vbnet
isForkConnector() As Long
```

**Return Value**

1 if the connector is a fork synch bar connector; 0 otherwise

**C/C++ Prototype**

```c
HRESULT isForkConnector (long* pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
**isHistoryConnector**

*Read method*

**Description**

The `isHistoryConnector` method determines whether the current connector is a history connector.

**Visual Basic**

**Syntax**

```vbs
isHistoryConnector() As Long
```

**Return Value**

1 if the connector is a history connector; 0 otherwise

**C/C++ Prototype**

```c
HRESULT isHistoryConnector (long* pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
**isJoinConnector**

**Read method**

**Description**

The **isJoinConnector** method determines whether the current connector is a join synch bar connector.

**Visual Basic**

**Syntax**

```vbnet
isJoinConnector() As Long
```

**Return Value**

1 if the connector is a join synch bar connector; 0 otherwise

**C/C++ Prototype**

```c
HRESULT isJoinConnector (long* pVal)
```

**Return Value**

HRESULT (0 for success, or a signed integer error code)
**isJunctionConnector**

**Read method**

**Description**

The `isJunctionConnector` method determines whether the current connector is a junction connector.

**Visual Basic**

**Syntax**

`isJunctionConnector() As Long`

**Return Value**

1 if the connector is a junction connector; 0 otherwise

**C/C++ Prototype**

`HRESULT isJunctionConnector (long* pVal)`

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
isStubConnector

**Read method**

**Description**

The `isStubConnector` method determines whether the current connector is a stub connector.

**Visual Basic**

**Syntax**

```vbnet
isStubConnector() As Long
```

**Return Value**

1 if the connector is a stub connector; 0 otherwise

**C/C++ Prototype**

```c
HRESULT isStubConnector (long* pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)

isTerminationConnector

**Read method**

**Description**

The `isTerminationConnector` method determines whether the current connector is a termination connector.

**Visual Basic**

**Syntax**

```vbnet
isTerminationConnector() As Long
```

**Return Value**

1 if the connector is a termination connector; 0 otherwise

**C/C++ Prototype**

```c
HRESULT isTerminationConnector (long* pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
setOfState
   Write method

Description
   The setOfState method updates the source state of the current connector with a new state.

Visual Basic
Syntax
   setOfState (OfState As RPState)
Arguments
   OfState
       The new source state for the connector

C/C++ Prototype
   HRESULT setOfState (IRPState* OfState)
Return Value
   HRESULT (0 for success, or a signed integer error code)
IRPConstraint Interface

The `IRPConstraint` interface represents a constraint in a Rhapsody model. It inherits from `IRPAnnotation`.

**VB Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>String</td>
<td>RW</td>
<td>The body of the constraint.</td>
</tr>
<tr>
<td>constraintsByMe</td>
<td>Collection of</td>
<td>RO</td>
<td>The model elements affected by this constraint.</td>
</tr>
<tr>
<td></td>
<td>RPModelElements</td>
<td></td>
<td>For example, if a constraint says that each Airplane must have at least</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>two Pilots, this collection will contain both the Airplane and Pilot</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>classes.</td>
</tr>
</tbody>
</table>

IRPControlledFile

Represents controlled files.

**fullPathFileName**

Property that represents the full path of the file.

**open**

Method that can be used to open the controlled file.
IRPDependency Interface

The IRPDependency interface represents the dependencies between model elements, for example, in terms of either an include or a friend relationship between classes. It inherits from IRPModelElement.

**VB Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dependent</td>
<td>RPModelElement</td>
<td>RW</td>
<td>The source element in the dependency relation</td>
</tr>
<tr>
<td>dependsOn</td>
<td>RPModelElement</td>
<td>RW</td>
<td>The target element in the dependency relation</td>
</tr>
</tbody>
</table>

IRPDeploymentDiagram Interface

The IRPDeploymentDiagram interface represents deployment diagrams. It inherits from IRPDiagram.

IRPDiagram Interface

The IRPDiagram interface is an abstract interface that provides the common functionality of Rhapsody diagrams. Currently, the functionality provided by IRPDiagram (in addition to IRPModelElement) is to render the view as a metafile. This class inherits from IRPUnit, because diagrams are also units.

**Method Summary**

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getElementsInDiagram</td>
<td>Returns a collection of all the model elements in the current diagram</td>
</tr>
<tr>
<td>getPicture</td>
<td>Renders this diagram into the specified extended metafile</td>
</tr>
<tr>
<td>getPictureAs</td>
<td>Saves a Rhapsody diagram in a specific graphic format.</td>
</tr>
<tr>
<td>getPictureAsDividedMetafiles</td>
<td>Enables you to split a large diagram into several metafiles when you export it</td>
</tr>
</tbody>
</table>
**getElementsInDiagram**

**Read method**

**Description**

The `getElementsInDiagram` method returns a collection of all the model elements in the current diagram.

**Visual Basic**

**Syntax**

```vbnet
getElementsInDiagram() As RPCollection
```

**Return Value**

A collection of all the model elements in the diagram

**C/C++ Prototype**

```c
HRESULT getElementsInDiagram (IRPCollection** pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)

**getPicture**

**Read method**

**Description**

The `getPicture` method renders this diagram into the specified extended metafile.

**Note:** If the file cannot be written, this method flags the error.

**Visual Basic**

**Syntax**

```vbnet
getPicture (filename As String)
```

**Arguments**

- `filename`

The name of the metafile that will contain the current diagram. The format of the created metafile is `.emf`. The created metafile is used later by the VB function `LoadPicture`, which creates a VB function object that can be used for placing pictures in documents.
C/C++ Prototype

HRESULT getPicture (BSTR filename)

Return Value

HRESULT (0 for success, or a signed integer error code)

getPictureAs

This method can be used to save a Rhapsody diagram in a specific graphic format. The method can also be used to retrieve diagram element information that can be used to create an HTML image map.

The method returns a list of the graphic files created.

getPictureAs(firstFileName As String, imageFormat As String, getImageMaps As Long, diagrammap As RPCollection, fileNames As RPCollection) As RPCollection

firstFileName

The naming convention to use for the files that will be created. For a detailed explanation, see getPictureAsDividedMetafiles.

imageFormat

The graphic format in which the diagram should be saved. This can be one of the following: EMF, BMP, JPEG, JPG, TIFF.

getImageMaps

Use this argument to indicate whether the function should also return a collection of objects that can be used to construct an HTML image map for the diagram. (Use 1 if you want this information, else use 0.)

diagrammap

The collection to use when returning objects containing the required information for constructing an HTML image map.

fileNames

The collection to use for the names of the graphic files created.
getPictureAsDividedMetafiles

Read method

Description

The `getPictureAsDividedMetafiles` method enables you to split a large diagram into several metafiles when you export it.

This method is influenced by the property `General::Graphics::ExportedDiagramScale`. See the definition provided for the property on the applicable Properties tab of the Features dialog box.

Note: If the file cannot be written, this method flags the error.

Visual Basic

Syntax

```visualbasic
getPictureAsDividedMetafiles (firstFileName As String) As RPCollection
```

Arguments

- `firstFileName` The naming convention for the created files. For example, if you passed the value "Foo" as the `firstFileName`:
  - If the diagram can be drawn on one page, the name of the metafile is `Foo`.
  - If the diagram is split into multiple pages, the first file will be named `FooZ_X_Y`. The variables used in the name are as follows:
    - `Z`—The number of the created file
    - `X`—The number of the page along the X vector
    - `Y`—The number of the page along the Y vector
  - For example, the file `Foo2_1_2` means that this is the second metafile created and it contains one page, which is the second page along the Y vector (the X vector is 1).
  - All the file names will be inserted in the sent strings list (fileNames).

Return Value

A collection that contains the names of the files that were created
**C/C++ Prototype**

```c
HRESULT getPictureAsDividedMetafiles (
  [in] BSTR firstFileName,
  [out, retval] IRPCollection** fileNames)
```

**Return Value**

HRESULT (0 for success, or a signed integer error code)

**VBA Sample**

```vba
Private Sub CommandButton1_Click()
Dim proj As RPProject
Dim d As RPDiagram
Dim col As RPCollection
On Error GoTo aa
Set proj = getProject
Set d = proj.findNestedElementRecursive(  
  "Dishwasher Cycle", "SequenceDiagram")
Set col = d.getPictureAsDividedMetafiles(  
  "D:\Temp\Diagram.emf")
Exit Sub
aa:
MsgBox errorMessage
End Sub
```
IRPEnumerationLiteral Interface

The IRPEnumerationLiteral interface supports the language-independent types introduced in Rhapsody 5.0. It inherits from IRPModelElement.

VB Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td>RPEvent</td>
<td>RW</td>
<td>An optional value for the literal</td>
</tr>
</tbody>
</table>
IRPEvent Interface

The IRPEvent interface represents an event. It derives from IRPInterfaceItem.

VB Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseEvent</td>
<td>RPEvent</td>
<td>RW</td>
<td>The pointer to the base event (if this event is inherited from it).</td>
</tr>
<tr>
<td>superEvent</td>
<td>RPEvent</td>
<td>RW</td>
<td>The pointer to the super event (if this event is inherited from it)</td>
</tr>
</tbody>
</table>
|         |        |        | As a read method, superEvent() provides the base event that an event was derived from. Thus, if event B is inherited from event A, B.superEvent() returns a pointer to A. As a write method, superEvent() inherits or reinherit an event from a new base (super) event. Thus, if you want event B to be inherited from A, set B.superEvent() = A.
IRPEventReception Interface

The IRPEventReception interface represents a relationship between a class and an event that is part of its interface. It derives from IRPInterfaceItem.

Method Summary

| getEvent | Returns the event for the current event reception that serves as part of the interface for a class |

getEvent

Read method

Description

The `getEvent` method returns the event for the current event reception that serves as part of the interface for a class.

Visual Basic

Syntax

```
getEvent() As RPEvent
```

Return Value

The `RPEvent` related to a class through the event reception interface

C/C++ Prototype

```
HRESULT getEvent (IRPEvent** pVal)
```

Return Value

`HRESULT (0 for success, or a signed integer error code)`
IRPExecutionOccurrence Interface

The IRPExecutionOccurrence interface represents an execution occurrence in a sequence diagram. It derives from IRPModelElement.

VB Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>RPMessage</td>
<td>RO</td>
<td>The start message for the execution occurrence</td>
</tr>
</tbody>
</table>

IRPExternalCodeGenerator Interface

The IRPExternalCodeGenerator interface is a dispatch interface that defines events that must be handled by the external code generator.

The interface inherits from IDispatch.

Using an External Code Generator

Beginning with Version 4.1, you can integrate an external code generator with Rhapsody 6.1. The code generator application is loaded when Rhapsody 6.1 is loaded. This code generator should be a full-featured code generator that can generate all the model code. When you specify an external code generator, Rhapsody 6.1 does not generate any code. Rhapsody 6.1 in Ada uses an external code generator.

You can set the environment variable ExternalGenerator in the [codegen] section of the rhapsody.ini file to the path of the external code generator executable. This executable will be loaded when Rhapsody 6.1 is loaded and terminates when Rhapsody 6.1 exits. If you do not set this environment variable, you must manually load your code generator after Rhapsody 6.1 is loaded. Note the following:

- This variable setting applies only to full-featured external code generators.
- If you do not load your external code generator, it cannot display messages in the Rhapsody 6.1 output window.

In addition, you can integrate makefiles generated by a makefile generator other than the Rhapsody 6.1 generator; all other code generation is done by Rhapsody 6.1.
Restrictions

Note the following restrictions:

- Because the active code view uses the annotations generated by Rhapsody 6.1 to find the location of a model element in a source file, searching a file generated by an external code generator (unannotated) might not be accurate. There are other annotation issues concerning roundtrip and error highlighting. Therefore, the external code generator must generate annotations to make all of these features work properly.
- If you specify an external code generator, you cannot use the *CG In Browser* feature to generate code.
- You can integrate a single external code generator with one instance of a Rhapsody 6.1 application, running on the same machine.
- You can integrate an external code generator with Rhapsody 6.1 on a Solaris platform only if the client supports the COM framework.
- This functionality is supported only by Rhapsody 6.1 Developer Edition as a separate, add-on feature.

Event Handling

When you trigger code generator operations, Rhapsody 6.1 fires events that are handled by the registered, external code generator. The following table lists the different events and when they are fired.

<table>
<thead>
<tr>
<th>Event</th>
<th>When Fired</th>
</tr>
</thead>
</table>
| **generate**| When you invoke any kind of generation command (forced or incremental), for selected classifiers, files, or for the entire configuration. The invocation can be explicit or by DMCA.  
|             | When called, the external code generator generates the elements according to the settings for the active configuration.  
|             | This method is called with all model elements that need to be generated.  
| **Abort**   | Is invoked when the user selects the *Abort* option during code generation. When the external code generator receives this event, it stops the code generation process and notifies Rhapsody 6.1 that it is done.  
| **getFileName** | Is invoked when Rhapsody 6.1 needs the file name and path of a model element.  
|             | If the event is not handled, Rhapsody 6.1 displays an error message stating that it could not get the file name from the external code generator.  
|             | Note that if the external code generator uses the same file mapping scheme as Rhapsody 6.1, you do not need to implement this event. |
Implementing the External Code Generator

To implement an external code generator, follow these steps:

1. Implement the event handlers for the `IRPExternalCodeGenerator` events:
   - Invoke the code generation process on another thread to return from the call to `generate` as soon as possible.
   - Notify the `IRPExternalCodeGeneratorInvoker` when the generation session has ended.

2. Instantiate your event handler class when the external code generator is loaded.

3. Get the `IRPApplication` object.

4. Get the `IRPExternalCodeGeneratorInvoker` singleton from the `IRPApplication` interface. See the method `getTheExternalCodeGeneratorInvoker` for more information.

5. Register the implemented `IRPExternalCodeGenerator` as the external code generator on the `IRPExternalCodeGeneratorInvoker` interface.

6. Print code generation messages using standard output. For example:
   ```
cout<<"Generating"<<class_name<<endl;
```

7. Terminate the external code generator process when `Exit` is called.

<table>
<thead>
<tr>
<th>Event</th>
<th>When Fired</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GetMainFileName</strong></td>
<td>Is invoked when Rhapsody 6.1 needs the main file name and path for a configuration. When the event is not handled, Rhapsody 6.1 displays an error message stating that it could not get the file name from the external code generator.</td>
</tr>
<tr>
<td><strong>GetTargetfileName</strong></td>
<td>Is invoked when Rhapsody 6.1 needs the target name and path for a configuration. When the event is not handled, Rhapsody 6.1 displays an error message stating that it could not get the makefile name from the external code generator. Note that if the external code generator uses the same file mapping scheme as Rhapsody 6.1, you do not need to implement this event.</td>
</tr>
<tr>
<td><strong>WhoAmI</strong></td>
<td>Is invoked to identify the external code generator.</td>
</tr>
<tr>
<td><strong>Exit</strong></td>
<td>Is invoked before Rhapsody 6.1 exits. When the external code generator receives this event, it performs the necessary cleanup and terminates its process.</td>
</tr>
</tbody>
</table>
Rhapsody Settings

You must set the following environment variables and properties:

- Set the `ExternalGenerator` environment variable in the `rhapsody.ini` file to the path to the implemented code generator executable. See “Using an External Code Generator” for more information.

- Set the `<lang>_CG::<Environment>::CodeGeneratorTool` property for the configuration that should be generated with the external code generator.

- Set the `<lang>_CG::Configuration::ExternalGenerationTimeout` property with a reasonable time for an average class generation session.

See the definition provided for the property on the applicable Properties tab of the Features dialog box.
Sample

////////////////////////////////////////////////////
// MyCodeGenerator.h: interface for the CMyCodeGenerator
// class.
////////////////////////////////////////////////////
...
class CMyCodeGenerator:
public
IDispEventImpl<1,CMyCodeGenerator,
&DIID_IRPExternalCodeGenerator, &LIBID_rhapsody,1,0>
{
  public:
    CMyCodeGenerator();
    virtual ~CMyCodeGenerator();
    void Register();

    //event handlers
    HRESULT __stdcall Generate(
        IDispatch* configuration,
        IDispatch* classifiers, IDispatch* files,
        BOOL genMain, BOOL genMake);
    BSTR __stdcall WhoAmI();
    BSTR __stdcall GetFileName(IDispatch* modelElement,
        IDispatch* configuration, int pathType,
        BOOL withExt);
    BSTR __stdcall GetTargetfileName(IDispatch*
        configuration, int pathType, BOOL withExt);
    BSTR __stdcall GetMainFileName(
        IDispatch* configuration, int pathType, BOOL withExt);
    BSTR __stdcall GetMakefileName(
        IDispatch* configuration, int pathType, BOOL withExt);
    VOID __stdcall OnExit();
    VOID __stdcall Abort();
BEGIN_SINK_MAP(CMyCodeGenerator)
  SINK_ENTRY_EX(/*nID=*/ 1,
    DIID_IRPExternalCodeGenerator,
    /*dispid=*/ 0x1, Generate)
  SINK_ENTRY_EX(/*nID=*/ 1,
    DIID_IRPExternalCodeGenerator,
    /*dispid=*/ 0x2, OnExit)
  SINK_ENTRY_EX(/*nID=*/ 1,
    DIID_IRPExternalCodeGenerator,
    /*dispid=*/ 0x3, GetFileName)
  SINK_ENTRY_EX(/*nID=*/ 1,
    DIID_IRPExternalCodeGenerator,
    /*dispid=*/ 0x4, GetTargetfileName)
  SINK_ENTRY_EX(/*nID=*/ 1,
    DIID_IRPExternalCodeGenerator,
    /*dispid=*/ 0x5, GetMainFileName)
  SINK_ENTRY_EX(/*nID=*/ 1,
    DIID_IRPExternalCodeGenerator,
    /*dispid=*/ 0x6, GetMakefileName)
  SINK_ENTRY_EX(/*nID=*/ 1,
    DIID_IRPExternalCodeGenerator,
    /*dispid=*/ 0x7, WhoAmI)
END_SINK_MAP(CMyCodeGenerator)
Rhapsody API Interfaces

```
SINK_ENTRY_EX(/*nID =*/ 1,
    DIID_RPExternalCodeGenerator,
    /*dispid =*/ 0x8, Abort)
END_SINK_MAP()
```

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Abort</strong></td>
<td>Is invoked when the user selects the <strong>Abort</strong> option during code generation</td>
</tr>
<tr>
<td><strong>Exit</strong></td>
<td>Is invoked before Rhapsody 6.1 exits</td>
</tr>
<tr>
<td><strong>generate</strong></td>
<td>Is invoked whenever a generation command of any kind is invoked</td>
</tr>
<tr>
<td><strong>getFileName</strong></td>
<td>Is invoked when Rhapsody 6.1 needs the file name and path of a model element</td>
</tr>
<tr>
<td><strong>GetMainFileName</strong></td>
<td>Is invoked when Rhapsody 6.1 needs the main file name and path for a configuration</td>
</tr>
<tr>
<td><strong>getMakefileName</strong></td>
<td>Is invoked when Rhapsody 6.1 needs the makefile name and path for a configuration</td>
</tr>
<tr>
<td><strong>GetTargetfileName</strong></td>
<td>Is invoked when Rhapsody 6.1 needs the target name and path for a configuration</td>
</tr>
<tr>
<td><strong>WhoAmI</strong></td>
<td>Is invoked to identify the external code generator</td>
</tr>
</tbody>
</table>
Abort

Description

The Abort event is invoked when the user selects the Abort option during code generation. When the external code generator receives this event, it stops the code generation process and notifies Rhapsody 6.1 that it is done.

Visual Basic

Syntax

Event Abort()

C/C++ Prototype

void Abort()

Exit

Description

The Exit event is invoked before Rhapsody 6.1 exits. When the external code generator receives this event, it performs the necessary cleanup and terminates its process.

Visual Basic

Syntax

Event Exit()

C/C++ Prototype

void Exit()
The `generate` event is invoked whenever a generation command of any kind is invoked (including forced or incremental generation for selected classifiers; or files for the entire configuration either explicitly by the user or by DMCA).

When called, the external code generator generates the elements according to the settings for the active configuration.

### Visual Basic

#### Syntax

Event `generate (activeConfiguration As Object, classifiersCollection As Object, filesCollection As Object, generateMainFile As Long, generateMakefile As Long)`

#### Arguments

- **activeConfiguration**
  
  A pointer to the active configuration for this generation session. If this value is not NULL, configuration files (main and make) are generated.

  The external code generator queries the `activeConfiguration` for its `RPConfiguration` interface.

- **classifiersCollection**
  
  The container of classes and package interfaces to be generated. The container can be NULL if no classifiers need to be generated. Packages in this container are generated without their aggregates (the package’s classes).

  The external code generator queries the `classifiersCollection` for its `RPCollection` interface.

- **filesCollection**
  
  The container of file and folder interfaces (RPFiles) to be generated. The container can be NULL if no files need to be generated.

  Model elements that are mapped to a file or folder in the `filesCollection` container will be added to the `classifiersCollection`. Therefore, the external generator does not query the file for its mapped classifiers. However, the code generator does check the files for text elements.

  The external code generator queries the `filesCollection` for its `RPCollection` interface.

- **generateMainFile** (1 or 0)
  
  Set this to 1 to generate the main configuration files.

- **generateMakefile** (1 or 0)
Set this to 1 to generate the makefile for the configuration.

**C/C++ Prototype**

```c
HRESULT generate (IDispatch* activeConfiguration,
                 IDispatch* classifiersCollection,
                 IDispatch* filesCollection, long generateMainFile,
                 long generateMakefile)
```

**Return Value**

`S_OK` for success, or an error code. If an error occurs, code generation is aborted.
getFileName

Description

The `getFileName` method is invoked when Rhapsody 6.1 needs the file name and path of a model element.

If the event is not handled, Rhapsody 6.1 displays an error message stating that it could not get the file name from the external code generator.

**Note:** If the external code generator uses the same file mapping scheme as Rhapsody 6.1, you do not need to implement this event.

Visual Basic

Syntax

Event getFileName (modelElement As Object, configuration As Object, pathType As Long, withExtensions As Long)

Arguments

- **modelElement**
  - The model element whose name you want to retrieve. The model element can be a class, actor, package, event, or file.
  - The external code generator queries the `modelElement` for its `RPModelElement` interface.

- **configuration**
  - The configuration for which the file name is requested.
  - The external code generator queries the `configuration` for its `RPConfiguration` interface.

- **pathType**
  - The requested path format. The possible values are as follows:
    1. Include the full path. For example: `C:\Project\Component\Config\Class1.h`
    2. Include only the name of the file. For example: `Class1.h`
    3. Include the path relative from the project directory. For example: `Component\Config\Class1.h`
    4. Include the path relative from the active configuration to the requested file.
For example, if the file is located under
C:\Project\Component\Subfolder\Class1.h, the external code generator
includes the following path: Subfolder\Class1.h.

withExtensions

Specifies whether to include the extension in the returned file name. For example,
Class1.h instead of Class1.

Return Value

The file names of the model elements, separated by commas. If there is more than one file in
the list, Rhapsody 6.1 assumes that the first file is the specification file and the others are
implementation files.

C/C++ Prototype

BSTR getFileName (IDispatch* modelElement,
IDispatch* configuration, int pathType,
long withExtensions)
GetMainFileName

Description

The GetMainFileName method is invoked when Rhapsody 6.1 needs the main file name and path for a configuration.

If the event is not handled, Rhapsody 6.1 displays an error message stating that it could not get the file name from the external code generator.

Visual Basic

Syntax

Event GetMainFileName (configuration As Object, pathType As Long, withExtensions As Long)

Arguments

configuration

The configuration for which the main file name is requested.

The external code generator queries the configuration for its RPConfiguration interface.

pathType

The requested path format. The possible values are as follows:

1—Include the full path. For example: C:\Project\Component\Config\Class1.h

2—Include only the name of the file. For example: Class1.h

3—Include the path relative from the project directory. For example: Component\Config\Class1.h

4—Include the path relative from the active configuration to the requested file.

For example, if the file is located under C:\Project\Component\Subfolder\Class1.h, the external code generator will include the following path: Subfolder\Class1.h.

withExtensions

Specifies whether to include the extension in the returned file name. For example, mainfile.cpp instead of mainfile.
Return Value

The main file names of the model elements, separated by commas. If there is more than one file in the list, Rhapsody 6.1 assumes that the first file is the specification file and the second is the implementation file.

C/C++ Prototype

BSTR GetMainFileName (IDispatch* configuration, int pathType, long withExtensions)
getMakefileName

Description

The `getMakefileName` method is invoked when Rhapsody 6.1 needs the makefile name and path for a configuration.

If the event is not handled, Rhapsody 6.1 displays an error message stating that it could not get the makefile name from the external code generator.

Visual Basic

Syntax

Event getMakefileName (configuration As Object, pathType As Long, withExtensions As Long)

Arguments

configuration

The configuration for which the file name is requested.

The external code generator queries the `configuration` for its `RPConfiguration` interface.

pathType

The requested path format. The possible values are as follows:

1—Include the full path. For example: `C:\Project\Component\Config\Class1.h`

2—Include only the name of the file. For example: `Class1.h`

3—Include the path relative from the project directory. For example: `Component\Config\Class1.h`

4—Include the path relative from the active configuration to the requested file.

For example, if the file is located under `C:\Project\Component\Subfolder\Class1.h`, the external code generator will include the following path: `Subfolder\Class1.h`.

withExtensions

Specifies whether to include the extension in the returned file name. For example, `makefile.mak` instead of `makefile`.

Return Value

The name of the makefile
C/C++ Prototype

BSTR getMakefileName (IDispatch* configuration,
int pathType, long withExtensions)

GetTargetfileName

Description

The GetTargetfileName method is invoked when Rhapsody 6.1 needs the target name and path for a configuration.

If the event is not handled, Rhapsody 6.1 displays an error message stating that it could not get the makefile name from the external code generator.

Note: If the external code generator uses the same file mapping scheme as Rhapsody 6.1, you do not need to implement this event.

Visual Basic

Syntax

Event GetTargetfileName (configuration As Object,
pathType As Long, withExtensions As Long)

Arguments

configuration

The configuration for which the file name is requested.

The external code generator queries the configuration for its RPConfiguration interface.

pathType

The requested path format. The possible values are as follows:

1—Include the full path. For example: C:\Project\Component\Config\Class1.h

2—Include only the name of the file. For example: Class1.h

3—Include the path relative from the project directory. For example: Component\Config\Class1.h

4—Include the path relative from the active configuration to the requested file.

For example, if the file is located under C:\Project\Component\Subfolder\Class1.h, the external code generator will include the following path: Subfolder\Class1.h.
withExtensions
   Specifies whether to include the extension in the returned file name. For example,
   target.exe instead of target.

Return Value
   The name of the target file

C/C++ Prototype
   BSTR GetTargetfileName (IDispatch* configuration,
   int pathType, long withExtensions)

WhoAmI

Description
   The WhoAmI event is invoked to identify the external code generator.

Visual Basic

Syntax
   Event WhoAmI()

C/C++ Prototype
   BSTR WhoAmI()

Return Value
   A string that identifies the name and version number of the external code generator. It is
   printed to the output window before the generate event is invoked.
IRPExternalCodeGeneratorInvoker Interface

The IRPExternalCodeGeneratorInvoker is the interface that invokes the external code generator. The invoker is the object that fires all the events defined by the IRPExternalCodeGenerator interface. The external code generator registers the invoker instance to get events, and notifies the IRPExternalCodeGeneratorInvoker when a code generation session is over.

This interface inherits from IDispatch.

Method Summary

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>notifyGenerationDone</td>
<td>Is called by the external code generator after a generation session invoked by the generate event is done</td>
</tr>
</tbody>
</table>
**notifyGenerationDone**

**Description**

The **notifyGenerationDone** method is called by the external code generator after a generation session invoked by the **generate** event is done. You cannot invoke a new code generation session or make any changes to the model between the call to the **generate** and **notifyGenerationDone** events. However, you can set the timeout period using the property `<lang>_CG::Configuration::ExternalGenerationTimeout`. See the definition provided for the property on the applicable Properties tab of the Features dialog box.

**Note:** The external code generator *must* call this method after a code generation session (invoked by the **generate** event) was done or aborted (by the **Abort** event).

**Visual Basic**

**Syntax**

```vbs
notifyGenerationDone()
```

**C/C++ Prototype**

```c
HRESULT notifyGenerationDone()
```

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
IRPFile Interface

The IRPFile interface represents a file or folder to be generated during code generation. It inherits from IRPModelElement.

**VB Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elements</td>
<td>Collection of RPClassifiers</td>
<td>RO</td>
<td>The elements to be mapped to the file or folder.</td>
</tr>
<tr>
<td>files</td>
<td>Collection of RPFiles</td>
<td>RO</td>
<td>If fileType is “folder,” files is the collection of all files contained in that folder.</td>
</tr>
<tr>
<td>fileType</td>
<td>String</td>
<td>RW</td>
<td>The file type (“folder,” “implementation,” “specification,” “logical,” or “other”).</td>
</tr>
<tr>
<td>path(fullPath)</td>
<td>String(path) Boolean(fullPath)</td>
<td>RO</td>
<td>The string containing the path to the component. If fullPath is true, the full path is returned: &lt;drive&gt;:&lt;model dir&gt;&lt;component dir&gt;&lt;config dir&gt; If fullPath is false, the path relative to the project is returned: &lt;component dir&gt;&lt;config dir&gt;</td>
</tr>
</tbody>
</table>
## Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>addElement</code></td>
<td>Adds an element to the current file</td>
</tr>
<tr>
<td><code>addPackageToScope</code></td>
<td>Adds the specified package to the scope of the file or folder</td>
</tr>
<tr>
<td><code>addTextElement</code></td>
<td>Adds text to the file</td>
</tr>
<tr>
<td><code>addToScope</code></td>
<td>Places an element within the scope of the current file or folder</td>
</tr>
<tr>
<td><code>getImpName</code></td>
<td>Retrieves the name of the current file's implementation file, including its extension and, if specified, its relative path</td>
</tr>
<tr>
<td><code>getSpecName</code></td>
<td>Retrieves the name of the current file’s specification file, including its extension and, if specified, its relative path</td>
</tr>
<tr>
<td><code>isEmpty</code></td>
<td>Determines whether the current file is empty</td>
</tr>
<tr>
<td><code>setPath</code></td>
<td>Sets the path to the specified file</td>
</tr>
</tbody>
</table>
addElement

Write method

Description

The `addElement` method adds an element to the current file or folder.

Visual Basic

Syntax

```visualbasic
addElement (element As RPClassifier, fileFragmentType As String)
```

Arguments

- `element`
  An `RPClassifier` that specifies the new element to be mapped to the current file. The possible values are as follows:
  - Actors
  - Classes
  - Data
  - Use cases
- `fileFragmentType`
  One of the following strings:
  - `undefFragment`—The element is not defined.
  - `textFragment`—The element is text.
  - `implFragment`—The implementation of the element is added to the file.
  - `specFragment`—The specification of the element is added to the file.
  - `moduleFragment`—Both implementation and specification of the element are added to the file.

C/C++ Prototype

```c
HRESULT addElement (IRPClassifier *element, BSTR fileFragmentType)
```

Return Value

- `HRESULT` (0 for success, or a signed integer error code)
addPackageToScope

Write method

Description

The addPackageToScope method adds the specified package to the scope of the file or folder.

Visual Basic

Syntax

addPackageToScope (p As RPPackage)

Arguments

p
The package to add

C/C++ Prototype

HRESULT addPackageToScope (IRPPackage *p)

Return Value

HRESULT (0 for success, or a signed integer error code)
addTextElement

Write method

Description

The addTextElement method adds text to the file.

Visual Basic

Syntax

addTextElement (text As String)

Arguments

text

The text to add to the file

C/C++ Prototype

HRESULT addTextElement (BSTR text)

Return Value

HRESULT (0 for success, or a signed integer error code)
**addToScope**

*Write method*

**Description**

The `addToScope` method places an element within the scope of the current file or folder. If the file represents a file, both the implementation and specification of the element are added to the file. If the file represents a folder, the element is added to the folder scope.

**Visual Basic**

**Syntax**

```vba
addToScope (element As RPClassifier)
```

**Arguments**

- `element`:
  The element to place in the scope of the file

**C/C++ Prototype**

```c
HRESULT addToScope (IRPClassifier *element)
```

**Return Value**

- `HRESULT (0 for success, or a signed integer error code)`
**getImpName**

**Read method**

**Description**

The `getImpName` method retrieves the name of the current file’s implementation file, including its extension and, if specified, its relative path.

**Visual Basic**

**Syntax**

```vbnet
GetImpName (includingPath As Long) As String
```

**Arguments**

- `includingPath` (1 or 0)
  
  Set this to 1 to include the relative path in the implementation file name.

**Return Value**

The name of the implementation file

**C/C++ Prototype**

```c
HRESULT getImpName (long includingPath, BSTR* pVal)
```

**Return Value**

- `HRESULT (0 for success, or a signed integer error code)`
getSpecName
   Read method

Description
   The getSpecName method retrieves the name of the current file’s specification file, including its extension and, if specified, its relative path.

Visual Basic

Syntax
   getSpecName (includingPath As Long) As String

Arguments
   includingPath(1 or 0)
   Set this to 1 to include the relative path in the specification file name.

Return Value
   The name of the specification file

C/C++ Prototype

HRESULT getSpecName (long includingPath, BSTR* pVal)

Return Value
   HRESULT (0 for success, or a signed integer error code)
**isEmpty**

**Read method**

**Description**

The `isEmpty` method determines whether the current file is empty.

**Visual Basic**

**Syntax**

`IsEmpty() As Long`

**Return Value**

1 if the file is empty; otherwise 0

**C/C++ Prototype**

`HRESULT isEmpty (long* pVal)`

**Return Value**

`HRESULT (0 for success, or a signed integer error code)"`
setPath

Write method

Description
The setPath method sets the path to the specified file.

Visual Basic
Syntax
setPath (path As String)

Arguments
path
The file path

C/C++ Prototype
HRESULT setPath (BSTR path)

Return Value
HRESULT (0 for success, or a signed integer error code)
IRPFlow Interface

The IRPFlow interface represents a flow. It inherits from IRPModelElement.

VB Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>conveyed</td>
<td>RPCollection</td>
<td>RO</td>
<td>A read-only collection of information elements conveyed by the flow.</td>
</tr>
<tr>
<td>direction</td>
<td>String</td>
<td>RW</td>
<td>A string specifying the direction of the flow. The possible values are:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• toEnd1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• toEnd2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• bidirectional</td>
</tr>
<tr>
<td>end1</td>
<td>RPModelElement</td>
<td>RW</td>
<td>An association to a model object that is one of the ends of the flow.</td>
</tr>
<tr>
<td>end1Port</td>
<td>RPPort</td>
<td>RO</td>
<td>Valid when end1 is an RPInstance that is connected via a port defined by a port</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>defined by the class of the instance.</td>
</tr>
<tr>
<td>end2</td>
<td>RPModelElement</td>
<td>RW</td>
<td>An association to a model object that is one of the ends of the flow.</td>
</tr>
<tr>
<td>end2Port</td>
<td>RPPort</td>
<td>RO</td>
<td>Valid when end2 is an RPInstance that is connected via a port defined by the instance</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addConveyed</td>
<td>Adds an information element to the conveyed collection</td>
</tr>
<tr>
<td>removeConveyed</td>
<td>Removes an information element to the conveyed collection</td>
</tr>
<tr>
<td>setEnd1ViaPort</td>
<td>Connects end1 of the flow to the specified instance via the given port (defined by the instance class)</td>
</tr>
<tr>
<td>setEnd2ViaPort</td>
<td>Connects end2 of the flow to the specified instance via the given port (defined by the instance class)</td>
</tr>
</tbody>
</table>
addConveyed

Write method

Description

The `addConveyed` method adds an information element to the `conveyed` collection.

Visual Basic

Syntax

`addConveyed (pElement As RPModelElement)`

Arguments

- `pElement`
  The information element to add

C/C++ Prototype

`HRESULT addConveyed (IRPModelElement* pElement)`

Return Value

`HRESULT` (0 for success, or a signed integer error code)
removeConveyed

Write method

Description

The `removeConveyed` method removes an information element from the `conveyed` collection.

Visual Basic

Syntax

```visualbasic
removeConveyed (pElement As RPModelElement)
```

Arguments

- `pElement`
  
  The information element to remove

C/C++ Prototype

```
HRESULT removeConveyed (IRPModelElement* pElement)
```

Return Value

- `HRESULT (0 for success, or a signed integer error code)`
**setEnd1ViaPort**

*Write method*

**Description**

The `setEnd1ViaPort` method connects end1 of the flow to the specified instance via the given port (defined by the instance class).

**Visual Basic**

**Syntax**

```vbnet
setEnd1ViaPort (pInstance As RPInstance, pPort As RPPort)
```

**Arguments**

- `pInstance`
  
  The instance to which to connect end1 of the flow

- `pPort`
  
  The port used to connect end1 of the flow to `pInstance`

**C/C++ Prototype**

```c
HRESULT setEnd1ViaPort (IRPInstance* pInstance,
                       IRPModelElement* pPort)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
setEnd2ViaPort

Write method

Description

The `setEnd2ViaPort` method connects `end2` of the flow to the specified instance via the given port (defined by the instance class).

Visual Basic

Syntax

```vbnet
setEnd2ViaPort (pInstance As RPInstance, pPort As RPPort)
```

Arguments

- `pInstance`
  The instance to which to connect `end2` of the flow
- `pPort`
  The port used to connect `end2` of the flow to `pInstance`

C/C++ Prototype

```c
HRESULT setEnd2ViaPort (IRPInstance* pInstance,
                        IRPModelElement* pPort)
```

Return Value

- `HRESULT (0 for success, or a signed integer error code)`
The \texttt{IRPFlowchart} interface represents an activity diagram (formerly referred to as a flowchart). It inherits from \texttt{IRPStatechart}.

### VB Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isAnalysisOnly</td>
<td>Long</td>
<td>RW</td>
<td>If this is set to 1 (as opposed to 0), this \texttt{IRPFlowchart} is for analysis only.</td>
</tr>
<tr>
<td>itsOwner</td>
<td>RPOperation</td>
<td>RW</td>
<td>The operation that owns this activity diagram</td>
</tr>
<tr>
<td>swimlanes</td>
<td>RPCollection</td>
<td>RO</td>
<td>The collection of swimlanes in the activity diagram</td>
</tr>
</tbody>
</table>

### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{addReferenceActivity}</td>
<td>Adds a reference activity to the activity diagram</td>
</tr>
<tr>
<td>\texttt{addSwimlane}</td>
<td>Adds a swimlane to the activity diagram</td>
</tr>
</tbody>
</table>
addReferenceActivity

Note
Currently, this method has not been implemented.

Write method

Description
The addReferenceActivity method adds the specified reference activity to the activity diagram.

Visual Basic

Syntax
addReferenceActivity (referenced As RPModelElement)
As RPState

Arguments
referenced
The referenced activity or activity chart

Return Value
The new reference activity

C/C++ Prototype
HRESULT addReferenceActivity (
    IRPModelElement* referenced, IRPState** pVal)

Return Value
HRESULT (0 for success, or a signed integer error code)
addSwimlane

**Note**
Currently, this method has not been implemented.

**Write method**

**Description**
The `addSwimlane` method adds the specified swimlane to the activity diagram.

**Visual Basic**

**Syntax**
```
addSwimlane (name As String) As RPSwimlane
```

**Arguments**
```
name
The name for the new swimlane
```

**Return Value**
The new RPSwimlane

**C/C++ Prototype**
```
HRESULT addSwimlane (BSTR name, IRPSwimlane** pVal)
```

**Return Value**
```
HRESULT (0 for success, or a signed integer error code)
```
IRPFlowItem Interface

The IRPFlowItem interface represents a flowItem. It inherits from IRPClassifier. IRPFlowItem is a limited classifier (it cannot own attributes, operations, types, and so on), but the interface does support generalization.

VB Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>represented</td>
<td>Collection of RPFlowItems</td>
<td>RO</td>
<td>A read-only collection of flow items represented by the flowItem</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addRepresented</td>
<td>Adds a flowItem to the represented collection</td>
</tr>
<tr>
<td>removeRepresented</td>
<td>Removes a flowItem from the represented collection</td>
</tr>
</tbody>
</table>
**addRepresented**

**Write method**

**Description**

The `addRepresented` method adds a flowItem to the `represented` collection.

**Visual Basic**

**Syntax**

`addRepresented (pElement As RPModelElement)`

**Arguments**

`pElement`

The flow item to add

**C/C++ Prototype**

`HRESULT addRepresented (IRPModelElement* pElement)`

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
**removeRepresented**

**Write method**

**Description**

The `removeRepresented` method removes a flowItem from the `represented` collection.

**Visual Basic**

**Syntax**

```vbnet
removeRepresented (pElement As RPModelElement)
```

**Arguments**

- `pElement`
  
  The flow item to remove

**C/C++ Prototype**

```c
HRESULT removeRepresented (IRPModelElement* pElement)
```

**Return Value**

- `HRESULT (0 for success, or a signed integer error code)}`
IRPGeneralization Interface

The IRPGeneralization interface represents an inheritance relation between two classifiers (class/use case/actor). It inherits from IRPModelElement.

VB Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseClass</td>
<td>RPClassifier</td>
<td>RW</td>
<td>The base class of the generalization</td>
</tr>
<tr>
<td>derivedClass</td>
<td>RPClassifier</td>
<td>RW</td>
<td>The derived class of the generalization</td>
</tr>
<tr>
<td>extensionPoint</td>
<td>String</td>
<td>RW</td>
<td>The extension point</td>
</tr>
<tr>
<td>isVirtual</td>
<td>Long</td>
<td>RO</td>
<td>A flag that indicates whether the generalization is virtual</td>
</tr>
<tr>
<td>visibility</td>
<td>String</td>
<td>RO</td>
<td>The visibility of the generalization (public, protected, or private)</td>
</tr>
</tbody>
</table>

The baseClass and derivedClass properties allow write access to update the generalization. For example, if class C is derived from class A and you want to derive it from class B instead, follow these steps:

```vbnet
C.getGeneralization.baseClass = B
```

Here, getGeneralization is used as pseudo-operation shorthand for the procedure involved in actually obtaining a Generalization object from a class.

Similarly, if class C is derived from A and you want to derive it from B instead, follow these steps:

```vbnet
B.getGeneralization.derivedClass = C
```
IRPGraphEdge Interface

The IRPGraphEdge interface represents a linear element of a diagram, such as a transition. It represents the UML GraphEdge class. IRPGraphEdge inherits from IRPGraphElement.

**VB Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source</td>
<td>RPGraphNode</td>
<td>RO</td>
<td>The point at which the edge is connected to the source</td>
</tr>
<tr>
<td>target</td>
<td>RPGraphNode</td>
<td>RO</td>
<td>The point at which the edge is connected to the target</td>
</tr>
</tbody>
</table>

IRPGraphElement Interface

The IRPGraphElement interface is the base for all graphical elements on a diagram. It represents the UML Interchange GraphElement class. IRPGraphElement inherits from IRPDispatch.

**VB Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>graphicalParent</td>
<td>RPGraphElement</td>
<td>RO</td>
<td>The owning object</td>
</tr>
<tr>
<td>modelObject</td>
<td>RPModelElement</td>
<td>RO</td>
<td>The graphical object</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getAllGraphicalProperties</td>
<td>Returns the list of graphical properties for a diagram element</td>
</tr>
<tr>
<td>getGraphicalProperty</td>
<td>Returns the specified graphical property for a diagram element</td>
</tr>
<tr>
<td>setGraphicalProperty</td>
<td>Allows the setting of graphical properties for a diagram element</td>
</tr>
</tbody>
</table>
**getAllGraphicalProperties**

Read method

*Description*

The `getAllGraphicalProperties` method returns the list of graphical properties for a diagram element.

*Visual Basic*

*Syntax*

```vbnet
getAllGraphicalProperties() As RPCollection
```

*Return Value*

An `RPCollection` that contains the read-only list of graphical properties

*C/C++ Prototype*

```c
HRESULT getAllGraphicalProperties (IRPCollection** pVal)
```

*Return Value*

`HRESULT` (0 for success, or a signed integer error code)
getGraphicalProperty

Read method

Description

The getGraphicalProperty method returns the value of the specified graphical property for a diagram element.

Visual Basic

Syntax

getGraphicalProperty(name As String)
As RPGraphicalProperty

Arguments

name

The name of the property whose value you want to retrieve (note that only the actual property name is required here, there is no need to specify the hierarchy, as is the case with getPropertyValue).

Return Value

The value of the specified property, or null if the specified key is unsupported or invalid

C/C++ Prototype

HRESULT getGraphicalProperty (BSTR name,
IRPGraphicalProperty **pVal)

Return Value

HRESULT (0 for success, or a signed integer error code)
**setGraphicalProperty**

Write method

**Description**

The `setGraphicalProperty` method allows the setting of graphical properties for a diagram element.

**Visual Basic**

**Syntax**

```visualbasic
setGraphicalProperty(name As String, value As String)
```

**Arguments**

- `name`
  The name of the graphical property whose value you want to set (note that only the actual property name is required here; there is no need to specify the hierarchy, as is the case with `setPropertyValue`).

- `value`
  The value of the specified graphical property

**C/C++ Prototype**

```c
HRESULT setGraphicalProperty([in] BSTR name, [in] BSTR value)
```

**Return Value**

- `HRESULT` (0 for success, or a signed integer error code)

**VBA Sample**

```vba
Dim proj As RPProject
Set d = proj.addNewAggr("ObjectModelDiagram", "MyDiagram")
Dim m As RPModelElement
Dim n1 As RPGraphNode
Dim n2 As RPGraphNode
Dim e As RPGraphEdge
Dim c1 As RPClass
Dim c2 As RPClass
Dim gp As RPGraphicalProperty

On Error GoTo aa
Set proj = getProject
Set d = proj.addNewAggr("ObjectModelDiagram", "MyDiagram")
Set m = proj.findNestedElementRecursive("C", "Class")
Set c1 = m
' Add node for existing element
```
Set n1 = d.AddNewNodeForElement(m, 10, 20, 50, 50)
Call n1.setGraphicalProperty("LineColor", "155.230.100")

' Add node with new element
Set n2 = d.AddNewNodeByType("Class", 110, 120, 50, 50)
Set c2 = n2.modelObject
c2.name = "D"

' Add edge for new dependency
Set e = d.AddNewEdgeByType("Dependency", n1, 60, 60, n2, 130, 140)
Set gp = e.getGraphicalProperty("LineStyle")
MsgBox gp.value
Exit Sub

aa:
MsgBox errorMessage
IRPGraphicalProperty Interface

The IRPGraphicalProperty interface represents a graphical elements on a diagram. It inherits from IRPDispatch.

**VB Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>String</td>
<td>RO</td>
<td>The name of the property</td>
</tr>
<tr>
<td>value</td>
<td>String</td>
<td>RO</td>
<td>The property value</td>
</tr>
</tbody>
</table>

IRPGraphNode Interface

The IRPGraphNode interface represents either a boxed element (for example, a class box) or a point element (for example, a connector) in a diagram. It represents the UML GraphNode class. IRPGraphNode inherits from IRPGraphElement.

**VB Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>String</td>
<td>RW</td>
<td>The body of the guard</td>
</tr>
</tbody>
</table>

IRPGuard Interface

The IRPGuard interface represents the guard of a transition in a statechart diagram. It inherits from IRPModelElement.

**VB Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>String</td>
<td>RW</td>
<td>The body of the guard</td>
</tr>
</tbody>
</table>
**IRPHyperLink Interface**

The IRPHyperLink interface enables you to read the attributes of hyperlink objects.

**Note:** You cannot create or modify hyperlinks using the COM API.

**VB Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>RPModelElement</td>
<td>RW</td>
<td>The target for the hyperlink</td>
</tr>
<tr>
<td>URL</td>
<td>String</td>
<td>RW</td>
<td>The URL for the hyperlink</td>
</tr>
</tbody>
</table>

**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getDisplayOption</td>
<td>Returns the display option (free text, target name, target label, or tag value) for the hyperlink</td>
</tr>
<tr>
<td>setDisplayOption</td>
<td>Sets the display option (free text, target name, target label, or tag value) for the hyperlink</td>
</tr>
</tbody>
</table>
getDisplayOption

Read method

Description

The `getDisplayOption` method returns the display option (free text, target name, target label, or tag value) for the hyperlink.

Visual Basic

Syntax

```vbnet
getDisplayOption (pVal As HYPNameType, [pDisplayName As String])
```

Arguments

- **pVal**
  - The hyperlink

Return Value

A string that represents the display option for the hyperlink

C/C++ Prototype

```c
HRESULT getDisplayOption (HYPNameType* pVal, BSTR *pDisplayName)
```

Return Value

`HRESULT` (0 for success, or a signed integer error code)
**setDisplayOption**

Write method

**Description**

The `setDisplayOption` method sets the display option (free text, target name, target label, or tag value) for the hyperlink.

**Visual Basic**

**Syntax**

```vbnet
setDisplayOption (pVal As HYPNameType, [pDisplayName AsString])
```

**Arguments**

- `pVal`
  - The hyperlink
- `pDisplayName`
  - The display type (free text, target name, target label, or tag value)

**C/C++ Prototype**

```c
HRESULT setDisplayOption (HYPNameType* pVal, BSTR *pDisplayName)
```

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
IRPI mageMap

Represents diagram element information that can be used to build an HTML image map for the diagram. IRPDiagram's `getPictureAs` method returns a collection of objects of this type.

**interfaceName**
This property is for future use.

**isGUID**
Indicates whether the target property is the GUID of the element.

**name**
Name of the element.

**pictureFileName**
Name of the image file.

**points**
String that represents the bounding rectangle for the element in the Rhapsody diagram (for example, "10,10,206,10,206,151,10,151").

**shape**
This property is for future use.

**target**
Target for the image map entry.
IRPInstance Interface

The IRPInstance interface represents an instance. It is derived from IRPRelation, because the instance is a relation between an owner and some class.

VB Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>instantiatedBy</td>
<td>RPOperation</td>
<td>RW</td>
<td>The constructor used to create the instance, as defined by the user within the instance features dialog box</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getInLinks</td>
<td>Retrieves the list of incoming links</td>
</tr>
<tr>
<td>getListOfInitializerArguments</td>
<td>Retrieves the list of initializer arguments</td>
</tr>
<tr>
<td>getOutLinks</td>
<td>Retrieves the list of outgoing links</td>
</tr>
<tr>
<td>setInitializerArgumentValue</td>
<td>Sets the value of the initializer argument</td>
</tr>
</tbody>
</table>
**getInLinks**

**Read method**

**Description**

The `getInLinks` method returns the list of links for which the instance is the target instance (identified by the “to” property of the link).

**Visual Basic**

**Syntax**

```vbnet
getInLinks() As RPCollection
```

**Return Value**

An `RPCollection` that contains the read-only list of incoming links

**C/C++ Prototype**

```c
HRESULT getInLinks (IRPCollection** pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
**getListOfInitializerArguments**

**Read method**

**Description**

The `getListOfInitializerArguments` method returns the list of arguments for the initializer, as defined by the user in the instance features dialog box.

**Visual Basic**

**Syntax**

```
getListOfInitializerArguments() As RPCollection
```

**Return Value**

An `RPCollection` that contains the values of the arguments passed to the initializer. This list is a read-only list of strings.

**C/C++ Prototype**

```
HRESULT getListOfInitializerArguments(
    IRPCollection** pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
**getOutLinks**

**Read method**

**Description**

The `getOutLinks` method returns the list of links for which the instance is the source instance (identified by the “from” property of the link).

**Visual Basic**

**Syntax**

```
getOutLinks() As RPCollection
```

**Return Value**

An `RPCollection` that contains the read-only list of outgoing links

**C/C++ Prototype**

```
HRESULT getOutLinks (IRPCollection** pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
setInitializerArgumentValue

Write method

Description

The `setInitializerArgumentValue` method sets the value of the initializer argument.

Visual Basic

Syntax

```vbnet
setInitializerArgumentValue(argName As String,
argValue as String)
```

Arguments

- argName
  - The name of the initializer argument
- argValue
  - The initial value of the initializer argument

C/C++ Prototype

```c
HRESULT setInitializerArgumentValue (BSTR argName,
BSTR argValue)
```

Return Value

- HRESULT (0 for success, or a signed integer error code)
IRPInteractionOccurrence Interface

The IRPInteractionOccurrence interface represents an interaction occurrence (reference sequence diagram). It derives from IRPModelElement.

VB Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>messagePoints</td>
<td>RPCollection</td>
<td>RO</td>
<td>The message points of the referenced sequence diagram</td>
</tr>
<tr>
<td>referenceSequenceDiagram</td>
<td>RPSequenceDiagram</td>
<td>RW</td>
<td>The sequence diagram being referenced</td>
</tr>
</tbody>
</table>
**IRPInterfaceItem Interface**

The IRPInterfaceItem interface represents the commonality of class interface elements. It derives from IRPModelElement.

**VB Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>arguments</td>
<td>Collection of RPArguments</td>
<td>RO</td>
<td>The arguments of this operation or event</td>
</tr>
<tr>
<td>signature</td>
<td>String</td>
<td>RO</td>
<td>The signature of this operation. For example:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot;f(int x, char *y)&quot;</td>
</tr>
</tbody>
</table>

**Method Summary**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addArgument</td>
<td>Adds an argument for the operation to the end of its argument list</td>
</tr>
<tr>
<td>addArgumentBeforePosition</td>
<td>Adds an argument for the operation at the specified position in its argument</td>
</tr>
<tr>
<td>getSignatureNoArgNames</td>
<td>Retrieves the signature of the current class interface element without</td>
</tr>
<tr>
<td></td>
<td>argument names</td>
</tr>
<tr>
<td>getSignatureNoArgTypes</td>
<td>Retrieves the signature of the current class interface element without</td>
</tr>
<tr>
<td></td>
<td>argument types</td>
</tr>
<tr>
<td>matchOnSignature</td>
<td>Determines whether the signature of the current class interface element</td>
</tr>
<tr>
<td></td>
<td>matches that of another IRPInterfaceItem</td>
</tr>
</tbody>
</table>
addArgument

Write method

Description

The `addArgument` method adds an argument for the operation to the end of its argument list.

Visual Basic

Syntax

```
addArgument (newVal As String) As RPArgument
```

Arguments

- `NewVal`
  - The new argument to append to the argument list

Return Value

The new argument added to the argument list

C/C++ Prototype

```
HRESULT addArgument (BSTR newVal, IRPArgument** argument)
```

Return Value

```
HRESULT (0 for success, or a signed integer error code)
```
addArgumentBeforePosition

Write method

Description

The `addArgumentBeforePosition` method adds an argument for the operation at the specified position in its argument list.

Visual Basic

Syntax

```
addArgumentBeforePosition (newVal As String, pos As Long)
As RPArgument
```

Arguments

- `newVal`:
  The new argument to add to the argument list
- `pos`:
  A long that represents the position of the argument in argument list (1,2,3,...,n; left to right)

Return Value

The new argument added to the argument list

C/C++ Prototype

```
HRESULT addArgumentBeforePosition (BSTR newVal, long pos,
IRPArgument** argument)
```

Return Value

`HRESULT` (0 for success, or a signed integer error code)
**getSignatureNoArgNames**

**Read method**

**Description**

The `getSignatureNoArgNames` method retrieves the signature of the current class interface element without argument names.

**Visual Basic**

**Syntax**

```vbnet
getSignatureNoArgNames() As String
```

**Return Value**

The signature of the element without argument names. For example:

```plaintext
f(string, int)
```

**C/C++ Prototype**

```c
HRESULT getSignatureNoArgNames (BSTR *pVal)
```

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
getSignatureNoArgTypes

Read method

Description

The `getSignatureNoArgTypes` method retrieves the signature of the current class interface element without argument types.

Visual Basic

Syntax

```vbnet
getSignatureNoArgTypes() As String
```

Return Value

The signature of the element without argument types. For example:

```
f(x, y)
```

C/C++ Prototype

```
HRESULT getSignatureNoArgTypes (BSTR *pVal)
```

Return Value

`HRESULT (0 for success, or a signed integer error code)`
**matchOnSignature**

**Read method**

**Description**

The `matchOnSignature` method determines whether the signature of the current class interface element matches that of another IRPInterfaceItem.

**Visual Basic**

**Syntax**

```vbnet
matchOnSignature (item As RPInterfaceItem) As Long
```

**Arguments**

- `item`
  A pointer to the RPInterfaceItem whose signature is being compared to that of the current interface item

**Return Value**

1 if the two signatures match; otherwise 0

**C/C++ Prototype**

```c
HRESULT matchOnSignature (IRPInterfaceItem* item, long *pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
IRPLink Interface

The IRPLink interface represents a link-end that instantiates a relation. It inherits from IRPModelElement.

**VB Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>end1Multiplicity</td>
<td>String</td>
<td>RW</td>
<td>The multiplicity of the first end of the link</td>
</tr>
<tr>
<td>end1Name</td>
<td>String</td>
<td>RW</td>
<td>The name of the first end of the link</td>
</tr>
<tr>
<td>end2Multiplicity</td>
<td>String</td>
<td>RW</td>
<td>The multiplicity of the second end of the link</td>
</tr>
<tr>
<td>end2Name</td>
<td>String</td>
<td>RW</td>
<td>The name of the second end of the link</td>
</tr>
<tr>
<td>from</td>
<td>RPInstance</td>
<td>RO</td>
<td>The source instance of the link.</td>
</tr>
<tr>
<td>instantiates</td>
<td>RPRelation</td>
<td>RO</td>
<td>The association the link instantiates.</td>
</tr>
<tr>
<td>other</td>
<td>RPLink</td>
<td>RO</td>
<td>The pair link. In most cases, this property is redundant.</td>
</tr>
<tr>
<td>to</td>
<td>RPInstance</td>
<td>RO</td>
<td>The target instance of the link.</td>
</tr>
</tbody>
</table>
The IRPMessage interface represents a message sent between two classifier roles in a collaboration. It inherits from IRPModelElement.

**VB Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actualParameterList</td>
<td>String</td>
<td>RO</td>
<td>A collection of strings that contain parameters.</td>
</tr>
<tr>
<td>communication</td>
<td>RPAssociationRole</td>
<td>RO</td>
<td>The communication connection. This is always NULL for sequence diagrams.</td>
</tr>
<tr>
<td>condition</td>
<td>Cstring</td>
<td>RO</td>
<td>This is meaningful only if the message is of type “condition”.</td>
</tr>
<tr>
<td>formalInterfaceItem</td>
<td>RPInterfaceItem</td>
<td>RO</td>
<td>This can be NULL for timeouts or “default” for CTOR, DTOR, and non-specified methods.</td>
</tr>
<tr>
<td>messageType</td>
<td>Cstring</td>
<td>RO</td>
<td>The message type (constructor, destructor, event, operation, triggered, timeout, cancelled timeout, condition, or unspecified).</td>
</tr>
<tr>
<td>returnValue</td>
<td>Cstring</td>
<td>RO</td>
<td>The name of the element that receives the return value.</td>
</tr>
<tr>
<td>sequenceNumber</td>
<td>Cstring</td>
<td>RO</td>
<td>The number or position in an ordered list. For sequence diagrams, Rhapsody deduces the number.</td>
</tr>
<tr>
<td>source</td>
<td>RPClассifierRole</td>
<td>RO</td>
<td>Specifies who sent the message.</td>
</tr>
<tr>
<td>target</td>
<td>RPClассifierRole</td>
<td>RO</td>
<td>Specifies who received the message.</td>
</tr>
<tr>
<td>timerValue</td>
<td>String</td>
<td>RO</td>
<td>The timer value</td>
</tr>
</tbody>
</table>

**Method Summary**

getSignature

Retrieves the prototype of the IRPMessage
getSignature

Read method

Description

The getSignature method retrieves the prototype of the IRPMessage.

Visual Basic

Syntax

gSignature () As String

Return Value

The signature

C/C++ Prototype

HRESULT getSignature (BSTR* pVal)

Return Value

HRESULT (0 for success, or a signed integer error code)
IRPMessagePoint Interface

The IRPMessagePoint interface represents an event in a sequence diagram. It inherits from IRPModelElement.

Note that in a collaboration diagram, all events are send/receive pairs with nothing in between them.

VB Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>RPMessage</td>
<td>RO</td>
<td>The message that the current event refers to</td>
</tr>
<tr>
<td>type</td>
<td>String</td>
<td>RO</td>
<td>“Send” or “receive”</td>
</tr>
</tbody>
</table>

Method Summary

**getClassifierRole** Retrieves the classifier role for this message point
**getClassifierRole**

*Read method*

**Description**

The `getClassifierRole` method retrieves the classifier role for this message point. This is the classifier role (object) that received this event and sent back a return message.

**Visual Basic**

**Syntax**

```vba
getClassifierRole() As RPClassifierRole
```

**Return Value**

The `RPClassifierRole` on which the message occurred

**C/C++ Prototype**

```c
HRESULT getClassifierRole (IRPClassifierRole** classifierRole)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
The IRPModelElement interface is the base abstract interface for all Rhapsody 6.1 metamodel elements. It consists of all the common functionality shared by all the elements in the model (except for the Application class). It acts as an abstract interface.

### VB Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>annotations</td>
<td>Collection of RPAnnotations</td>
<td>RO</td>
<td>The annotations that belong to this model element.</td>
</tr>
<tr>
<td>associationClasses</td>
<td>Collection of IRPAssociationClasses</td>
<td>RO</td>
<td>The association classes connected to this model element.</td>
</tr>
<tr>
<td>constraints</td>
<td>Collection of RPConstraints</td>
<td>RO</td>
<td>The constraints that belong to this model element.</td>
</tr>
<tr>
<td>constraintsByHim</td>
<td>Collection of RPConstraints</td>
<td>RO</td>
<td>The constraints that affect this model element.</td>
</tr>
<tr>
<td>dependencies</td>
<td>Collection of RPDependency</td>
<td>RO</td>
<td>The model elements on which this model element depends.</td>
</tr>
<tr>
<td>description</td>
<td>String</td>
<td>RW</td>
<td>The description of this model element.</td>
</tr>
<tr>
<td>descriptionHTML</td>
<td>String</td>
<td>RW</td>
<td>The description of the model element in HTML format.</td>
</tr>
<tr>
<td>descriptionRTF</td>
<td>String</td>
<td>RW</td>
<td>The description of the model element in RTF format.</td>
</tr>
<tr>
<td>displayName</td>
<td>String</td>
<td>RW</td>
<td>The display name.</td>
</tr>
<tr>
<td>GUID</td>
<td>String</td>
<td>RW</td>
<td>The GUID value.</td>
</tr>
<tr>
<td>hyperLinks</td>
<td>Collection of IRPHyperLink-s</td>
<td>RO</td>
<td>The hyperlinks added to an element.</td>
</tr>
<tr>
<td>isOfMetaclass (metaclass)</td>
<td>Long</td>
<td>RO</td>
<td>This is equal to 1 (as opposed to 0) if the current model element is a member of this metaclass. Requires the string metaclass.</td>
</tr>
<tr>
<td>isShowDisplayName</td>
<td>Long</td>
<td>RW</td>
<td>Specifies whether to show the display name.</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Access</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------</td>
<td>--------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>mainDiagram</td>
<td>RPDiagram</td>
<td>RW</td>
<td>The main diagram of this element. Currently, this property is valid only for classes, packages, actors, and use cases.</td>
</tr>
<tr>
<td>metaClass</td>
<td>String</td>
<td>RO</td>
<td>The metaclass of this model element.</td>
</tr>
<tr>
<td>name</td>
<td>String</td>
<td>RW</td>
<td>The name of this model element.</td>
</tr>
<tr>
<td>ofTemplate</td>
<td>RPModelElement</td>
<td>RW</td>
<td>If the model element is an instantiation, this method will return the template used to instantiate it.</td>
</tr>
<tr>
<td>owner</td>
<td>RPModelElement</td>
<td>RW</td>
<td>The object in which this model element is defined. You can use this property to establish ownership. For example, suppose c is a class and p is a package:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Dim c as RPClass</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Dim p as RPPackage</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>set c = ...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>set p = ...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>c.owner = p</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This will work for any two objects where one can contain the other.</td>
</tr>
<tr>
<td>project</td>
<td>RPProject</td>
<td>RO</td>
<td>The project that owns this element.</td>
</tr>
<tr>
<td>requirementTraceabilityHandle</td>
<td>long</td>
<td>RW</td>
<td>The handle to this model element used by requirement traceability tools.</td>
</tr>
<tr>
<td>stereotype</td>
<td>RPSTereotype</td>
<td>RW</td>
<td>The stereotype attached to this model element.</td>
</tr>
<tr>
<td>templateParameters</td>
<td>Collection of RPTemplateParameter</td>
<td>RO</td>
<td>If this model element is a template, the method returns the template's parameters.</td>
</tr>
</tbody>
</table>
### Method Summary

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addDependency</td>
<td>RPTemplateInstantiation</td>
<td>RW</td>
<td>Adds a dependency relationship to the specified object</td>
</tr>
<tr>
<td>addDependencyTo</td>
<td></td>
<td></td>
<td>Creates a new dependency between two objects</td>
</tr>
<tr>
<td>addNewAggr</td>
<td></td>
<td></td>
<td>Used to add a new model element to the current element, for example, adding a class to a package</td>
</tr>
<tr>
<td>addProperty</td>
<td></td>
<td></td>
<td>Adds a new property/value pair for the current element</td>
</tr>
<tr>
<td>addStereotype</td>
<td></td>
<td></td>
<td>Adds a stereotype relationship to the specified object</td>
</tr>
<tr>
<td>becomeTemplateInstantiationOf</td>
<td></td>
<td></td>
<td>Creates a template instantiation of another template (of another template class)</td>
</tr>
<tr>
<td>clone</td>
<td></td>
<td></td>
<td>Clones the element</td>
</tr>
<tr>
<td>deleteDependency</td>
<td></td>
<td></td>
<td>Deletes a dependency</td>
</tr>
<tr>
<td>deleteFromProject</td>
<td></td>
<td></td>
<td>Deletes the current model element from the project open in Rhapsody 6.1</td>
</tr>
<tr>
<td>errorMessage</td>
<td></td>
<td></td>
<td>Returns the most recent error message</td>
</tr>
<tr>
<td>findElementsByFullName</td>
<td></td>
<td></td>
<td>Searches for the specified element</td>
</tr>
<tr>
<td>findNestedElement</td>
<td></td>
<td></td>
<td>Retrieves the specified element nested in a model element</td>
</tr>
<tr>
<td>findNestedElementRecursive</td>
<td></td>
<td></td>
<td>Retrieves the specified element from a given model element at any level of nesting within that element</td>
</tr>
<tr>
<td>getErrorMessage</td>
<td></td>
<td></td>
<td>Returns the most recent error message</td>
</tr>
</tbody>
</table>
### Rhapsody Interfaces

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getFullPathName</code></td>
<td>Retrieves the full path name of a model element as a string</td>
</tr>
<tr>
<td><code>getFullPathNameIn</code></td>
<td>Retrieves the full path name of a model element as a string</td>
</tr>
<tr>
<td><code>getNestedElements</code></td>
<td>Retrieves the elements defined in the current object</td>
</tr>
<tr>
<td><code>getNestedElementsRecursive</code></td>
<td>Recursively retrieves the elements defined in the model element for the object and for objects defined in it</td>
</tr>
<tr>
<td><code>getOverriddenProperties</code></td>
<td>Retrieves the list of properties whose default values have been overridden</td>
</tr>
<tr>
<td><code>getPropertyValue</code></td>
<td>Returns the value associated with the specified key value</td>
</tr>
<tr>
<td><code>getPropertyValueExplicit</code></td>
<td>Returns an explicit value if it has been assigned to the metamodel</td>
</tr>
<tr>
<td><code>getTag</code></td>
<td>Returns the tag for the specified model element</td>
</tr>
<tr>
<td><code>HighLightElement</code></td>
<td>Highlights the current model element</td>
</tr>
<tr>
<td><code>openFeaturesDialog</code></td>
<td>Displays the information for an element in the Features dialog. Depending on parameter provided, opens new dialog or uses an already-open dialog.</td>
</tr>
<tr>
<td><code>removeProperty</code></td>
<td>Removes the property from the model element</td>
</tr>
<tr>
<td><code>removeStereotype</code></td>
<td>Removes the stereotype</td>
</tr>
<tr>
<td><code>setPropertyValue</code></td>
<td>Modifies the value of the specified property</td>
</tr>
<tr>
<td><code>setTagValue</code></td>
<td>Assigns the specified tag to the model element</td>
</tr>
<tr>
<td><code>synchronizeTemplateInstantiation</code></td>
<td>Is used to synchronize between a template and a template instantiation parameter</td>
</tr>
</tbody>
</table>
addDependency

Write method

Description

The `addDependency` method adds a dependency relationship to the specified object.

Visual Basic

Syntax

```
addDependency (dependsOnName As String, dependsOnType As String) As RPDependency
```

Arguments

- `dependsOnName`: The name of the object that this element depends on
- `dependsOnType`: The type of object that this element depends on

Return Value

The newly created dependency

C/C++ Prototype

```
HRESULT addDependency (BSTR dependsOnName, BSTR dependsOnType, IRPDependency** pVal)
```

Return Value

`HRESULT (0 for success, or a signed integer error code)`
addDependencyTo

Write method

Description

The addDependencyTo method creates a new dependency relationship between two objects.

Visual Basic

Syntax

addDependencyTo (element As RPModelElement) As RPDependency

Arguments

    element
    The name of the object that the current object depends on

Return Value

The newly created dependency

C/C++ Prototype

HRESULT addDependencyTo (IRPModelElement* element, IRPDependency** pVal)

Return Value

HRESULT (0 for success, or a signed integer error code)
addNewAggr

Write method

Description

The `addNewAggr` method is used to add a new model element to the current element, for example, adding a new class to a package or adding a new diagram to a project.

Visual Basic

Syntax

```visualbasic
addNewAggr (metaType As String, name As String) As RPModelElement
```

Arguments

- metaType
  - The type of element to add (the string to use is the name of the appropriate metaclass).
  - **Note**
    - The list of metaclass names that can be used for this argument can be found in the file `metaclasses.txt` in the `Doc` directory of your Rhapsody installation.

- name
  - The name to use for the new element

Return Value

The newly created element

C/C++ Prototype

```csharp
HRESULT addNewAggr (BSTR metaType, BSTR name, IRPModelElement** newObject)
```

Return Value

`HRESULT` (0 for success, or a signed integer error code)

Example

```plaintext
Set proj = getProject
Set d = proj.addNewAggr(“ObjectModelDiagram”, “MyDiagram”)
```
addProperty

Write method

Description

The addProperty method adds a new property/value pair for the current element. This method is capable of flagging an error. For more information, see Error Handling.

Visual Basic

Syntax

addProperty (propertyKey As String, propertyType As String, propertyValue As String)

Arguments

propertyKey
The name of the new property.

propertyType
The property type. The possible values are as follows:

Int
String
Enum, <xxx>, <yyy>, <zzz> (i.e., Enum, followed by each of the defined values, for example: Enum,No,Prefix,Suffix)

Bool

propertyValue
The default value of the new property.

C/C++ Prototype

HRESULT addProperty (BSTR propertyKey, BSTR propertyType, BSTR propertyValue)

Return Value

HRESULT (0 for success, or a signed integer error code)
add Stereotype

Write method

Description

The `addStereotype` method adds a stereotype relationship to the specified object.

Visual Basic

Syntax

```vbnet
addStereotype (name As String, metaType As String) As RPStereotype
```

Arguments

- `name`: The name of the object in the new stereotype relationship
- `metaType`: The type of the object in the new stereotype relationship

Return Value

The newly created stereotype relationship

C/C++ Prototype

```c
HRESULT addStereotype (BSTR name, BSTR metaType, 
IRPSTereotype** stereotype)
```

Return Value

`HRESULT` (0 for success, or a signed integer error code)

Example

```vbscript
Sub addNetwork(c As RPClass)
Dim o As RPOperation
    c.addOperation ("serialize")
    c.addOperation ("unserialize")
    c.addConstructor ("")
On Error Resume Next
    c.addDestructor ("")
    x = c.addStereotype("G3Network", "Class")
End Sub
```
**becomeTemplateInstantiationOf**

**Write method**

**Description**

The `becomeTemplateInstantiationOf` method creates a template instantiation of another template (of another template class).

**Visual Basic**

**Syntax**

```vbnet
becomeTemplateInstantiationOf (newVal As RPModelElement)
```

**Arguments**

`newVal`

The template object that the template is an instantiation of

**C/C++ Prototype**

```c
HRESULT becomeTemplateInstantiationOf (IRPModelElement *newVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
clone

Write method

Description

The clone method clones the element, names it, and adds it to the new owner.

Visual Basic

Syntax

clone (name As String, newOwner As RPModelElement) As RPModelElement

Arguments

name
The name to use for the cloned element
newOwner
The new owner of the cloned element

C/C++ Prototype

HRESULT clone (BSTR string, IRPModelElement *newOwner,
               IRPModelElement** pVal)

Return Value

HRESULT (0 for success, or a signed integer error code)
deleteDependency

Write method

Description

The `deleteDependency` method deletes a dependency.

Visual Basic

Syntax

deleDepe (dependency As RPDependency)

Arguments

dependency
The dependency to delete

C/C++ Prototype

HRESULT deleteDependency (IRPDependency* dependency)

Return Value

HRESULT (0 for success, or a signed integer error code)
**deleteFromProject**

**Write method**

**Description**

The `deleteFromProject` method deletes the current model element from the project open in Rhapsody 6.1.

**Visual Basic**

**Syntax**

```vbnet
deleteFromProject()
```

**C/C++ Prototype**

```cpp
HRESULT deleteFromProject()
```

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
errorMessage

Read method

Description
The `errorMessage` method returns the most recent error message.

Visual Basic

Syntax
```vbnet
errorMessage() As String
```

Return Value
The most recent error message (a string)

C/C++ Prototype

```c
HRESULT errorMessage (BSTR* __MIDL_0020)
```

Return Value
`HRESULT` (0 for success, or a signed integer error code)
findElementsByFullName

Read method

Description

The `findElementsByFullName` method searches for the specified element.

Visual Basic

Syntax

```vba
findElementsByFullName (name As String, metaClass As String) As RPModelElement
```

Arguments

- `name`
  - The name of the element to look for
- `metaClass`
  - The element’s metaclass

Return Value

The specified element

C/C++ Prototype

```c
HRESULT findElementsByFullName (BSTR name, BSTR metaClass, IRPModelElement** pVal)
```

Return Value

`HRESULT` (0 for success, or a signed integer error code)

Example

Suppose you have a class A, under package P. The following VBA code will find this class using the `findElementsByFullName` API call:

```vba
Dim proj As RPProject
Dim m As RPModelElement
Set proj = getProject
Set m = proj.findElementsByFullName("A in P", "Class")
MsgBox m.name
```

Note

This method requires that you use the “full” notation, e.g., “A in P”. Otherwise, the method will not return the specified element.
findNestedElement

Read method

Description

The `findNestedElement` method retrieves the specified element nested in a model element.

For example, if `x` is of type `IRPModelElement` (or a type inherited from it), the following call returns an attribute of `x` named `A` (or null if there is no such element):

```
x.findNestedElement('A','Attribute')
```

Visual Basic

Syntax

```
findNestedElement (name As String, metaClass As String)
As RPModelElement
```

Arguments

- `name`: The name of the element
- `metaClass`: The name of the metaclass

Return Value

- If found, the retrieved `RPModelElement`; otherwise, NULL

C/C++ Prototype

```
HRESULT findNestedElement (BSTR name, BSTR metaClass, IRPModelElement** pVal)
```

Return Value

- `HRESULT` (0 for success, or a signed integer error code)

Example

```
Sub addUi(c As RPClass)
Dim x As Object
Dim p As RPPackage
Dim theClass As RPClass
' all gui objects are derived from GUI.UIBase
c.Description = "gui class"
On Error Resume Next
Set p = pr.findNestedElement("GUI", "Package")
Set theClass = p.findNestedElement("UIBase", "Class")
c.addGeneralization theClass
If Not Err.Number = 0 Then
```

MsgBox (errorMessage)
End If

c.addStereotype "G3UI", "Class"
End Sub

**findNestedElementRecursive**

**Read method**

**Description**

The **findNestedElementRecursive** method retrieves the specified element from a given model element at any level of nesting within that element.

For example, if `x` is of type `IRPModelElement` (or a type inherited from it), the following call returns an attribute named `A` (or null if there is no such element) of `x`, or of any element nested within `x` at any level of ownership:

```
x.findNestedElementRecursive('A','Attribute')
```

**Visual Basic**

**Syntax**

```
IRPModelElement findNestedElementRecursive(
   name As String, metaClass As String) As RPModelElement
```

**Arguments**

- **name**
  - The name of the element
- **metaClass**
  - The name of the metaclass

**Return Value**

If found, the retrieved `RPModelElement`; otherwise, NULL

**C/C++ Prototype**

```
HRESULT findNestedElementRecursive (BSTR name,
   BSTR metaClass, IRPModelElement** pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
getErrorMessage
   Read method

Description
   The getErrorMessage method returns the most recent error message.

Visual Basic

Syntax
   String getErrorMessage (__MIDL_0019 As String)

Return Value
   The most recent error message

C/C++ Prototype

   HRESULT getErrorMessage (BSTR* __MIDL_0019)

Return Value
   HRESULT (0 for success, or a signed integer error code)
**getFullPathName**

**Read method**

**Description**

The `getFullPathName` method retrieves the full path name of a model element as a string with the following format:

```plaintext
<package>::<class>
```

**Visual Basic**

**Syntax**

```vbnet
getFullPathName() As String
```

**Return Value**

The full path of the model element

**C/C++ Prototype**

```c
HRESULT getFullPathName (BSTR* name)
```

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`

**Example**

The following macro checks each transition to see if it has a trigger.

```vbnet
Sub checkNullTransitions()
    Dim elem As RPModelElement
    For Each elem In getProject.getNestedElementsRecursive
        If elem.metaClass = "Transition" Then
            Dim trans As RPTransition
            Set trans = elem
            If trans.getItsTrigger Is Nothing Then
                Debug.Print "The trigger in transition " + 
                trans.getFullPathName + "' is null!"
            End If
        End If
    Next elem
End Sub
...
**getFullPathNameIn**

*Read method*

**Description**

The `getFullPathNameIn` method retrieves the full path name of a model element as a string in the following format:

```<class> in <package>```

**Visual Basic**

**Syntax**

```getFullPathNameIn() As String```

**Return Value**

The full path of the model element

**C/C++ Prototype**

```HRESULT getFullPathNameIn (BSTR* name)```

**Return Value**

```HRESULT (0 for success, or a signed integer error code)```
**getNestedElements**

Read method

**Description**

The `getNestedElements` method retrieves the elements defined in the current object.

**Visual Basic**

**Syntax**

```vb
getNestedElements() As RPCollection
```

**Return Value**

A collection of model elements defined in the current object

**C/C++ Prototype**

```c
HRESULT getNestedElements (IRPCollection** __MIDL_0017)
```

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
**getNestedElementsRecursive**

**Write method**

**Description**

The `getNestedElementsRecursive` method recursively retrieves the elements defined in the model element for the object and for objects defined in it.

**Visual Basic**

**Syntax**

```visualbasic
getNestedElementsRecursive() As RPCollection
```

**Return Value**

A collection of model elements defined in the current object and the objects nested within it

**C/C++ Prototype**

```c
HRESULT getNestedElementsRecursive(
    IRPCollection** __MISL__0018)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)

**Example**

The following macro checks each transition to see if it has a trigger.

```visualbasic
Sub checkNullTransitions()
    Dim elem As RPModelElement
    For Each elem In getProject.getNestedElementsRecursive
        If elem.metaClass = "Transition" Then
            Dim trans As RPTransition
            Set trans = elem
            If trans.getItsTrigger Is Nothing Then
                Debug.Print "The trigger in transition " + trans.getFullPathName + " is null!"
            End If
        End If
    Next elem
End Sub
```

...
**getOverriddenProperties**

**Read method**

**Description**

The `getOverriddenProperties` method retrieves the list of properties whose default values have been overridden.

**Visual Basic**

**Syntax**

```vbnet
getOverriddenProperties (recursive As Long) As RPCollection
```

**Arguments**

- `recursive`
  Specifies whether to include the properties of ascendants of the unit

**C/C++ Prototype**

```c
HRESULT getOverriddenProperties (long recursive,
                                IRPCollection **pVal)
```

**Return Value**

- `HRESULT (0 for success, or a signed integer error code)`
**getPropertyValue**

*Read method*

**Description**

The `getPropertyValue` method returns the value associated with the specified key value. This method is capable of flagging an error.

**Visual Basic**

**Syntax**

```vbnet
getPropertyValue (propertyKey As String) As String
```

**Arguments**

- `propertyKey` - The name of the property whose value is to be retrieved

**Return Value**

The value of a property explicitly assigned to this instance or the default value (the value propagated from the containers of the instance as a default).

**Notes**

Property-related API calls can cause the following error conditions:

- **RP_BAD_PROPERTY_KEY_ERROR**—Illegal property key syntax (that is, not in a "<subject>.<metaclass>.<name>" format).
- **RP_MISSING_PROPERTY_ERROR**—The property requested does not exist.
- **RP_PROPERTY_EXISTS_ERROR**—You are attempting to add a property that already exists.

**C/C++ Prototype**

```c
HRESULT getPropertyValue (BSTR propertyKey, 
BSTR* propertyValue)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
Example

```java
Set elt = getSelectedElement
theFileName = elt.getPropertyValue("UserProperties.ExternalFile.FileName")
theFilePath = elt.getPropertyValue("UserProperties.ExternalFile.FilePath")
theFileType = elt.getPropertyValue("UserProperties.ExternalFile.FileType")
```

**getPropertyValueExplicit**

**Read method**

**Description**

The `getPropertyValueExplicit` method is similar to the `getPropertyValue` method, but it does not return a default value. Instead, it returns an explicit value if it has been assigned to the metamodel.

This method is capable of flagging an error. For more information, see Error Handling

**Visual Basic**

**Syntax**

```visualbasic
getPropertyValueExplicit (propertyKey As String) As String
```

**Arguments**

- `propertyKey`:
  The name of the property whose value is to be retrieved

**Return Value**

The explicit value of the property, if one has been assigned to the metamodel instance

**C/C++ Prototype**

```c
HRESULT getPropertyValueExplicit (BSTR propertyKey, BSTR* propertyValue)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
**getTag**

**Read method**

**Description**

The `getTag` method returns the tag for the specified model element.

**Visual Basic**

**Syntax**

```
getTag (name As String) As RPTag
```

**Arguments**

- `name` The name of the element whose tag you want to retrieve

**Return Value**

The tag

**C/C++ Prototype**

```
HRESULT getTag (BSTR name, IRPTag **pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
openFeaturesDialog

Description

The method `openFeaturesDialog` displays the information for an element in the Features dialog. Depending on the parameter you provide, a new Features dialog will be opened or an already-open Features dialog will be used to display the information:

- 1 - opens a new dialog
- 0 - displays information in already-open dialog; opens a new dialog if there is not a Features dialog currently open.

Syntax

`openFeaturesDialog(newDialog As Long)`

Example

The code below displays the information for class C in a new Features dialog. P is the name of the package that contains the class.

```vbnet
Dim proj As RPProject
Dim m As RPModelElement
Set proj = getProject
Set m = proj.findElementsByFullName("C in P", "Class")
m.openFeaturesDialog(1)
```
HighLightElement

Read method

Description

The HighLightElement method highlights the current element.

Visual Basic

Syntax

HighLightElement ()

C/C++ Prototype

HRESULT highLightElement ()

Return Value

HRESULT (0 for success, or a signed integer error code)
removeProperty

Write method

Description

The removeProperty method removes the property from the model element.

This method is capable of flagging an error.

Visual Basic

Syntax

removeProperty (propertyKey As String)

Arguments

propertyKey

The name of the property to be removed

C/C++ Prototype

HRESULT removeProperty (BSTR propertyKey)

Return Value

HRESULT (0 for success, or a signed integer error code)
removeStereotype

Write method

Description

The removeStereotype method removes the stereotype from the model element.

Visual Basic

Syntax

removeStereotype (stereotype As RPSterotype)

Arguments

stereotype

The name of the stereotype to be removed

C/C++ Prototype

HRESULT removeStereotype (IRPStereotype* stereotype)

Return Value

HRESULT (0 for success, or a signed integer error code)
**setPropertyValue**

**Write method**

**Description**

The `setPropertyValue` method modifies the value of the specified property.

This method is capable of flagging an error. For more information, see Error Handling.

**Visual Basic**

**Syntax**

```vbnet
setPropertyValue (propertyKey As String,
                   propertyValue As String)
```

**Arguments**

- `propertyKey`
  
  The name of the property whose value is to be set

  **Note:** When providing the name of the property, the delimiter used should be a period, not a colon, for example, `CPP_CG.Attribute.AccessorGenerate`.

- `propertyValue`
  
  The value to be assigned to the property

**C/C++ Prototype**

```c
HRESULT setPropertyValue (BSTR propertyKey, BSTR propertyValue)
```

**Return Value**

- `HRESULT (0 for success, or a signed integer error code)`

**Sample Code**

```csharp
element.setPropertyValue "CPP_CG.Attribute.AccessorGenerate", "True"
```
**setTagValue**

**Write method**

**Description**

The `setTagValue` method assigns the specified tag to the current model element.

**Visual Basic**

**Syntax**

```
setTagValue (tag As RPTag, val As String) AS RPTag
```

**Arguments**

- `tag`
  - The name of the tag to add to the element
- `val`
  - The value of the new tag

**Return Value**

The new tag

**C/C++ Prototype**

```
HRESULT setTagValue (IRPTag *tag, BSTR val, IRPTag **pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
**synchronizeTemplateInstantiation**

**Write method**

**Description**

The `synchronizeTemplateInstantiation` method is used to synchronize between a template and a template instantiation parameter. For example, if you add a parameter to a template, this method updates the template instantiation. It is activated on template instantiation.

**Visual Basic**

**Syntax**

```
synchronizeTemplateInstantiation ()
```

**C/C++ Prototype**

```
HRESULT synchronizeTemplateInstantiation ()
```

**Return Value**

```
HRESULT (0 for success, or a signed integer error code)
```
Rhapsody Interfaces

**IRPModule Interface**

The IRPModule interface represents a Rhapsody module. It inherits from IRPInstance.

**IRPNode Interface**

The IRPNode interface represents a node. It derives from IRPClassifier.

**VB Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>componentInstances</td>
<td>RPCollection</td>
<td>RO</td>
<td>The list of component instances</td>
</tr>
<tr>
<td>CPUType</td>
<td>String</td>
<td>RW</td>
<td>The CPU type</td>
</tr>
</tbody>
</table>

**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>addComponentInstance</code></td>
<td>Adds a new component instance</td>
</tr>
<tr>
<td><code>deleteComponentInstance</code></td>
<td>Deletes the specified component instance</td>
</tr>
<tr>
<td><code>findComponentInstance</code></td>
<td>Retrieves the specified component instance</td>
</tr>
</tbody>
</table>

**addComponentInstance**

Write method

**Description**

The `addComponentInstance` method adds a component instance.

**Visual Basic**

**Syntax**

```vbnet
addComponentInstance (name As String) As RPComponentInstance
```

**Arguments**

- `name`
  The name of the new component instance

**Return Value**

The new component instance
C/C++ Prototype

HRESULT addComponentInstance (BSTR name, IRPComponentInstance** pVal)

Return Value

HRESULT (0 for success, or a signed integer error code)

deleteComponentInstance

Write method

Description

The deleteComponentInstance method deletes the specified component instance.

Visual Basic

Syntax

deleteComponentInstance (BSTR name)

Arguments

name

The name of the new component instance

C/C++ Prototype

HRESULT deleteComponentInstance (BSTR name)

Return Value

HRESULT (0 for success, or a signed integer error code)

findComponentInstance

Read method

Description

The findComponentInstance method retrieves the specified component instance.

Visual Basic

Syntax

findByComponentInstance (name As String)

As RFComponentInstance
Arguments

name

The name of the component instance to look for

Return Value

The component instance

C/C++ Prototype

HRESULT findComponentInstance (BSTR name,
IRPComponentInstance** pVal)

Return Value

HRESULT (0 for success, or a signed integer error code)
IRPOObjectModelDiagram Interface

The IRPOObjectModelDiagram interface represents an object model diagram. It inherits from IRPDiagram.

Currently, IRPOObjectModelDiagram does not expose additional functionality to IRPDiagram.

IRPOperation Interface

The IRPOperation interface is an abstract class that represents an operation. It derives from IRPInterfaceItem.

**VB Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>String</td>
<td>RW</td>
<td>The body of the operation.</td>
</tr>
<tr>
<td>flowchart</td>
<td>RPFlowchart</td>
<td>RW</td>
<td>The activity chart of the operation.</td>
</tr>
<tr>
<td>initializer</td>
<td>String</td>
<td>RW</td>
<td>If this operation is a constructor, this is a string containing the constructor initialization list.</td>
</tr>
<tr>
<td>isAbstract</td>
<td>Long</td>
<td>RW</td>
<td>This is equal to 1 (as opposed to 0) if the operation is abstract.</td>
</tr>
<tr>
<td>isCgDerived</td>
<td>Long</td>
<td>RO</td>
<td>This is equal to 1 (as opposed to 0) if this operation is automatically generated by Rhapsody 6.1.</td>
</tr>
<tr>
<td>isConst</td>
<td>Long</td>
<td>RO</td>
<td>This is equal to 1 (as opposed to 0) if the operation is a const.</td>
</tr>
<tr>
<td>isCtor</td>
<td>Long</td>
<td>RO</td>
<td>This is equal to 1 (as opposed to 0) if the operation is a constructor.</td>
</tr>
<tr>
<td>isdtor</td>
<td>Long</td>
<td>RO</td>
<td>This is equal to 1 (as opposed to 0) if the operation is a destructor.</td>
</tr>
<tr>
<td>isFinal</td>
<td>Long</td>
<td>RW</td>
<td>This is equal to 1 (as opposed to 0) if the operation is final (Java only).</td>
</tr>
</tbody>
</table>
### Method Summary

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isStatic</td>
<td>Long</td>
<td>RO</td>
<td>This is equal to 1 (as opposed to 0) if the operation is a static.</td>
</tr>
<tr>
<td>isTrigger</td>
<td>Long</td>
<td>RO</td>
<td>This is equal to 1 (as opposed to 0) if the operation is triggered.</td>
</tr>
<tr>
<td>isVirtual</td>
<td>Long</td>
<td>RO</td>
<td>This is equal to 1 (as opposed to 0) if the operation is virtual.</td>
</tr>
<tr>
<td>returns</td>
<td>RPClassifier</td>
<td>RW</td>
<td>The return type of this operation. In previous versions, this property was called &quot;returnType&quot;.</td>
</tr>
<tr>
<td>returnType</td>
<td>RPType</td>
<td>RW</td>
<td>The return type of this operation.</td>
</tr>
<tr>
<td>visibility</td>
<td>String</td>
<td>RW</td>
<td>The visibility of this operation (public, protected, or private).</td>
</tr>
</tbody>
</table>

**deleteArgument** Deletes an argument from the current operation

**deleteFlowchart** Deletes an activity diagram from the current operation

**getImplementationSignature** Returns a string representing the signature of the operation as it will appear in the generated code.

**setReturnTypeDeclaration** Specifies a new value for the return type declaration
**deleteArgument**

**Write method**

**Description**

The *deleteArgument* method deletes an argument from the current operation.

**Visual Basic**

**Syntax**

```vbnet
deleteArgument (argument As RPArgument)
```

**Arguments**

- `argument`
  
  The argument to be deleted

**C/C++ Prototype**

```c
HRESULT deleteArgument (IRPArgument* argument)
```

**Return Value**

- `HRESULT (0 for success, or a signed integer error code)`
**deleteFlowchart**

**Write method**

**Description**

The `deleteFlowchart` method deletes an activity diagram from the current operation.

**Visual Basic**

**Syntax**

```vbnet
deleteFlowchart()
```

**C/C++ Prototype**

```c
HRESULT deleteFlowchart()
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)

**getImplementationSignature**

Returns a string representing the signature of the operation as it will appear in the generated code.
**setReturnTypeDeclaration**

Write method

**Description**

The `setReturnTypeDeclaration` method specifies a new value for the return type declaration.

**Visual Basic**

**Syntax**

```vbnet
setReturnTypeDeclaration (newVal As String)
```

**Arguments**

- `newVal`
  The new value for the return type declaration

**C/C++ Prototype**

```c
HRESULT setReturnTypeDeclaration (BSTR newVal)
```

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
**IRPPackage Interface**

The IRPPackage interface represents Rhapsody packages, which are essentially definition spaces for diagrams and other model elements. It inherits from IRPUnit.

**VB Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>actors</td>
<td>Collection of RPActors</td>
<td>RO</td>
<td>The collection of actors defined in this package</td>
</tr>
<tr>
<td>classes</td>
<td>Collection of RPClasses</td>
<td>RO</td>
<td>The collection of classes defined in this package</td>
</tr>
<tr>
<td>collaboration Diagrams</td>
<td>Collection of RPCollaboration Diagrams</td>
<td>RO</td>
<td>The collection of collaboration diagrams defined in this package</td>
</tr>
<tr>
<td>componentDiagrams</td>
<td>Collection of RPComponent Diagrams</td>
<td>RO</td>
<td>The collection of component diagrams defined in this package</td>
</tr>
<tr>
<td>deploymentDiagrams</td>
<td>Collection of RPDeployment Diagrams</td>
<td>RO</td>
<td>The collection of deployment diagrams defined in the package</td>
</tr>
<tr>
<td>events</td>
<td>Collection of RPEvents</td>
<td>RO</td>
<td>The collection of events defined in this package</td>
</tr>
<tr>
<td>eventsBaseId</td>
<td>Long</td>
<td>RO</td>
<td>The event base identifier</td>
</tr>
<tr>
<td>globalFunctions</td>
<td>Collection of RPOperations</td>
<td>RO</td>
<td>The collection of global functions defined in the package</td>
</tr>
<tr>
<td>globalObjects</td>
<td>Collection of RPRelations</td>
<td>RO</td>
<td>The collection of global objects defined in the package</td>
</tr>
<tr>
<td>globalVariables</td>
<td>Collection of RPAttributes</td>
<td>RO</td>
<td>The collection of global variables defined in the package</td>
</tr>
<tr>
<td>flowItems</td>
<td>Collection of RPFlowItems</td>
<td>RO</td>
<td>The collection of information items defined in this package</td>
</tr>
<tr>
<td>flows</td>
<td>Collection of RPFloows</td>
<td>RO</td>
<td>The collection of flows defined in this package</td>
</tr>
<tr>
<td>nestedClassifiers</td>
<td>Collection of classifiers</td>
<td>RO</td>
<td>The collection of classifiers defined in this package</td>
</tr>
<tr>
<td>nodes</td>
<td>RPCollection</td>
<td>RO</td>
<td>The list of package nodes</td>
</tr>
<tr>
<td>objectModelDiagrams</td>
<td>Collection of RPObjectDiagrams</td>
<td>RO</td>
<td>The collection of object model diagrams defined in this package</td>
</tr>
</tbody>
</table>
Method Summary

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>packages</td>
<td>Collection of RPPackages</td>
<td>RO</td>
<td>The collection of packages nested inside this package</td>
</tr>
<tr>
<td>SavedInSeparateDirectory</td>
<td>Long</td>
<td>RW</td>
<td>Determines whether each package is saved in a separate directory</td>
</tr>
<tr>
<td>sequenceDiagrams</td>
<td>Collection of RPSequenceDiagrams</td>
<td>RO</td>
<td>The collection of sequence diagrams defined in this package</td>
</tr>
<tr>
<td>types</td>
<td>Collection of RPTypes</td>
<td>RO</td>
<td>The collection of data types defined in this package</td>
</tr>
<tr>
<td>useCaseDiagrams</td>
<td>Collection of RPUseCaseDiagrams</td>
<td>RO</td>
<td>The collection of use case diagrams defined in this package</td>
</tr>
<tr>
<td>useCases</td>
<td>Collection of RPUseCases</td>
<td>RO</td>
<td>The collection of use cases defined in this package</td>
</tr>
<tr>
<td>userDefinedStereotypes</td>
<td>Collection of RPStereotypes</td>
<td>RO</td>
<td>The collection of user-defined stereotypes defined in this package</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addActor</td>
<td>Adds the specified actor to the current package</td>
</tr>
<tr>
<td>addClass</td>
<td>Adds the specified class to the current package</td>
</tr>
<tr>
<td>addCollaborationDiagram</td>
<td>Adds the specified collaboration diagram to the current package</td>
</tr>
<tr>
<td>addComponentDiagram</td>
<td>Adds the specified component diagram to the current package</td>
</tr>
<tr>
<td>addDeploymentDiagram</td>
<td>Adds the specified deployment diagram to the current package</td>
</tr>
<tr>
<td>addEvent</td>
<td>Adds the specified event to the current package</td>
</tr>
<tr>
<td>addFlowItems</td>
<td>Adds the specified flowItem to the flowItems collection</td>
</tr>
<tr>
<td>addFlows</td>
<td>Adds the specified flow to the flows collection</td>
</tr>
<tr>
<td>addGlobalFunction</td>
<td>Adds the specified global function to this package</td>
</tr>
<tr>
<td>addGlobalObject</td>
<td>Adds a global object (instance) to the current package</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>addGlobalVariable</td>
<td>Adds the specified global variable to the current package</td>
</tr>
<tr>
<td>addLink</td>
<td>Adds a link between two objects to the current package</td>
</tr>
<tr>
<td>addNestedPackage</td>
<td>Adds a nested package to the current package</td>
</tr>
<tr>
<td>addNode</td>
<td>Adds the specified node to the current package</td>
</tr>
<tr>
<td>addObjectModelDiagram</td>
<td>Adds the specified OMD to the current package</td>
</tr>
<tr>
<td>addSequenceDiagram</td>
<td>Adds the specified sequence diagram to the current package</td>
</tr>
<tr>
<td>addType</td>
<td>Adds the specified type to the current package</td>
</tr>
<tr>
<td>addUseCase</td>
<td>Adds the specified use case to the current package</td>
</tr>
<tr>
<td>addUseCaseDiagram</td>
<td>Adds the specified UCD to the current package</td>
</tr>
<tr>
<td>deleteActor</td>
<td>Deletes the specified actor from the current package</td>
</tr>
<tr>
<td>deleteClass</td>
<td>Deletes the specified class from the current package</td>
</tr>
<tr>
<td>deleteCollaborationDiagram</td>
<td>Deletes the specified collaboration diagram from the current package</td>
</tr>
<tr>
<td>deleteComponentDiagram</td>
<td>Deletes the specified component diagram from the current package</td>
</tr>
<tr>
<td>deleteDeploymentDiagram</td>
<td>Deletes the specified deployment diagram from the current package</td>
</tr>
<tr>
<td>deleteEvent</td>
<td>Deletes the specified event from the current package</td>
</tr>
<tr>
<td>deleteFlowItems</td>
<td>Deletes the specified flowItem from the flowItems collection</td>
</tr>
<tr>
<td>deleteFlows</td>
<td>Deletes the specified flow from the flows collection</td>
</tr>
<tr>
<td>deleteGlobalFunction</td>
<td>Deletes the specified global function from the current package</td>
</tr>
<tr>
<td>deleteGlobalObject</td>
<td>Deletes the specified global object from the current package</td>
</tr>
<tr>
<td>deleteGlobalVariable</td>
<td>Deletes the specified global variable from the current package</td>
</tr>
<tr>
<td>deleteNode</td>
<td>Deletes the specified node from the current package</td>
</tr>
<tr>
<td>deleteObjectModelDiagram</td>
<td>Deletes the specified OMD from the current package</td>
</tr>
<tr>
<td>deletePackage</td>
<td>Deletes the current package</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>deleteSequenceDiagram</td>
<td>Deletes the specified sequence diagram from the current package</td>
</tr>
<tr>
<td>deleteType</td>
<td>Deletes the specified type from the current package</td>
</tr>
<tr>
<td>deleteUseCase</td>
<td>Deletes the specified use case from the current package</td>
</tr>
<tr>
<td>deleteUseCaseDiagram</td>
<td>Deletes the specified use case diagram from the current package</td>
</tr>
<tr>
<td>findActor</td>
<td>Retrieves the specified actor, if it belongs to the current package</td>
</tr>
<tr>
<td>findAllByName</td>
<td>Searches all the elements and finds the first element of the specified name and metaclass in the current package</td>
</tr>
<tr>
<td>findClass</td>
<td>Retrieves the specified class, if it belongs to the current package</td>
</tr>
<tr>
<td>findEvent</td>
<td>Retrieves the specified event, if it belongs to the current package</td>
</tr>
<tr>
<td>findGlobalFunction</td>
<td>Retrieves the specified global function, if it belongs to the current package</td>
</tr>
<tr>
<td>findGlobalObject</td>
<td>Retrieves the specified global object, if it belongs to the current package</td>
</tr>
<tr>
<td>findGlobalVariable</td>
<td>Retrieves the specified global variable, if it belongs to the current package</td>
</tr>
<tr>
<td>findNode</td>
<td>Retrieves the specified node, if it belongs to the current package</td>
</tr>
<tr>
<td>findType</td>
<td>Retrieves the specified data type, if it belongs to the current package</td>
</tr>
<tr>
<td>findUsage</td>
<td>Retrieves the usage of the specified element in the current package</td>
</tr>
<tr>
<td>findUseCase</td>
<td>Retrieves the specified use case, if it belongs to the current package</td>
</tr>
<tr>
<td>recalculateEventsBaseId</td>
<td>Recalculates the events base ID of the package</td>
</tr>
</tbody>
</table>
addActor

Write method

Description

The addActor method adds the specified actor to the current package.

Visual Basic

Syntax

addActor (name As String) As RPActor

Arguments

name

The name of actor to add to this package

Return Value

The new actor added to the package

C/C++ Prototype

HRESULT addActor (BSTR name, IRPActor** pVal)

Return Value

HRESULT (0 for success, or a signed integer error code)
**addClass**

Write method

**Description**

The **addClass** method adds the specified class to the current package.

**Visual Basic**

**Syntax**

```
addClass (name As String) As RPClass
```

**Arguments**

- `name`  
  
  The name of the class to be added

**Return Value**

The class added to this package

**C/C++ Prototype**

```
HRESULT addClass (BSTR name, IRPClass** pVal)
```

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
addCollaborationDiagram

Write method

Description

The `addCollaborationDiagram` method adds the specified collaboration diagram to the current package.

Visual Basic

Syntax

```vbnet
addCollaborationDiagram (name As String) As RPCollaborationDiagram
```

Arguments

- `name`

  The name of the collaboration diagram to be added

Return Value

The new collaboration diagram added to this package

C/C++ Prototype

```c
HRESULT addCollaborationDiagram (BSTR name, 
   IRPCollaborationDiagram** pVal)
```

Return Value

- `HRESULT` (0 for success, or a signed integer error code)
addComponentDiagram

Write method

Description

The `addComponentDiagram` method adds the specified component diagram to the current package.

Visual Basic

Syntax

```vbnet
addComponentDiagram (name As String) As RPComponentDiagram
```

Arguments

- **name**
  
  The name of the component diagram to be added

Return Value

The new component diagram added to this package

C/C++ Prototype

```c
HRESULT addComponentDiagram (BSTR name,
                           IRPComponentDiagram** pVal)
```

Return Value

`HRESULT` (0 for success, or a signed integer error code)
**addDeploymentDiagram**

**Write method**

**Description**

The `addDeploymentDiagram` method adds the specified deployment diagram to the current package.

**Visual Basic**

**Syntax**

```vbnet
addDeploymentDiagram (name As String) As RPDeploymentDiagram
```

**Arguments**

- `name`  
  The name of the deployment diagram to be added

**Return Value**

The new deployment diagram added to this package

**C/C++ Prototype**

```c
HRESULT addDeploymentDiagram (BSTR name, IRPDeploymentDiagram** pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
addEvent

Write method

Description

The addEvent method adds the specified event to the current package.

Visual Basic

Syntax

addEvent (name As String) As RPEvent

Arguments

name

The name of the event to be added

Return Value

The new event added to this package

C/C++ Prototype

HRESULT addEvent (BSTR name, IRPEvent** pVal)

Return Value

HRESULT (0 for success, or a signed integer error code)
**addFlowItems**

**Write method**

**Description**

The *addFlowItems* method adds the specified flowItem to the *flowItems* collection.

**Visual Basic**

**Syntax**

```
addFlowItems (name As String) As RPFlowItem
```

**Arguments**

- `name` The name of the flowItem to add to the collection

**Return Value**

The new flowItem added to this package

**C/C++ Prototype**

```
HRESULT addFlowItems (BSTR name,
                      IRPFlowItem** ppItem)
```

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
addFlows

Write method

Description

The addFlows method adds the specified flow to the flows collection.

Visual Basic

Syntax

addFlows (name As String) As RFlow

Arguments

name

The name of the flow to add to the collection

Return Value

The new flow added to this package

C/C++ Prototype

HRESULT addFlows (BSTR name, IRPFlow** ppFlow)

Return Value

HRESULT (0 for success, or a signed integer error code)
addGlobalFunction

Write method

Description

The *addGlobalFunction* method adds the specified global function to this package.

Visual Basic

Syntax

    addGlobalFunction (name As String) As RPOperation

Arguments

    name
    The global function to be added

Return Value

    The new global function added to this package

C/C++ Prototype

    HRESULT addGlobalFunction (BSTR name, 
    IRPOperation** pVal)

Return Value

    HRESULT (0 for success, or a signed integer error code)
addGlobalObject

Write method

Description

The addGlobalObject method adds a global object (instance) to the current package.

Visual Basic

Syntax

addGlobalObject (name As String,
otherClassName As String,
otherClassPackageName As String) As RPRelation

Arguments

name
The name of the global instance to add
otherClassName
The name of the class-defining instance
otherClassPackageName
The name of the package with the class-defining instance

Return Value

The new global instance in this package

C/C++ Prototype

HRESULT addGlobalObject (BSTR name, BSTR otherClassName,
BSTR otherClassPackageName, IRPRelation** pVal)

Return Value

HRESULT (0 for success, or a signed integer error code)
addGlobalVariable

Write method

Description

The `addGlobalVariable` method adds the specified global variable to the current package.

Visual Basic

Syntax

```
addGlobalVariable (name As String) As RPAttribute
```

Arguments

- `name`
  The name of the global variable to add

Return Value

The new global variable added to this package

C/C++ Prototype

```
HRESULT addGlobalVariable (BSTR name,
    IRPAttribute** pVal)
```

Return Value

`HRESULT (0 for success, or a signed integer error code)`
**addLink**

The `addLink` method adds a link between two objects to the current package.

**Syntax**

```
addLink(fromPart As RInstance, toPart As RInstance, assoc As RRelation, fromPort As RPort, toPort As RPort) As RLink
```

**Arguments**

- `fromPart`, `toPart`  
  The objects that are being linked.

- `assoc`  
  Association that is being instantiated (optional).

- `fromPort`, `toPort`  
  Ports that are being linked (optional).
addNestedPackage

Write method

Description

The `addNestedPackage` method adds a nested package to the current package.

Visual Basic

Syntax

```vbnet
addNestedPackage (name As String) As RPPackage
```

Arguments

- **name**
  
  The name of the nested package to add

Return Value

The nested package added to this package

C/C++ Prototype

```c
HRESULT addNestedPackage (BSTR name, IRPPackage** pVal)
```

Return Value

- `HRESULT` (0 for success, or a signed integer error code)
addNode

Write method

Description

The `addNode` method adds a node to the current package.

Visual Basic

Syntax

    addNode (name As String) As RPNode

Arguments

    name
    The name of the node to add

Return Value

The new node added to this package

C/C++ Prototype

    HRESULT addNode (BSTR name, IRPNode** pVal)

Return Value

    HRESULT (0 for success, or a signed integer error code)
addObjectModelDiagram

Write method

Description

The `addObjectModelDiagram` method adds the specified OMD to the current package.

Visual Basic

Syntax

```vbnet
addObjectModelDiagram (name As String) As RPObjectModelDiagram
```

Arguments

- `name` - The name of the OMD to add

Return Value

The OMD added to this package

C/C++ Prototype

```c
HRESULT addObjectModelDiagram (BSTR name, IRPObjectModelDiagram** pVal)
```

Return Value

`HRESULT (0 for success, or a signed integer error code)`
addSequenceDiagram

Write method

Description

The `addSequenceDiagram` method adds the specified sequence diagram to the current package.

Visual Basic

Syntax

```vbscript
addSequenceDiagram (name As String) As RPSequenceDiagram
```

Arguments

- `name`:
  The name of the sequence diagram to add

Return Value

The sequence diagram added to this package

C/C++ Prototype

```c
HRESULT addSequenceDiagram (BSTR name, IRPSequenceDiagram** pVal)
```

Return Value

```c
HRESULT (0 for success, or a signed integer error code)
```
addType

Write method

Description

The `addType` method adds the specified type to the current package.

Visual Basic

Syntax

```
addType (name As String) As RPType
```

Arguments

- `name`:
  The name of the type to add

Return Value

- The new type added to this package

C/C++ Prototype

```
HRESULT addType (BSTR name, IRPType** pVal)
```

Return Value

- `HRESULT (0 for success, or a signed integer error code)`
addUseCase

Write method

Description

The `addUseCase` method adds the specified use case to the current package.

Visual Basic

Syntax

```vbnet
addUseCase (name As String) As RPUseCase
```

Arguments

- `name`
  - The name of the use case to add

Return Value

The use case added to this package

C/C++ Prototype

```c
HRESULT addUseCase (BSTR name, IRPUseCase** pVal)
```

Return Value

HRRESULT (0 for success, or a signed integer error code)
addUseCaseDiagram

Write method

Description

The addUseCaseDiagram method adds the specified UCD to the current package.

Visual Basic

Syntax

addUseCaseDiagram (name As String) As RPUseCaseDiagram

Arguments

name
The name of the UCD to add

Return Value

The UCD added to this package

C/C++ Prototype

HRESULT addUseCaseDiagram (BSTR name,
IRPUseCaseDiagram** pVal)

Return Value

HRESULT (0 for success, or a signed integer error code)
deleteActor

Write method

Description

The deleteActor method deletes the specified actor from the current package.

Visual Basic

Syntax

deleateActor (actor As RPActor)

Arguments

actor

The actor to delete

C/C++ Prototype

HRESULT deleteActor (IRPActor *actor)

Return Value

HRESULT (0 for success, or a signed integer error code)
deleteClass

Write method

Description

The `deleteClass` method deletes the specified class from the current package.

Visual Basic

Syntax

deleceteClass (theClass As RPClass)

Arguments

  theClass
   The class to delete

C/C++ Prototype

   HRESULT deleteClass (IRPClass *theClass)

Return Value

   HRESULT (0 for success, or a signed integer error code)
**deleteCollaborationDiagram**

**Write method**

**Description**

The `deleteCollaborationDiagram` method deletes the specified collaboration diagram from the current package.

**Visual Basic**

**Syntax**

```visualbasic
deleteCollaborationDiagram (name As String)
```

**Arguments**

- `name`  
  The name of the collaboration diagram to delete

**C/C++ Prototype**

```c
HRESULT deleteCollaborationDiagram (BSTR name)
```

**Return Value**

- `HRESULT` (0 for success, or a signed integer error code)
**deleteComponentDiagram**

**Write method**

**Description**

The `deleteComponentDiagram` method deletes the specified component diagram from the current package.

**Visual Basic**

**Syntax**

```vbnet
deleteComponentDiagram (name As String)
```

**Arguments**

- `name`
  
The name of the component diagram to delete

**C/C++ Prototype**

```c
HRESULT deleteComponentDiagram (BSTR name)
```

**Return Value**

- `HRESULT (0 for success, or a signed integer error code)`
**deleteDeploymentDiagram**

**Write method**

**Description**

The **deleteDeploymentDiagram** method deletes the specified deployment diagram from the current package.

**Visual Basic**

**Syntax**

```vbnet
deleteDeploymentDiagram (name As String)
```

**Arguments**

- **name**
  
  The name of the deployment diagram to delete

**C/C++ Prototype**

```c
HRESULT deleteDeploymentDiagram (BSTR name)
```

**Return Value**

- **HRESULT** (0 for success, or a signed integer error code)
**deleteEvent**

**Write method**

**Description**

The `deleteEvent` method deletes the specified event from the current package.

**Visual Basic**

**Syntax**

`deleteEvent (event As RPEvent)`

**Arguments**

- `event`
  
  The event to delete

**C/C++ Prototype**

`HRESULT deleteEvent (IRPEvent *event)`

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
**deleteFlowItems**

**Write method**

**Description**

The `deleteFlowItems` method deletes the specified flowItem from the `flowItems` collection.

**Visual Basic**

**Syntax**

`deleteFlowItems (pItem As RPFlowItem)`

**Arguments**

`pItem`

The name of the flowItem to remove from the collection

**C/C++ Prototype**

`HRESULT deleteFlowItems (IRPFlowItem* pItem)`

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
**deleteFlows**

**Write method**

**Description**

The `deleteFlows` method deletes the specified flow from the `flows` collection.

**Visual Basic**

**Syntax**

```vbnet
deleteFlows (pFlow As RPFlow)
```

**Arguments**

- `pFlow`

  The name of the flow to delete from the collection

**C/C++ Prototype**

```c
HRESULT deleteFlows (IRPFlow* pFlow)
```

**Return Value**

- `HRESULT` (0 for success, or a signed integer error code)
**deleteGlobalFunction**

**Write method**

**Description**

The `deleteGlobalFunction` method deletes the specified global function from the current package.

**Visual Basic**

**Syntax**

```vbnet
deleteGlobalFunction (operation As RPOperation)
```

**Arguments**

- **operation**
  - The global function to delete

**C/C++ Prototype**

```c
HRESULT deleteGlobalFunction (IRPOperation* operation)
```

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
deleteGlobalObject

Write method

Description

The `deleteGlobalObject` method deletes the specified global object from the current package.

Visual Basic

Syntax

```vbnet
deleteGlobalObject (relation As RPRelation)
```

Arguments

- `relation`  
  The global object to delete

C/C++ Prototype

```c
HRESULT deleteGlobalObject (IRPRelation* relation)
```

Return Value

- `HRESULT (0 for success, or a signed integer error code)`
deleteGlobalVariable

Write method

Description

The `deleteGlobalVariable` method deletes the specified global variable from the current package.

Visual Basic

Syntax

```visual-basic
deleteGlobalVariable (attribute As RPAttribute)
```

Arguments

attribute

The global variable to delete

C/C++ Prototype

```c
HRESULT deleteGlobalVariable (IRPAttribute* attribute)
```

Return Value

HRESULT (0 for success, or a signed integer error code)
**deleteNode**

**Write method**

**Description**

The `deleteNode` method deletes the specified node from the current package.

**Visual Basic**

**Syntax**

```vbnet
deleteNode (name As String)
```

**Arguments**

- `name` - The name of the node to delete

**C/C++ Prototype**

```c
HRESULT deleteNode (BSTR name)
```

**Return Value**

- `HRESULT (0 for success, or a signed integer error code)`
**deleteObjectModelDiagram**

*Write method*

**Description**

The `deleteObjectModelDiagram` method deletes the specified OMD from the current package.

**Visual Basic**

**Syntax**

`deleteObjectModelDiagram (name As String)`

**Arguments**

- **name**
  
  The name of the OMD to delete

**C/C++ Prototype**

```c
HRESULT deleteObjectModelDiagram (BSTR name)
```

**Return Value**

- `HRESULT (0 for success, or a signed integer error code)`
deletePackage

Write method

Description

The deletePackage method deletes the current package.

Visual Basic

Syntax

deletePackage()

C/C++ Prototype

HRESULT deletePackage()

Return Value

HRESULT (0 for success, or a signed integer error code)
deleteSequenceDiagram

Write method

Description

The deleteSequenceDiagram method deletes the specified sequence diagram from the current package.

Visual Basic

Syntax

deleteSequenceDiagram (name As String)

Arguments

name

The name of the sequence diagram to delete

C/C++ Prototype

HRESULT deleteSequenceDiagram (BSTR name)

Return Value

HRESULT (0 for success, or a signed integer error code)
deleteType

Write method

Description

The deleteType method deletes the specified type from the current package.

Visual Basic

Syntax

deleateType (type As RType)

Arguments

  type
  The type to delete

C/C++ Prototype

HRESULT deleteType (IRPType *type)

Return Value

HRESULT (0 for success, or a signed integer error code)
deleteUseCase

Write method

Description

The `deleteUseCase` method deletes the specified use case from the current package.

Visual Basic

Syntax

`deleteUseCase (useCase As RPUseCase)`

Arguments

useCase

The use case to delete

C/C++ Prototype

`HRESULT deleteUseCase (IRPUseCase *useCase)`

Return Value

`HRESULT (0 for success, or a signed integer error code)`
**deleteUseCaseDiagram**

**Write method**

**Description**

The `deleteUseCaseDiagram` method deletes the specified use case diagram from the current package.

**Visual Basic**

**Syntax**

```vbnet
deleteUseCaseDiagram (name As String)
```

**Arguments**

- **name**
  
  The name of the UCD to delete

**C/C++ Prototype**

```c
HRESULT deleteUseCaseDiagram (BSTR name)
```

**Return Value**

- `HRESULT` (0 for success, or a signed integer error code)
findActor

Read method

Description

The findActor method retrieves the specified actor, if it belongs to the current package.

Visual Basic

Syntax

findActor (name As String) As RPActor

Arguments

name

The name of the actor to find

Return Value

If found, the RPActor; otherwise, NULL.

C/C++ Prototype

HRESULT findActor (BSTR name, IRPActor** pVal)

Return Value

HRESULT (0 for success, or a signed integer error code)
**findAllByName**

**Read method**

**Description**

The `findAllByName` method searches all the elements and finds the first element of the specified name and metaclass in the current package.

**Visual Basic**

**Syntax**

```vbnet
findAllByName (name As String, metaClass As String) As RPModelElement
```

**Arguments**

- **name**
  The name of the element to find
- **metaClass**
  The name of the metaclass to find

**Return Value**

The first `RPModelElement` that matches the specified name and metaclass, or NULL if not found

**C/C++ Prototype**

```c
HRESULT findAllByName (BSTR name, BSTR metaClass, IRPModelElement** pVal)
```

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
**findClass**

**Read method**

**Description**

The `findClass` method retrieves the specified class, if it belongs to the current package.

**Visual Basic**

**Syntax**

```vbnet
findClass (name As String) As RPClass
```

**Arguments**

- **name**
  
  The name of the class to find

**Return Value**

- The `RPClass`, or NULL if not found

**C/C++ Prototype**

```c
HRESULT findClass (BSTR name, IRPClass** pVal)
```

**Return Value**

- `HRESULT` (0 for success, or a signed integer error code)

---

**findEvent**

**Read method**

**Description**

The `findEvent` method retrieves the specified event, if it belongs to the current package.

**Visual Basic**

**Syntax**

```vbnet
findEvent (name As String) As RPEvent
```

**Arguments**

- **name**
  
  The name of the event to find
**Return Value**

The `RPEvent*`, or NULL if not found

**C/C++ Prototype**

```c
HRESULT findEvent (BSTR name, IRPEvent** pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
findGlobalFunction

Read method

Description

The `findGlobalFunction` method retrieves the specified global function, if it belongs to the current package.

Visual Basic

Syntax

`findGlobalFunction (name As String) As RPOperation`

Arguments

- `name`
  
The name of the global function to find

Return Value

- The `RPOperation`, or NULL if not found

C/C++ Prototype

`HRESULT findGlobalFunction (BSTR name, IRPOperation** pVal)`

Return Value

- `HRESULT` (0 for success, or a signed integer error code)
**findGlobalObject**

**Read method**

**Description**

The `findGlobalObject` method retrieves the specified global object, if it belongs to the current package.

**Visual Basic**

**Syntax**

```vba
findGlobalObject (name As String) As RPRelation
```

**Arguments**

- `name`
  
  The name of the global object to find

**Return Value**

- The `RPRelation`, or NULL if not found

**C/C++ Prototype**

```c
HRESULT findGlobalObject (BSTR name, IRPRelation** pVal)
```

**Return Value**

- `HRESULT (0 for success, or a signed integer error code)`
findGlobalVariable

Read method

Description

The `findGlobalVariable` method retrieves the specified global variable, if it belongs to the current package.

Visual Basic

Syntax

`findGlobalVariable (name As String) As RPAttribute`

Arguments

- **name**
  The name of the global variable to look for

Return Value

The `RPAttribute`, or NULL if not found

C/C++ Prototype

```c
HRESULT findGlobalVariable (BSTR name, 
                         IRPAttribute** pVal)
```

Return Value

`HRESULT` (0 for success, or a signed integer error code)
findNode

Read method

Description

The **findNode** method retrieves the specified node, if it belongs to the current package.

Visual Basic

Syntax

```vbnet
findNode (name As String) As RPNode
```

Arguments

- **name**
  
The name of the node to look for

Return Value

The **RPNode**, or NULL if not found

C/C++ Prototype

```c
HRESULT findNode (BSTR name, IRPNode** pVal)
```

Return Value

**HRESULT** (0 for success, or a signed integer error code)
**findType**

**Read method**

**Description**

The `findType` method retrieves the specified data type, if it belongs to the current package.

**Visual Basic**

**Syntax**

```vbnet
findType (name As String) As RType
```

**Arguments**

- `name`:
  The name of the type to find

**Return Value**

The `RType`, or NULL if not found

**C/C++ Prototype**

```c
HRESULT findType (BSTR name, IRPType** pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
findUsage

Read method

Description

The findUsage method retrieves the usage of the specified element in the current package.

Visual Basic

Syntax

findUsage (objToFind As IRPModelElement) As RPCollection

Arguments

objToFind

The model element to look for in the current package

Return Value

The collection of model elements that reference objToFind in this package

C/C++ Prototype

HRESULT findUsage (IRPModelElement* objToFind, IRPCollection** pVal)

Return Value

HRESULT (0 for success, or a signed integer error code)
**findUseCase**

*Read method*

**Description**

The `findUseCase` method retrieves the specified use case, if it belongs to the current package.

**Visual Basic**

**Syntax**

```visualbasic
findUseCase (name As String) As RPUseCase
```

**Arguments**

- `name`
  
  The name of the use case to find

**Return Value**

- The `RPUseCase`, or NULL if not found

**C/C++ Prototype**

```c
HRESULT findUseCase (BSTR name, IRPUseCase** pVal)
```

**Return Value**

- `HRESULT` (0 for success, or a signed integer error code)
**recalculateEventsBaseId**

Write method

**Description**

The `recalculateEventsBaseId` method recalculates the events base ID of the package.

**Visual Basic**

**Syntax**

```
recalculateEventsBaseId() As Long
```

**Return Value**

The events base ID

**C/C++ Prototype**

```
HRESULT recalculateEventsBaseId (long *success)
```

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
IRPPin Interface

The IRPPin interface represents action pins added to actions, or activity parameters added to action blocks, in an activity diagram. It inherits from IRPConnector.

To add an action pin to an action, use addConnector, for example:

```vbnet
action1.addConnector("InPin")
```

or

```vbnet
action1.addConnector("OutPin")
```

**VB Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isParameter</td>
<td>long</td>
<td>RW</td>
<td>Indicates whether the element is an action pin or an activity parameter. If this is equal to 1 (as opposed to 0), the element is an activity parameter.</td>
</tr>
<tr>
<td>pinDirection</td>
<td>String</td>
<td>RW</td>
<td>The possible values for this property are “In”, “Out”, “InOut”</td>
</tr>
<tr>
<td>pinType</td>
<td>RPClassifier</td>
<td>RW</td>
<td>Represents the pin’s argument type.</td>
</tr>
</tbody>
</table>

**Sample Code**

```vbnet
Sub action_pin_sample_IRPPin()

Dim currentProject As RPProject
Dim newPackage As RPPackage
Dim newClass As RPClass
Dim newActivityDiagram As RPFlowchart
Dim washingAction As RPState
Dim pinOnWashing As RPPin
Dim intType As RPModelElement

On Error GoTo errorHandlingCode

Set currentProject = getProject
```

API Reference Manual
Set newPackage = currentProject.addPackage("Package_One")
Set newClass = newPackage.addClass("Class_A")
Set newActivityDiagram = newClass.addActivityDiagram

' set to Analysis-only because action pins are only available on analysis-only diagrams
newActivityDiagram.isAnalysisOnly = 1

Set washingAction = newActivityDiagram.rootState.addState("Washing")

' create a pin whose direction is out
Set pinOnWashing = washingAction.addConnector("OutPin")

' set pin type to int
Set intType = currentProject.findNestedElementRecursive("int", "Type")
pinOnWashing.pinType = intType

Exit Sub
errorHandlingCode:
MsgBox errorMessage
End Sub
IRPPort Interface

The IRPPort interface represents a Rhapsody port. It inherits from IRPInstance.

VB Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>contract</td>
<td>RPClass</td>
<td>RW</td>
<td>Specifies the port contract.</td>
</tr>
<tr>
<td>isBehavioral</td>
<td>Long</td>
<td>RW</td>
<td>Determines whether messages sent to the port are relayed to the owner class.</td>
</tr>
<tr>
<td>isReversed</td>
<td>Long</td>
<td>RW</td>
<td>If this is equal to 1 (as opposed to 0), the provided interfaces become the required interfaces, and the required interfaces become the provided interfaces.</td>
</tr>
<tr>
<td>providedInterfaces</td>
<td>Collection of RPClasses</td>
<td>RO</td>
<td>The collection of provided interfaces for the port.</td>
</tr>
<tr>
<td>requiredInterfaces</td>
<td>Collection of RPClasses</td>
<td>RO</td>
<td>The collection of required interfaces for the port.</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addProvidedInterface</td>
<td>Adds the specified interface to the collection of provided interfaces</td>
</tr>
<tr>
<td>addRequiredInterface</td>
<td>Adds the specified interface to the collection of required interfaces</td>
</tr>
<tr>
<td>removeProvidedInterface</td>
<td>Removes the specified interface from the collection of provided interfaces</td>
</tr>
<tr>
<td>removeRequiredInterface</td>
<td>Removes the specified interface from the collection of required interfaces</td>
</tr>
</tbody>
</table>

Example

The following script converts a black-box analysis block to a white-box analysis block, and vice versa. It simply toggles all the ports of a block to behavioral or non-behavioral.
Public Sub ConvertPortsBB()
Dim curBlock As RPBlock
Dim port As RPPort
Set curBlock = getSelectedElement
For Each port In curBlock.ObjectAsObjectType.ports
    port.isBehavioral = 1
Next
End Sub

Public Sub ConvertPortsWB()
Dim curBlock As RPBlock
Dim port As RPPort
Set curBlock = getSelectedElement
For Each port In curBlock.ObjectAsObjectType.ports
    port.isBehavioral = 0
Next
End Sub
addProvidedInterface

Write method

Description

The `addProvidedInterface` method adds the specified interface to the collection of provided interfaces.

Visual Basic

Syntax

```visualbasic
addProvidedInterface (newVal As RPClass)
```

Arguments

- `newVal`
  
  The name of the class to add to the collection of provided interfaces for the port

C/C++ Prototype

```c
HRESULT addProvidedInterface (IRPClass* newVal)
```

Return Value

```c
HRESULT (0 for success, or a signed integer error code)
```
addRequiredInterface

Write method

Description

The addRequiredInterface method adds the specified interface to the collection of required interfaces.

Visual Basic

Syntax

addRequiredInterface (newVal As RPClass)

Arguments

newVal

The name of the class to add to the collection of required interfaces for the port

C/C++ Prototype

HRESULT addRequiredInterface (IRPClass* newVal)

Return Value

HRESULT (0 for success, or a signed integer error code)
**removeProvidedInterface**

**Write method**

**Description**

The `removeProvidedInterface` method removes the specified interface from the collection of provided interfaces.

**Visual Basic**

**Syntax**

```vbnet
removeProvidedInterface (newVal As RPClass)
```

**Arguments**

- **newVal**
  
  The name of the class to remove from the collection of provided interfaces for the port

**C/C++ Prototype**

```c
HRESULT removeProvidedInterface (IRPClass* newVal)
```

**Return Value**

- **HRESULT (0 for success, or a signed integer error code)**
removeRequiredInterface

Write method

Description

The removeRequiredInterface method removes the specified interface from the collection of required interfaces.

Visual Basic

Syntax

removeRequiredInterface (newVal As RPClass)

Arguments

newVal

The name of the class to remove from the collection of provided interfaces for the port

C/C++ Prototype

HRESULT removeRequiredInterface (IRPClass* newVal)

Return Value

HRESULT (0 for success, or a signed integer error code)
The `IRPProfile` interface represents a profile. It inherits from `IRPPackage`.

### IRPProject Interface

The `IRPProject` interface represents a Rhapsody project (model). Use the `Application.openProject()` method to obtain a handle to the project. The `IRPProject` object is a singleton instance that aggregates all other instances. This class inherits from `IRPPackage`.

*Project is a concrete interface that inherits from `IRPPackage`.*

#### VB Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>activeComponent</td>
<td><code>RPComponent</code></td>
<td>RW</td>
<td>The active component in the package.</td>
</tr>
<tr>
<td>activeConfiguration</td>
<td><code>RPConfiguration</code></td>
<td>RW</td>
<td>The active configuration in the active component. The setting must be to a configuration from the active component, otherwise an error is flagged.</td>
</tr>
<tr>
<td>allStereotypes</td>
<td><code>Collection of RPSTereotypes</code></td>
<td>RO</td>
<td>A collection of all the stereotypes used in the current project.</td>
</tr>
<tr>
<td>components</td>
<td><code>Collection of RPComponents</code></td>
<td>RO</td>
<td>A collection of all the components used in this project.</td>
</tr>
<tr>
<td>defaultDirectoryScheme</td>
<td><code>String</code></td>
<td>RW</td>
<td>The default directory scheme.</td>
</tr>
<tr>
<td>profiles</td>
<td><code>Collection of RPProfiles</code></td>
<td>RO</td>
<td>The collection of profiles used in this project.</td>
</tr>
</tbody>
</table>

#### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>addComponent</code></td>
<td>Adds the specified component to the current project</td>
</tr>
<tr>
<td><code>addPackage</code></td>
<td>Adds the specified package to the current project</td>
</tr>
<tr>
<td>Method</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>addProfile</td>
<td>Adds the specified profile to the current project</td>
</tr>
<tr>
<td>checkEventsBaseIdsSolveCollisions</td>
<td>Checks the values of the events base IDs for all packages in the model, detects collisions between the IDs, and resolves any incorrect values and collisions</td>
</tr>
<tr>
<td>close</td>
<td>Closes the current project</td>
</tr>
<tr>
<td>deleteComponent</td>
<td>Deletes the specified component from the current project</td>
</tr>
<tr>
<td>findComponent</td>
<td>Retrieves the specified component from the current project</td>
</tr>
<tr>
<td>GenerateReport</td>
<td>Generates a ReporterPLUS report for the model.</td>
</tr>
<tr>
<td>getNewCollaboration</td>
<td>Retrieves the new collaboration for the current project</td>
</tr>
<tr>
<td>highlightFromCode</td>
<td>Takes a filename and line number as arguments and then highlights in the Rhapsody browser the element that is associated with the line of code specified.</td>
</tr>
<tr>
<td>importPackageFromRose</td>
<td>Imports the specified package from Rational Rose</td>
</tr>
<tr>
<td>importProjectFromRose</td>
<td>Imports the specified project from Rational Rose</td>
</tr>
<tr>
<td>recalculateEventsBaseIds</td>
<td>Recalculates the events base IDs used by Rhapsody 6.1</td>
</tr>
<tr>
<td>save</td>
<td>Saves the current project</td>
</tr>
<tr>
<td>saveAs</td>
<td>Saves the current project to the specified file name and location</td>
</tr>
<tr>
<td>setActiveComponent</td>
<td>Sets the active configuration for the current project</td>
</tr>
<tr>
<td>setActiveConfiguration</td>
<td>Sets the active configuration for the current project</td>
</tr>
</tbody>
</table>
**addComponent**

**Write method**

**Description**

The `addComponent` method adds the specified component to the current project.

**Visual Basic**

**Syntax**

```
addComponent (name As String) As RPComponent
```

**Arguments**

- `name`
  The name of the component to add

**Return Value**

The `RPComponent` added to the current project

**C/C++ Prototype**

```
HRESULT addComponent (BSTR name, 
                      IRPComponent** component)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
**addPackage**

### Write method

#### Description

The `addPackage` method adds the specified package to the current project.

#### Visual Basic

**Syntax**

```vbnet
addPackage (name As String) As RPPackage
```

**Arguments**

- `name`  
The name of the package to add

**Return Value**

The `RPPackage` added to this project

#### C/C++ Prototype

```c
HRESULT addPackage (BSTR name, IRPPackage** package)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
addProfile

Write method

Description

The addProfile method adds the specified profile to the current project.

Visual Basic

Syntax

addProfile (name As String) As RProfile

Arguments

name

The name of the profile to add

Return Value

The RProfile added to this project

C/C++ Prototype

HRESULT addProfile (BSTR name, IRProfile** profile)

Return Value

HRESULT (0 for success, or a signed integer error code)
checkEventsBaseIdsSolveCollisions

**Read method**

**Description**

The `checkEventsBaseIdsSolveCollisions` method checks the values of the events base IDs for all packages in the model, detects collisions between the IDs, and resolves any incorrect values and collisions.

**Visual Basic**

**Syntax**

```vbnet
checkEventsBaseIdsSolveCollisions()
```

**C/C++ Prototype**

```c
HRESULT checkEventsBaseIdsSolveCollisions()
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
**close**

**Read method**

**Description**

The `close` method closes the current project.

Note that helper applications might not close the current document. Therefore, you should not use `close` in a VBA macro that you specify as a helper.

**Visual Basic**

**Syntax**

```
close()
```

**C/C++ Prototype**

```
HRESULT close()
```

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
**deleteComponent**

**Write method**

**Description**

The `deleteComponent` method deletes the specified component from the current project.

**Visual Basic**

**Syntax**

```visual-basic
deleteComponent (component As RPComponent)
```

**Arguments**

- `component`  
  The component to delete

**C/C++ Prototype**

```c
HRESULT deleteComponent (IRPComponent* component)
```

**Return Value**

- `HRESULT` (0 for success, or a signed integer error code)
findComponent

Read method

Description

The `findComponent` method retrieves the specified component from the current project.

Visual Basic

Syntax

```vbnet
findComponent (name As String) As RPComponent
```

Arguments

name

The name of the component to find

Return Value

The `RPComponent`, or NULL if not found

C/C++ Prototype

```c
HRESULT findComponent (BSTR name, IRPComponent** pVal)
```

Return Value

`HRESULT` (0 for success, or a signed integer error code)
Rhapsody Interfaces

GenerateReport

GenerateReport(modelscope As String, templatename As String, docType As String, filename As String, showDocument As Long, silentMode As Long)

Allows you to generate a ReporterPLUS report for the model. (When this method is used to generate a report, the Rhapsody model is saved before the report is generated.)

modelscope—the name of the package for which the report should be generated. If empty, a report is generated for the entire model. (This is similar to the "scope" command-line option for ReporterPLUS.)

templatename—the name of the template to use. If empty, then the ReporterPLUS report generation wizard will be launched and it will display the name of the last template used.

docType—the type of output to generate (doc, html, ppt, txt). If empty, the ReporterPLUS report generation wizard will be launched and it will display the last output type used.

filename—the filename to use for the generated report. If empty, the ReporterPLUS report generation wizard will be displayed and it will display the filename of the last generated report.

showDocument—In general, the user will be asked if they want to view the report after generation only if they have requested this by selecting View > Options > Ask to open after generating report from the main menu in ReporterPLUS. However, if the user has specified silent generation mode using the parameter silentMode, this parameter can be used to request that the generated document be displayed. To display the report, set this parameter to 1, otherwise use 0.

silentMode—If the template name, document type, or output file name has not been specified using the appropriate parameter, the ReporterPLUS report generation wizard is displayed so the user can provide the missing information. This is the behavior if this parameter is set to 0. If you want to prevent the wizard from being launched in such cases, you can specify silent generation mode by setting this parameter to 1. If set to silent mode, no report will be generated if one or more of the above parameters was not provided. (The report generation status dialog is displayed regardless of the value of this parameter.)

Sample code:

Dim proj As RPProject
Set proj = getProject
proj.GenerateReport "", "C:\Rhapsody\reporterplus\Templates\Class.tpl", "html", "C:\testreport.html", 0, 0
**getNewCollaboration**

*Read method*

**Description**

The `getNewCollaboration` method returns the new collaboration for the current project.

**Visual Basic**

**Syntax**

```vbnet
getNewCollaboration() As RPCollaboration
```

**Return Value**

The `RPCollaboration`

**C/C++ Prototype**

```c
HRESULT getNewCollaboration(
    IRPCollaboration** collaboration)
```

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
**highlightFromCode**

The method `highlightFromCode` takes a filename and line number as arguments and then highlights in the Rhapsody browser the element that is associated with the line of code specified.

The filename argument should consist of the absolute path for the file.

**Syntax**

```vbnet
highlightFromCode(filename As String, lineNumber As Long) As RPModelElement
```

**Example**

```vbnet
dim proj as RPProject
dim m as RPModelElement
set proj = getProject
set m = proj.highlightFromCode("C:\Temp\P\DefaultComponent\DefaultConfig\C.cpp", 30)
```
importPackageFromRose

Write method

Description

The importPackageFromRose method imports the specified package from Rational Rose into Rhapsody 6.1.

Visual Basic

Syntax

importPackageFromRose (projectName As String, packageName As String, logFileName As String)

Arguments

projectName
The name of the project
packageName
The name of the package
logFileName
The name of the log file

C/C++ Prototype

importPackageFromRose (BSTR projectName, BSTR packageName, BSTR logFileName)

Return Value

HRESULT (0 for success, or a signed integer error code)
import Project From Rose

Write method

Description

The `importProjectFromRose` method imports the specified project from Rational Rose into Rhapsody 6.1.

Visual Basic

Syntax

```vbnet
importProjectFromRose (projectName As String, logFileName As String)
```

Arguments

- `projectName` The name of the project
- `logFileName` The name of the log file

C/C++ Prototype

```c
HRESULT importProjectFromRose (BSTR projectName, BSTR logFileName)
```

Return Value

- `HRESULT (0 for success, or a signed integer error code)`
recalculateEventsBaseIds

Write method

Description

The `recalculateEventsBaseIds` method recalculates the events base IDs used by the project.

Visual Basic

Syntax

```visualbasic
recalculateEventsBaseIds ()
```

C/C++ Prototype

```c
HRESULT recalculateEventsBaseIds ()
```

Return Value

`HRESULT` (0 for success, or a signed integer error code)
save

Read method

Description

The `save` method saves the current project.

**Note:** This method flags an error if one occurs.

Visual Basic

Syntax

```vbnet
save()
```

C/C++ Prototype

```c
HRESULT save()
```

Return Value

`HRESULT` (0 for success, or a signed integer error code)
**saveAs**

**Read method**

**Description**

The `saveAs` method saves the current project to the specified file name and location.

**Note:** This method flags an error if one occurs.

**Visual Basic**

**Syntax**

```vbnet
saveAs (filename As String)
```

**Arguments**

- `filename`:
  The name of the file to which to save the project

**C/C++ Prototype**

```c
HRESULT saveAs (BSTR filename)
```

**Return Value**

- `HRESULT (0 for success, or a signed integer error code)`: 


setActiveComponent
Write method

Description
The setActiveComponent method sets the active component for the current project.

Note: This method flags an error if one occurs.

Visual Basic

Syntax
setActiveComponent (name As String)

Arguments
name
The name of the active component

C/C++ Prototype

HRESULT setActiveComponent (BSTR name)

Return Value

HRESULT (0 for success, or a signed integer error code)
**setActiveConfiguration**

*Write method*

**Description**

The `setActiveConfiguration` method sets the active configuration for the current project.

**Visual Basic**

**Syntax**

```vbnet
setActiveConfiguration (name As String)
```

**Arguments**

- **name**
  
  The name of the active configuration

**C/C++ Prototype**

```c
HRESULT setActiveConfiguration (BSTR name)
```

**Return Value**

- `HRESULT` (0 for success, or a signed integer error code)
IRPRelation Interface

The IRPRelation interface represents a relationship between two classes (ofClass and otherClass). It inherits from IRPUnit.

VB Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inverse</td>
<td>RPRelation</td>
<td>RO</td>
<td>If the relation is symmetric, this is a pointer to the peer relation.</td>
</tr>
<tr>
<td>isNavigable</td>
<td>Long</td>
<td>RW</td>
<td>A flag indicating whether the relation is navigable.</td>
</tr>
<tr>
<td>isSymmetric</td>
<td>Long</td>
<td>RO</td>
<td>A flag indicating whether the relation is bidirectional.</td>
</tr>
<tr>
<td>multiplicity</td>
<td>String</td>
<td>RW</td>
<td>The multiplicity of the relation.</td>
</tr>
<tr>
<td>ObjectAsObjectType</td>
<td>RPClass</td>
<td>RO</td>
<td>If this relation is a Rhapsody in C object, it is returned as a class.</td>
</tr>
<tr>
<td>ofClass</td>
<td>RPClassifier</td>
<td>RW</td>
<td>The source class of the relation.</td>
</tr>
<tr>
<td>otherClass</td>
<td>RPClassifier</td>
<td>RW</td>
<td>The target class of the relation.</td>
</tr>
<tr>
<td>qualifier</td>
<td>String</td>
<td>RW</td>
<td>The qualifier of the relation, if one exists.</td>
</tr>
</tbody>
</table>
## Method Summary

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>relationLabel</td>
<td>String</td>
<td>RW</td>
<td>The link name given to the relation.</td>
</tr>
<tr>
<td>relationLinkName</td>
<td>String</td>
<td>RW</td>
<td>The name of the relation link.</td>
</tr>
<tr>
<td>relationRoleName</td>
<td>String</td>
<td>RW</td>
<td>The name of role of the participating elements in the relation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A relation consists of two designations: a role name and a relation name.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>For example, two people can be in a relation called “marriage” (relation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>name) with each person designated by their role within the marriage as</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>“spouse” (role name). For IRPRelation objects, the relation name is mapped</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>to the IRPModelElement property name and the property relationRoleName is</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>provided for the relation’s role name.</td>
</tr>
<tr>
<td>relationType</td>
<td>String</td>
<td>RW</td>
<td>The relation type (Association, Aggregation, or Composition).</td>
</tr>
<tr>
<td>visibility</td>
<td>String</td>
<td>RW</td>
<td>The visibility of the relation (Public, Protected, or Private).</td>
</tr>
</tbody>
</table>

### isTypelessObject

**Tests an object to see if it is defined explicitly or implicitly**

### makeUnidirect

**Changes the current relation from a unidirectional (symmetric) one to one that is directional from the me of this relation to me’s inverse**

### setInverse

**Adds or updates the inverse relation**

**isTypelessObject**

*Read method*
Description

The `isTypelessObject` method tests an object to see if it is defined explicitly (“object of type X”) or implicitly (“typeless” or “unique”).

Visual Basic

Syntax

```vbnet
isTypelessObject() As Long
```

Return Value

1 if the relation is typeless; otherwise 0

C/C++ Prototype

```c
HRESULT isTypelessObject (long *pVal)
```

Return Value

HRESULT (0 for success, or a signed integer error code)
**makeUnidirect**

**Write method**

**Description**

The `makeUnidirect` method changes the current relation from a unidirectional (symmetric) one to one that is directional from the `me` of this relation to `me`’s inverse.

**Visual Basic**

**Syntax**

`makeUnidirect()`

**C/C++ Prototype**

`HRESULT makeUnidirect()`

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
**setInverse**

Write method

Description

The *setInverse* method adds or updates the inverse relation. It provides a means for turning a unidirectional relation into a symmetric one.

Visual Basic

Syntax

```visualbasic
setInverse (roleName As String, linkType As String)
```

Arguments

- **roleName**
  The role name for the relation
- **linkType**
  The type of link (unidirectional or symmetric)

C/C++ Prototype

```c
HRESULT setInverse (BSTR roleName, BSTR linkType)
```

Return Value

- **HRESULT** (0 for success, or a signed integer error code)
IRPRequirement Interface

The IRPRequirement interface represents a Rhapsody requirement. It inherits from IRPAnnotation.

IRPSequenceDiagram Interface

The IRPSequenceDiagram interface represents a sequence diagram. It inherits from IRPDiagram.

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getLogicalCollaboration</td>
<td>Retrieves the logic behind the collaboration diagram</td>
</tr>
<tr>
<td>getRelatedUseCases</td>
<td>Retrieves use cases related to the current sequence diagram</td>
</tr>
</tbody>
</table>

getLogicalCollaboration

Read method

Description

The getLogicalCollaboration method retrieves the logic behind the collaboration diagram.

Visual Basic

Syntax

getLogicalCollaboration() As RPCollaboration

Return Value

The collaboration diagram

C/C++ Prototype

HRESULT getLogicalCollaboration (IRPCollaboration** collaboration)

Return Value

HRESULT (0 for success, or a signed integer error code)

getRelatedUseCases

Read method
Description

The `getRelatedUseCases` method retrieves use cases related to the current sequence diagram.

Visual Basic

Syntax

```vbnet
getRelatedUseCases() As RPCollection
```

Return Value

A collection of use cases related to this sequence diagram

C/C++ Prototype

```c
HRESULT getRelatedUseCases (IRPCollection** pVal)
```

Return Value

```c
HRESULT (0 for success, or a signed integer error code)
```
IRPState Interface

The **IRPState** interface represents a state in a statechart. It inherits from **IRPStateVertex**.

**VB Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>defaultTransition</td>
<td>RPTransition*</td>
<td>RO</td>
<td>The default transition of this state, if there is one.</td>
</tr>
<tr>
<td>entryAction</td>
<td>String</td>
<td>RW</td>
<td>The actions executed when this state is entered.</td>
</tr>
<tr>
<td>exitAction</td>
<td>String</td>
<td>RW</td>
<td>The actions executed when this state is exited.</td>
</tr>
<tr>
<td>isOverridden</td>
<td>Long</td>
<td>RO</td>
<td>If this is equal to 1 (as opposed to 0), the state is overridden.</td>
</tr>
<tr>
<td>isReferenceActivity</td>
<td>Long</td>
<td>RO</td>
<td>If this is equal to 1 (as opposed to 0), the state is an activity reference.</td>
</tr>
<tr>
<td>itsStateChart</td>
<td>RPStateChart</td>
<td>RO</td>
<td>The statechart of this state.</td>
</tr>
<tr>
<td>itsSwimlane</td>
<td>RPSwimlane</td>
<td>RW</td>
<td>The swimlane of this state.</td>
</tr>
<tr>
<td>nestedStateChart</td>
<td>RPStateChart</td>
<td>RO</td>
<td>The statechart nested inside of this state.</td>
</tr>
<tr>
<td>referenceToActivity</td>
<td>RModelElement</td>
<td>RW</td>
<td>The referenced activity or activity diagram.</td>
</tr>
</tbody>
</table>
**Method Summary**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stateType</td>
<td>String</td>
<td>RW</td>
<td>The type of this state. The possible values are as follows:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Or—state that contains no concurrent states</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• And—state that contains two or more concurrent states</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• LocalTermination—termination state element</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Block—action block element</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Action—action element</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• SubActivity—subactivity element</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• ObjectFlow—object node element</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• ReferenceActivity—call behavior element</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• CallOperation—call operation element</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• EventState—send action element</td>
</tr>
<tr>
<td>subStateVertices</td>
<td>RPCollection of RPStateVertex</td>
<td>RO</td>
<td>A collection of transitions and states that connect to this state.</td>
</tr>
</tbody>
</table>

**addConnector** Adds a connector to the statechart

**addState** Adds a state to the statechart

**addStaticReaction** Adds a static reaction to the statechart

**addTerminationState** Adds a termination state to the statechart

**createDefaultTransition** Creates a default transition in the statechart

**createNestedStatechart** Creates a nested statechart

**deleteConnector** Deletes the specified connector from the statechart

**deleteStaticReaction** Deletes the specified static reaction from the statechart
addConnector

Write method

Description

The **addConnector** method adds a connector to the current state.

Visual Basic

Syntax

```vbnet
addConnector (type As String) As RPConnector
```

Arguments

- **type**
  
  The connector type. The possible values are as follows:

<table>
<thead>
<tr>
<th>Condition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Returns the full text name of this state within its statecharts</td>
</tr>
<tr>
<td></td>
<td>Returns the base state from which the current state inherits</td>
</tr>
<tr>
<td></td>
<td>Retrieves the list of logical states</td>
</tr>
<tr>
<td></td>
<td>Returns a collection of static reaction transitions originating from the current state</td>
</tr>
<tr>
<td></td>
<td>Returns a collection of substates belonging to the current state</td>
</tr>
<tr>
<td></td>
<td>Determines whether this state is an And state</td>
</tr>
<tr>
<td></td>
<td>Determines whether the current state is a compound state</td>
</tr>
<tr>
<td></td>
<td>Determines whether the current state is a leaf state</td>
</tr>
<tr>
<td></td>
<td>Determines whether the current state is a root state</td>
</tr>
<tr>
<td></td>
<td>Overrides inheritance for the current state</td>
</tr>
<tr>
<td></td>
<td>Resets the inheritance of the entry action of the current state</td>
</tr>
<tr>
<td></td>
<td>Resets the inheritance of the exit action of the current state</td>
</tr>
<tr>
<td></td>
<td>Sets the static reaction for the current state</td>
</tr>
<tr>
<td></td>
<td>Removes the override on inheritance for this state</td>
</tr>
</tbody>
</table>
Fork
History
Join
Termination

Return Value

The new connector

C/C++ Prototype

HRESULT addConnector (BSTR type,
IRPConnector** connector)

Return Value

HRESULT (0 for success, or a signed integer error code)
**addState**

*Write method*

**Description**

The `addState` method adds a new state to the statechart.

**Visual Basic**

**Syntax**

```vbscript
addState (name As String) As RPState
```

**Arguments**

- `name`  
The name of the new state

**Return Value**

The new state added to the statechart

**C/C++ Prototype**

```c
HRESULT addState (BSTR name, IRPState** state)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
**addStaticReaction**

**Write method**

**Description**

The `addStaticReaction` method adds a static reaction to the state.

**Visual Basic**

**Syntax**

```vbnet
addStaticReaction (trigger As RPInterfaceItem) As RPTransition
```

**Arguments**

- `trigger`
  
  The trigger to add to the statechart

**Return Value**

- The new static reaction

**C/C++ Prototype**

```c
HRESULT addStaticReaction (IRPInterfaceItem* trigger, 
                        IRPTransition** pVal)
```

**Return Value**

- `HRESULT (0 for success, or a signed integer error code)`
addTerminationState

Write method

Description

The addTerminationState method adds a termination state to the statechart.

Visual Basic

Syntax

addTerminationState() As RPState

Return Value

The new termination state

C/C++ Prototype

HRESULT addTerminationState (IRPState** state)

Return Value

HRESULT (0 for success, or a signed integer error code)
**createDefaultTransition**

**Write method**

**Description**

The `createDefaultTransition` method creates a default transition.

**Visual Basic**

**Syntax**

```vbnet
createDefaultTransition (from As RPState) As RPTransition
```

**Arguments**

`from`:

The default state to which the default transition points

**Return Value**

The default transition

**C/C++ Prototype**

```c
HRESULT createDefaultTransition (IRPState* from,
                                 IRPTransition** transition)
```

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
createNestedStatechart
Write method

Description
The `createNestedStatechart` method creates a nested statechart (substatechart).

Visual Basic
Syntax

createNestedStatechart() As RPStatechart

Return Value
The nested statechart

C/C++ Prototype

HRESULT createNestedStatechart (IRPStatechart** pVal)

Return Value
HRESULT (0 for success, or a signed integer error code)
**deleteConnector**

**Write method**

**Description**

The `deleteConnector` method deletes the specified connector from the statechart.

**Visual Basic**

**Syntax**

`deleteConnector (connector As RPConnector)`

**Arguments**

- `connector`
  
  The connector to delete

**C/C++ Prototype**

```c
HRESULT deleteConnector (IRPConnector* connector)
```

**Return Value**

- `HRESULT (0 for success, or a signed integer error code)`
**deleteStaticReaction**

**Write method**

**Description**

The `deleteStaticReaction` method deletes the specified static reaction.

**Visual Basic**

**Syntax**

```vb
deleteStaticReaction (pVal As RPTransition)
```

**Argument**

- `pVal`
  
The static reaction to delete

**C/C++ Prototype**

```c
HRESULT deleteStaticReaction (IRPTransition *pVal)
```

**Return Value**

- `HRESULT` (0 for success, or a signed integer error code)

**entryAction**

**Write method**

**Description**

The `entryAction` method specifies an entry action for the state.

**Visual Basic**

**Syntax**

```vb
entryAction(body As String)
```

**Arguments**

- `body`
  
The entry action

**C/C++ Prototype**

```c
HRESULT entryAction(BSTR body)
```
Return Value

HRESULT (0 for success, or a signed integer error code)

**exitAction**

Write method

Description

The `exitAction` method defines an exit action for the state.

Visual Basic

Syntax

`exitAction(body As String)`

Arguments

- **body**
  The exit action

C/C++ Prototype

`HRESULT exitAction(BSTR body)`

Return Value

HRESULT (0 for success, or a signed integer error code)

**getFullNameInStatechart**

Read method

Description

The `getFullNameInStatechart` method returns the full text name of this state within its statecharts.

Dot notation is used to indicate statechart nesting. For example, if statechart C is in statechart B, which is in statechart A, the full text name of the C statechart is A.B.C.

Visual Basic

Syntax

`getFullNameInStatechart() As String`
Return Value

The full textual name of a state within its statecharts

C/C++ Prototype

HRESULT getFullNameInStatechart (BSTR* pVal)

Return Value

HRESULT (0 for success, or a signed integer error code)

getInheritsFrom

Read method

Description

The getInheritsFrom method returns the base state from which the current state inherits.

Visual Basic

Syntax

getInheritsFrom() As RPState

Return Value

The base state that this state inherits from

C/C++ Prototype

HRESULT getInheritsFrom (IRPState** pVal)

Return Value

HRESULT (0 for success, or a signed integer error code)

getLogicalStates

Read method

Description

The getLogicalStates method retrieves the list of logical states.

Visual Basic

Syntax

getLogicalStates() As RPCollection
Return Value
The list of logical states

C/C++ Prototype
HRESULT getLogicalStates (IRPCollection** pVal)

Return Value
HRESULT (0 for success, or a signed integer error code)

getStaticReactions
Read method

Description
The getStaticReactions method returns a collection of static reaction transitions originating from the current state.

Given a transition with a trigger T, guard condition G, and static reactions A, if T occurs and G is true, the static reactions (also known as reactions in state) are executed while the object is still in its original state.

Visual Basic

Syntax
getStaticReactions() As RPCollection

Return Value
A collection of the static reaction transitions originating from the current state

C/C++ Prototype
HRESULT getStaticReactions (IRPCollection** pVal)

Return Value
HRESULT (0 for success, or a signed integer error code)

getSubStates
Read method

Description
The getSubStates method returns a collection of substates belonging to the current state.
Typically, this method retrieves the state members of a state ("substates"), unless the state contains a nested statechart. In this case, to see the substates, you must descend further into the nested statechart.

**Visual Basic**

**Syntax**

getSubStates() As RPCollection

**Return Value**

A collection of nested substates belonging to this state

**C/C++ Prototype**

HRESULT getSubStates (IRPCollection** pVal)

**Return Value**

HRESULT (0 for success, or a signed integer error code)

**isAnd**

**Read method**

**Description**

The `isAnd` method determines whether this state is an And state.

**Visual Basic**

**Syntax**

isAnd() As Long

**Return Value**

1 if this state is an And state; otherwise 0

**C/C++ Prototype**

HRESULT isAnd (long* pVal)

**Return Value**

HRESULT (0 for success, or a signed integer error code)

**isCompound**

**Read method**
**Description**

The `isCompound` method determines whether the current state is a compound state.

**Visual Basic**

**Syntax**

```vbnet
isCompound() As Long
```

**Return Value**

1 if this state is a compound state; otherwise 0

**C/C++ Prototype**

```c
HRESULT isCompound (long* pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)

---

**isLeaf**

**Read method**

**Description**

The `isLeaf` method determines whether the current state is a leaf state.

**Visual Basic**

**Syntax**

```vbnet
isLeaf() As Long
```

**Return Value**

1 if this state is a leaf state; otherwise 0

**C/C++ Prototype**

```c
HRESULT isLeaf (long* pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)

---

**isRoot**

**Read method**
Description

The `isRoot` method determines whether the current state is a root state.

Visual Basic

Syntax

```vbnet
isRoot() As Long
```

Return Value

1 if this state is a root state; otherwise 0

C/C++ Prototype

```c
HRESULT isRoot (long* pVal)
```

Return Value

HRESULT (0 for success, or a signed integer error code)

**overrideInheritance**

**Note**

Currently, this method has not been implemented.

Write method

Description

The `overrideInheritance` method overrides inheritance for the current state.

Visual Basic

Syntax

```vbnet
overrideInheritance()
```

C/C++ Prototype

```c
HRESULT overrideInheritance()
```

Return Value

HRESULT (0 for success, or a signed integer error code)
resetEntryActionInheritance

Write method

Description

The resetEntryActionInheritance method resets the inheritance of the entry action of the current state.

Visual Basic

Syntax

resetEntryActionInheritance() As RPState

Return Value

The updated state

C/C++ Prototype

HRESULT resetEntryActionInheritance (IRPState** pVal)

resetExitActionInheritance

Write method

Description

The resetExitActionInheritance method resets the inheritance of the exit action for the current state.

Visual Basic

Syntax

resetExitActionInheritance() As RPState

Return Value

The updated state

C/C++ Prototype

HRESULT resetExitActionInheritance (IRPState** pVal)
Return Value

HRESULT (0 for success, or a signed integer error code)

**setStaticReaction**

Write method

Description

The **setStaticReaction** method sets the static reaction for the current state.

Visual Basic

Syntax

```visualbasic
setStaticReaction (trigVal As String, guardVal As String, actionVal As String)
```

Arguments

- `trigVal`
  The new value for the trigger
- `guardVal`
  The new value for the guard
- `actionVal`
  The new value for the action

C/C++ Prototype

```c
HRESULT setStaticReaction (BSTR trigVal, BSTR guardVal, BSTR actionVal)
```

Return Value

HRESULT (0 for success, or a signed integer error code)

**unoverrideInheritance**

Note

Currently, this method has not been implemented.

Write method

Description

The **unoverrideInheritance** method removes the override on inheritance for the current state.
Visual Basic

Syntax

unoverrideInheritance()

C/C++ Prototype

HRESULT unoverrideInheritance()

Return Value

HRESULT (0 for success, or a signed integer error code)
**stateType**

**Read method**

**Description**

The `stateType` method specifies the state type of the current state.

**Visual Basic**

**Syntax**

`stateType(type As String)`

**Arguments**

- `type`  
  The state type. The possible values are as follows:
  - `Or`—state that contains no concurrent states
  - `And`—state that contains two or more concurrent states
  - `LocalTermination`—termination state element
  - `Block`—action block element
  - `Action`—action element
  - `SubActivity`—subactivity element
  - `ObjectFlow`—object node element
  - `ReferenceActivity`—call behavior element
  - `CallOperation`—call operation element
  - `EventState`—send action element

**C/C++ Prototype**

`HRESULT stateType(BSTR pVal)`

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
IRPStatechart Interface

The IRPStatechart interface represents a statechart diagram. It inherits from IRPDiagram.

Note: You cannot create a statechart using the APIs. The statechart must already exist for you to use the APIs on it.

**VB Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isOverridden</td>
<td>Long</td>
<td>RO</td>
<td>If this is equal to 1 (as opposed to 0), the state is overridden. Currently, this property has not been implemented.</td>
</tr>
<tr>
<td>itsClass</td>
<td>RPClass</td>
<td>RO</td>
<td>The class of this statechart.</td>
</tr>
<tr>
<td>rootState</td>
<td>RPState</td>
<td>RO</td>
<td>The default (starting) state of this statechart.</td>
</tr>
</tbody>
</table>

**Method Summary**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createGraphics</td>
<td>Creates graphics in the Rhapsody statechart</td>
</tr>
<tr>
<td>deleteState</td>
<td>Deletes the specified state from the Rhapsody statechart</td>
</tr>
<tr>
<td>findTrigger</td>
<td>Determines whether the current statechart has a trigger for the specified class interface element</td>
</tr>
<tr>
<td>getAllTriggers</td>
<td>Returns a collection of all the triggers for the current statechart</td>
</tr>
<tr>
<td>getInheritsFrom</td>
<td>Returns a pointer to the base statechart from which the current statechart inherits</td>
</tr>
<tr>
<td>overrideInheritance</td>
<td>Overrides inheritance for the current state</td>
</tr>
<tr>
<td>unoverrideInheritance</td>
<td>Removes the override on inheritance for the current state</td>
</tr>
</tbody>
</table>
**createGraphics**

**Write method**

**Description**

The `createGraphics` method creates graphics in the Rhapsody 6.1 statechart using the information in the COM API methods.

**Visual Basic**

**Syntax**

```vb
createGraphics()
```

**C/C++ Prototype**

```c
HRESULT createGraphics()
```

**Return Value**

```c
HRESULT (0 for success, or a signed integer error code)
```
deleteState
   Write method

Description
   The deleteState method deletes the specified state from the statechart.

Visual Basic

Syntax
   deleteState (state As RPState)

Arguments
   state
       The state to delete

C/C++ Prototype

HRESULT deleteState (IRPState* state)

Return Value

HRESULT (0 for success, or a signed integer error code)
**findTrigger**

**Read method**

**Description**

The `findTrigger` method determines whether the current statechart has a trigger for the specified class interface element.

**Visual Basic**

**Syntax**

```vbnet
findTrigger (item As RPInterfaceItem) As Long
```

**Arguments**

- `item`
  
  The state to check

**Return Value**

1 if this statechart has a trigger; otherwise 0

**C/C++ Prototype**

```c
HRESULT findTrigger (IRPInterfaceItem* item, long *pVal)
```

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
**getAllTriggers**

*Read method*

**Description**

The `getAllTriggers` method returns a collection of all the triggers for the current statechart.

**Visual Basic**

**Syntax**

```
getAllTriggers() As RPICollection
```

**Return Value**

A collection of all the triggers (`RPInterfaceItems`) for this statechart.

**C/C++ Prototype**

```
HRESULT getAllTriggers (IRPCollection** pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
getInheritsFrom

Read method

Description

The `getInheritsFrom` method returns a pointer to the base statechart from which the current statechart inherits.

Visual Basic

Syntax

```
getInheritsFrom() As RPStatechart
```

Return Value

The base statechart from which this statechart inherits

C/C++ Prototype

```
HRESULT getInheritsFrom (IRPStatechart** pVal)
```

Return Value

`HRESULT (0 for success, or a signed integer error code)`
overrideInheritance

Note
Currently, this method has not been implemented.

Write method

Description
The overrideInheritance method overrides inheritance for the current state.

Visual Basic

Syntax
overrideInheritance()

C/C++ Prototype

HRESULT overrideInheritance()

Return Value
HRESULT (0 for success, or a signed integer error code)
**unoverrideInheritance**

**Note**

Currently, this method has not been implemented.

**Write method**

**Description**

The `unoverrideInheritance` method removes the override on inheritance for the current state.

**Visual Basic**

**Syntax**

```vbnet
unoverrideInheritance()
```

**C/C++ Prototype**

```c
HRESULT unoverrideInheritance()
```

**Return Value**

HRESULT (0 for success, or a signed integer error code)
IRPStateVertex Interface

The IRPStateVertex interface represents all model elements that can be connectors or states. It is an abstract interface that inherits from IRPModelElement.

VB Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent</td>
<td>RPState</td>
<td>RW</td>
<td>The parent state or connector</td>
</tr>
</tbody>
</table>

Method Summary

- **addTransition**
  - Creates a transition

- **deleteTransition**
  - Deletes a transition

- **getInTransitions**
  - Returns a collection of transitions that are directed into the current state or connector

- **getOutTransitions**
  - Returns a collection of transitions that are directed out of the current state or connector

addTransition

Write method

**Description**

The **addTransition** method creates a transition.

**Visual Basic**

**Syntax**

```
addTransition (to As IRPStateVertex) As IRPTransition
```

**Arguments**

- `to`
  - The “to” state for the transition

**Return Value**

- The new transition
C/C++ Prototype

HRESULT addTransition (IRPStateVertex *to, IRPTransition** transition)

Return Value

HRESULT (0 for success, or a signed integer error code)

deleteTransition
Write method

Description
The deleteTransition method deletes the specified transition.

Visual Basic

Syntax

deleteTransition (transition As RPTransition)

Arguments

transition
The transition to delete

C/C++ Prototype

HRESULT deleteTransition (IRPTransition *transition)

Return Value

HRESULT (0 for success, or a signed integer error code)

getInTransitions
Read method

Description
The getInTransitions method returns a collection of transitions that are directed into the current state or connector.

Visual Basic

Syntax

getInTransitions() As RPCollection
Return Value

A collection of transitions going into this state or connector

C/C++ Prototype

HRESULT getInTransitions (IRPCollection** pVal)

Return Value

HRESULT (0 for success, or a signed integer error code)
getOutTransitions

Read method

Description

The `getOutTransitions` method returns a collection of transitions that are directed out of the current state or connector.

Visual Basic

Syntax

```vb
getOutTransitions() As RPCollection
```

Return Value

A collection of transitions going out of this state or connector

C/C++ Prototype

```c
HRESULT getOutTransitions (IRPCollection** pVal)
```

Return Value

```c
HRESULT (0 for success, or a signed integer error code)
```
parent

Read method

Description

The parent method returns the parent state.

Visual Basic

Syntax

parent(newVal As RPState)

Arguments

newVal

The parent state

C/C++ Prototype

HRESULT parent(IRPState* newVal)

Return Value

HRESULT (0 for success, or a signed integer error code)
Rhapsody API Interfaces

IRPStereotype Interface

The IRPStereotype interface represents a stereotype in the model. It inherits from IRPModelElement.

**VB Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>icon</td>
<td>String</td>
<td>RO</td>
<td>The icon string attached to the stereotype</td>
</tr>
<tr>
<td>ofMetaClass</td>
<td>String</td>
<td>RO</td>
<td>The metaclass to which the stereotype applies</td>
</tr>
</tbody>
</table>

IRPStructureDiagram Interface

The IRPStructureDiagram interface represents a Rhapsody structure diagram. It inherits from IRPDiagram.

IRPSwimlane Interface

The IRPSwimlane interface represents a swimlane in an activity diagram. It inherits from IRPModelElement.

**VB Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>contents</td>
<td>RPCollection</td>
<td>RO</td>
<td>A collection of states in the swimlane</td>
</tr>
<tr>
<td>represents</td>
<td>RPModelElement</td>
<td>RW</td>
<td>The object that implements the swimlane</td>
</tr>
</tbody>
</table>
IRPTag Interface

The IRPTag interface represents a tag. It inherits from IRPVariable.

**VB Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tagMetaClass</td>
<td>String</td>
<td>RW</td>
<td>The metaclass for the tag</td>
</tr>
<tr>
<td>value</td>
<td>String</td>
<td>RW</td>
<td>The default value for the tag</td>
</tr>
</tbody>
</table>

IRPTemplateInstantiation Interface

The IRPTemplateInstantiation interface represents a global variable in a Rhapsody model. It inherits from IRPModelElement.

**VB Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>templateInstantiationParameters</td>
<td>Collection of RPTemplateInstantiationParameters</td>
<td>RO</td>
<td>A collection of parameters used for instantiation</td>
</tr>
</tbody>
</table>
IRPTemplateInstantiationParameter Interface

The IRPTemplateInstantiationParameter interface represents a parameter used in template instantiation in a Rhapsody model. It inherits from the IRPModelElement.

VB Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>argValue</td>
<td>String</td>
<td>RW</td>
<td>The argument value for this parameter of a template instantiation</td>
</tr>
</tbody>
</table>

IRPTemplateParameter Interface

The IRPTemplateParameter interface represents a parameter for a template in a Rhapsody model. It inherits from IRPVariable.

VB Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>typeName</td>
<td>RPType</td>
<td>RW</td>
<td>The type of this template parameter</td>
</tr>
</tbody>
</table>

Method Summary

| setClassType | Sets or changes the current template parameter to a class type parameter |
**setClassType**

**Write method**

**Description**

The `setClassType` method sets or changes the current template parameter to a class type parameter. For example, parameter `<int X>` becomes `<class X>`.

**Visual Basic**

**Syntax**

```
setClassType()
```

**C/C++ Prototype**

```
HRESULT setClassType()
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
IRPTransition Interface

The IRPTransition interface represents a transition in a statechart. It inherits from IRPModelElement.

VB Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isOverridden</td>
<td>Long</td>
<td>RO</td>
<td>If this is equal to 1 (as opposed to 0), the transition is overridden.</td>
</tr>
<tr>
<td>itsLabel</td>
<td>String</td>
<td>RO</td>
<td>The transition label for this transition.</td>
</tr>
<tr>
<td>itsSource</td>
<td>RPStateVertex</td>
<td>RW</td>
<td>The source state of this transition.</td>
</tr>
<tr>
<td>itsStateChart</td>
<td>RPStatechart</td>
<td>RW</td>
<td>The statechart of this transition.</td>
</tr>
<tr>
<td>itsTarget</td>
<td>RPStateVertex</td>
<td>RW</td>
<td>The target state of this transition.</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getInheritsFrom</td>
<td>Returns the base transition from which the current transition inherits</td>
</tr>
<tr>
<td>getItsAction</td>
<td>Returns the action code of the current transition</td>
</tr>
<tr>
<td>getItsGuard</td>
<td>Returns the guard condition of the current transition</td>
</tr>
<tr>
<td>getItsTrigger</td>
<td>Returns the trigger (event or triggered operation) of the current transition</td>
</tr>
<tr>
<td>getOfState</td>
<td>Returns the source state for which this transition is the default transition</td>
</tr>
<tr>
<td>isDefaultTransition</td>
<td>Determines whether the current transition is a default transition</td>
</tr>
<tr>
<td>isStaticReaction</td>
<td>Determines whether this is a static reaction</td>
</tr>
<tr>
<td>itsCompoundSource</td>
<td>Returns a collection of states that act as multiple sources for this single transition</td>
</tr>
<tr>
<td>overrideInheritance</td>
<td>Overrides inheritance for the current transition</td>
</tr>
<tr>
<td>resetLabelInheritance</td>
<td>Resets the label inheritance</td>
</tr>
<tr>
<td>setItsAction</td>
<td>Updates the current transition with a new action</td>
</tr>
</tbody>
</table>
**getInheritsFrom**

**Read method**

**Description**

The `getInheritsFrom` method returns the base transition from which the current transition inherits.

**Visual Basic**

**Syntax**

```vbnet
getInheritsFrom() As RPTransition
```

**Return Value**

The base transition from which this transition inherits

**C/C++ Prototype**

```c
HRESULT getInheritsFrom (IRPTransition** pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
**getItsAction**

*Read method*

**Description**

The `getItsAction` method returns the action code of the current transition.

**Visual Basic**

**Syntax**

```vbnet
getItsAction() As RPAction
```

**Return Value**

The action code of this transition

**C/C++ Prototype**

```c
HRESULT getItsAction (IRPAction** pVal)
```

**Return Value**

`HRESULT (0 for success, or a signed integer error code)}`
getItsGuard

Read method

Description

The `getItsGuard` method returns the guard condition of the current transition.

Visual Basic

Syntax

```vbnet
getItsGuard() As RPGuard
```

Return Value

The guard condition of this transition

C/C++ Prototype

```c
HRESULT getItsGuard (IRPGuard** pVal)
```

Return Value

`HRESULT (0 for success, or a signed integer error code)`
getItsTrigger

Read method

Description

The getItsTrigger method returns the trigger (event or triggered operation) of the current transition.

Visual Basic

Syntax

getItsTrigger() As RPTrigger

Return Value

The trigger of this transition

C/C++ Prototype

HRESULT getItsTrigger (IRPTrigger** pVal)

Return Value

HRESULT (0 for success, or a signed integer error code)

Example

The following macro checks each transition to see if it has a trigger.

Sub checkNullTransitions()
    Dim elem As RPMModelElement
    For Each elem In getProject.getNestedElementsRecursive
        If elem.metaClass = "Transition" Then
            Dim trans As RPTransition
            Set trans = elem
            If trans.getItsTrigger Is Nothing Then
                Debug.Print "The trigger in transition " +
                trans.getFullPathName + "' is null!"
            End If
        End If
    Next elem
End Sub
**getOfState**

**Read method**

**Description**

The `getOfState` method returns the source state for which this transition is the default transition.

Suppose you want to figure out what event sequences lead to a state A. One way to retrieve those values is to travel backwards from A, looking for all the transitions going into it. If they are normal transitions, you can continue to their source. If they are default transitions, you must find the parent using the method `getOfState`.

**Visual Basic**

**Syntax**

```plaintext
getOfState() As RPState
```

**Return Value**

The parent state for which this transition is the default transition. If this transition is the default transition of its statechart, this method returns the parent; otherwise, it returns a NULL value.

**C/C++ Prototype**

```plaintext
HRESULT getOfState (IRPState** pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
**isDefaultTransition**

**Read method**

**Description**

The `isDefaultTransition` method determines whether the current transition is a default transition.

**Visual Basic**

**Syntax**

`isDefaultTransition() As Long`

**Return Value**

1 if this transition is a default transition; otherwise 0

**C/C++ Prototype**

`HRESULT isDefaultTransition (long *pVal)`

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`

**isStaticReaction**

**Read method**

**Description**

The `isStaticReaction` method determines whether this is a static reaction.

**Visual Basic**

**Syntax**

`isStaticReaction() As Long`

**Return Value**

1 if this is a static reaction; otherwise 0

**C/C++ Prototype**

`HRESULT isStaticReaction (long *pVal)`

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
**itsCompoundSource**

**Read method**

**Description**

The `itsCompoundSource` method returns a collection of states that act as multiple sources for this single transition.

For example, consider a junction connector. There can be many transitions from different states that are resolved into one transition leaving a junction connector. For the transition leaving a junction connector, this method gives all the source states.

**Visual Basic**

**Syntax**

```vbnet
itsCompoundSource() As RPCollection
```

**Return Value**

A collection of source states (`RPStateVertexes`) for this transition

**C/C++ Prototype**

```c
HRESULT itsCompoundSource (IRPCollection** pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
overrideInheritance

Note

Currently, this method has not been implemented.

Write method

Description

The overrideInheritance method overrides inheritance for the current transition.

Visual Basic

Syntax

overrideInheritance()

C/C++ Prototype

HRESULT overrideInheritance()

Return Value

HRESULT (0 for success, or a signed integer error code)
Rhapsody Interfaces

resetLabelInheritance
Write method

Description
The `resetLabelInheritance` method resets the label inheritance.

Visual Basic
Syntax
resetLabelInheritance() As RPTransition

Return Value
The updated RPTransition

C/C++ Prototype
HRESULT resetLabelInheritance (IRPTransition** pVal)

Return Value
HRESULT (0 for success, or a signed integer error code)

setItsAction
Write method

Description
The `setItsAction` method updates the current transition with a new action.

Visual Basic
Syntax
setItsAction (action As String) As RPAction

Return Value
The new action for the transition

C/C++ Prototype
HRESULT setItsAction (BSTR action, IRPAction** pVal)

Return Value
HRESULT (0 for success, or a signed integer error code)
setItsGuard
Write method

Description

The `setItsGuard` method updates the current transition with a new guard.

Visual Basic

Syntax

```vbnet
setItsGuard() As RPGuard
```

Return Value

The new guard for this transition

C/C++ Prototype

```c
HRESULT setItsGuard (BSTR guard, IRPGuard** pVal)
```

Return Value

`HRESULT` (0 for success, or a signed integer error code)

setItsLabel
Write method

Description

The `setItsLabel` method updates this transition with a new label (trigger[guard]/action)

Visual Basic

Syntax

```vbnet
setItsLabel (trigger As String, guard As String,
            action As String)
```

Arguments

- `trigger`
  The new trigger value for this transition
- `guard`
  The new guard value for this transition
- `action`
  The new action value for this transition
**C/C++ Prototype**

```
HRESULT setItsLabel (BSTR trigger, BSTR guard, 
                    BSTR action)
```

**Return Value**

HRESULT (0 for success, or a signed integer error code)

**setItsTrigger**

**Write method**

**Description**

The `setItsTrigger` method updates the current transition with a new trigger.

**Visual Basic**

**Syntax**

```
setItsTrigger (trigger As String) As RPTrigger
```

**Return Value**

The new trigger for this transition

**C/C++ Prototype**

```
HRESULT setItsTrigger (BSTR trigger, IRPTrigger** pVal)
```

**Return Value**

HRESULT (0 for success, or a signed integer error code)

**unoverrideInheritance**

**Note**

Currently, this method has not been implemented.

**Write method**

**Description**

The `unoverrideInheritance` method removes the override on inheritance for the current transition.
**Visual Basic**

**Syntax**

`unoverrideInheritance()`

**C/C++ Prototype**

```c
HRESULT unoverrideInheritance()
```

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
IRPTrigger Interface

The IRPTrigger interface represents a trigger of a transition in a statechart. It inherits from IRPModelElement.

VB Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>body</td>
<td>String</td>
<td>RW</td>
<td>The body of this trigger</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getItsOperation</td>
<td>Returns the event or triggered operation of the current trigger</td>
</tr>
<tr>
<td>isOperation</td>
<td>Determines whether the current trigger is an operation (event or triggered operation)</td>
</tr>
<tr>
<td>isTimeout</td>
<td>Determines whether the current trigger is a timeout</td>
</tr>
</tbody>
</table>

getItsOperation

Read method

Description

The getItsOperation method returns the event or triggered operation of the current trigger.

If the current trigger’s transition is labeled \( E[C]/A \) (where \( E \) is the event (event or triggered operation) the trigger refers to, \( C \) is the guard condition, and \( A \) is the action), this method returns the event \( E \) to which this trigger refers.

Visual Basic

Syntax

```vbnet
getItsOperation() As RPInterfaceItem
```

Return Value

The operation of this trigger

C/C++ Prototype

```c
HRESULT getItsOperation (IRPInterfaceItem** pVal)
```
Return Value

HRESULT (0 for success, or a signed integer error code)

isOperation

Read method

Description

The isOperation method determines whether the current trigger is an operation (event or triggered operation).

Visual Basic

Syntax

isOperation() As Long

Return Value

1 if this trigger is an operation; otherwise 0

C/C++ Prototype

HRESULT isOperation (long* pVal)

Return Value

HRESULT (0 for success, or a signed integer error code)
isTimeout

Read method

Description

The `isTimeout` method determines whether the current trigger is a timeout.

Visual Basic

Syntax

`isTimeout() As Long`

Return Value

1 if this trigger is a timeout; otherwise 0

C/C++ Prototype

`HRESULT isTimeout (long* pVal)`

Return Value

`HRESULT (0 for success, or a signed integer error code)`
**IRPType Interface**

The IRPType interface represents Rhapsody 6.1 data types. It inherits from IRPClassifier.

### VB Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>declaration</td>
<td>String</td>
<td>RW</td>
<td>The type declaration.</td>
</tr>
<tr>
<td>enumerationLiterals</td>
<td>Collection of RPEnumeration Literals</td>
<td>RO</td>
<td>A container that can be manipulated only if the kind of the type is Enumerated</td>
</tr>
<tr>
<td>isPredefined</td>
<td>Long</td>
<td>RO</td>
<td>A flag that indicates whether this type is a Rhapsody predefined types. Predefined types are defined in the package unit files: Share\Properties\Predefined&lt;lang&gt;.sbs</td>
</tr>
<tr>
<td>isTypedef</td>
<td>Long</td>
<td>RO</td>
<td>A flag that indicates whether this type is defined with a typedef</td>
</tr>
<tr>
<td>isTypedefConstant</td>
<td>Long</td>
<td>RW</td>
<td>A flag that indicates whether the typedef is defined as a constant (is read-only, such as the const qualifier in C++)</td>
</tr>
<tr>
<td>isTypedefOrdered</td>
<td>Long</td>
<td>RW</td>
<td>A flag that indicated whether the order of the reference type items is significant</td>
</tr>
<tr>
<td>isTypedefReference</td>
<td>Long</td>
<td>RW</td>
<td>A flag that indicates whether the typedef is referenced as a reference (such as a pointer (*) or an address (&amp;) in C++)</td>
</tr>
<tr>
<td>kind</td>
<td>String</td>
<td>RW</td>
<td>Stores the type kind.</td>
</tr>
<tr>
<td>typedefBaseType</td>
<td>RPClassifier</td>
<td>RW</td>
<td>Specifies the basic type of the typedef</td>
</tr>
<tr>
<td>typedefMultiplicity</td>
<td>String</td>
<td>RW</td>
<td>Specifies the multiplicity of the typedef</td>
</tr>
</tbody>
</table>
### Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>addEnumerationLiteral</code></td>
<td>Creates an enumeration literal</td>
</tr>
<tr>
<td><code>isArray</code></td>
<td>Determines whether the current type is an array</td>
</tr>
<tr>
<td><code>isEnum</code></td>
<td>Determines whether the current type is an enumerated type</td>
</tr>
<tr>
<td><code>isEqualTo</code></td>
<td>Tests for equality between the type of the type and the type itself</td>
</tr>
<tr>
<td><code>isImplicit</code></td>
<td>Determines whether the current type is an implicit type</td>
</tr>
<tr>
<td><code>isKindEnumeration</code></td>
<td>Determines whether the current type is an enumeration</td>
</tr>
<tr>
<td><code>isKindLanguage</code></td>
<td>Determines whether the current type is a language declaration type</td>
</tr>
<tr>
<td><code>isKindStructure</code></td>
<td>Determines whether the current type is a structure</td>
</tr>
<tr>
<td><code>isKindTypedef</code></td>
<td>Determines whether the current type is a typedef</td>
</tr>
<tr>
<td><code>isKindUnion</code></td>
<td>Determines whether the current type is a union</td>
</tr>
<tr>
<td><code>isPointer</code></td>
<td>Determines whether the current type is a pointer</td>
</tr>
<tr>
<td><code>isPointerToPointer</code></td>
<td>Determines whether the current type is a pointer to another pointer</td>
</tr>
<tr>
<td><code>isReference</code></td>
<td>Determines whether the current type is a reference</td>
</tr>
<tr>
<td><code>isReferenceToPointer</code></td>
<td>Determines whether the current type is a reference to a pointer</td>
</tr>
<tr>
<td><code>isStruct</code></td>
<td>Determines whether the current type is a struct</td>
</tr>
<tr>
<td><code>isTemplate</code></td>
<td>Determines whether the current type is a template</td>
</tr>
<tr>
<td><code>isUnion</code></td>
<td>Determines whether the current type is a union</td>
</tr>
</tbody>
</table>
**addEnumerationLiteral**

**Write method**

**Description**

The `addEnumerationLiteral` method creates an enumeration literal.

**Visual Basic**

**Syntax**

```visualbasic
addEnumerationLiteral (name As String)
    As RPEnumerationLiteral
```

**Arguments**

- `name`:
  The name of the enumeration literal to create

**Return Value**

The new enumeration literal

**C/C++ Prototype**

```c
HRESULT addEnumerationLiteral (BSTR name,
    IRPEnumerationLiteral** pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
**isArray**

*Read method*

**Description**

The `isArray` method determines whether the current type is an array.

**Visual Basic**

**Syntax**

`isArray() As Long`

**Return Value**

1 if the type is an array; 0 otherwise

**C/C++ Prototype**

`HRESULT isArray (long* pVal)`

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`

**isEnum**

*Read method*

**Description**

The `isEnum` method determines whether the current type is an enumerated type.

**Visual Basic**

**Syntax**

`isEnum() As Long`

**Return Value**

1 if the type is an array; 0 otherwise

**C/C++ Prototype**

`HRESULT isEnum (long* pVal)`

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
isEqualTo

Read method

Description

The isEqualTo method tests for equality between the type of the type and the type itself.

Visual Basic

Syntax

isEqualTo() As Long

Return Value

The method returns 1 if the “type of the type” is equal to the type depended on, otherwise 0.

For example, if the type definition is typedef x, the type is equal to the type it depends on. However, if the type definition is typedef x*, the type of the type is a pointer, and is therefore different from the type itself.

C/C++ Prototype

HRESULT isEqualTo (long* pVal)

Return Value

HRESULT (0 for success, or a signed integer error code)
isImplicit
Read method
Description
The isImplicit method determines whether the current type is an implicit type.
Visual Basic
Syntax
isImplicit() As Long
Return Value
1 if the type is an implicit type; 0 otherwise
C/C++ Prototype
HRESULT isImplicit (long* pVal)
Return Value
HRESULT (0 for success, or a signed integer error code)

isKindEnumeration
Read method
Description
The isKindEnumeration method determines whether the current type is an enumeration.
Visual Basic
Syntax
isKindEnumeration() As Long
Return Value
1 if the type is an enumeration; 0 otherwise
C/C++ Prototype
HRESULT isKindEnumeration (long* pVal)
Return Value
HRESULT (0 for success, or a signed integer error code)
**isKindLanguage**

**Read method**

**Description**

The `isKindLanguage` method determines whether the current type is a language declaration type.

**Visual Basic**

**Syntax**

```vbnet
isKindLanguage() As Long
```

**Return Value**

1 if the type is a language declaration type; 0 otherwise

**C/C++ Prototype**

```c
HRESULT isKindLanguage (long* pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
**isKindStructure**

**Read method**

**Description**

The *isKindStructure* method determines whether the current type is a structure.

**Visual Basic**

**Syntax**

```vbnet
isKindStructure() As Long
```

**Return Value**

1 if the type is a structure; 0 otherwise

**C/C++ Prototype**

```c
HRESULT isKindStructure (long* pVal)
```

**Return Value**

HRESULT (0 for success, or a signed integer error code)
**isKindTypedef**

**Read method**

**Description**

The `isKindTypedef` method determines whether the current type is a `typedef`.

**Visual Basic**

**Syntax**

```vbnet
isKindTypedef() As Long
```

**Return Value**

1 if the type is a `typedef`; 0 otherwise

**C/C++ Prototype**

```c
HRESULT isKindTypedef (long* pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
**isKindUnion**

**Read method**

**Description**

The `isKindUnion` method determines whether the current type is a union.

**Visual Basic**

**Syntax**

```vba
isKindUnion() As Long
```

**Return Value**

1 if the type is a union; 0 otherwise

**C/C++ Prototype**

```c
HRESULT isKindUnion (long* pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)

**isPointer**

**Read method**

**Description**

The `isPointer` method determines whether the current type is a pointer.

**Visual Basic**

**Syntax**

```vba
isPointer() As Long
```

**Return Value**

1 if the type is a pointer; 0 otherwise

**C/C++ Prototype**

```c
HRESULT isPointer (long* pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
**isPointerToPointer**

**Read method**

**Description**

The `isPointerToPointer` method determines whether the current type is a pointer to another pointer.

**Visual Basic**

**Syntax**

```vbnet
isPointerToPointer() As Long
```

**Return Value**

1 if the type is a pointer to a pointer; 0 otherwise

**C/C++ Prototype**

```c
HRESULT isPointerToPointer (long* pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
**isReference**

**Read method**

**Description**

The `isReference` method determines whether the current type is a reference.

**Visual Basic**

**Syntax**

```vbnet
isReference() As Long
```

**Return Value**

1 if the type is a reference; 0 otherwise

**C/C++ Prototype**

```c
HRESULT isReference (long* pVal)
```

**Return Value**

`HRESULT` (0 for success, or a signed integer error code)
isReferenceToPointer

**Read method**

**Description**

The `isReferenceToPointer` method determines whether the current type is a reference to a pointer.

**Visual Basic**

**Syntax**

```vbs
isReferenceToPointer() As Long
```

**Return Value**

1 if this type is a reference to a pointer; otherwise 0

**C/C++ Prototype**

```c
HRESULT isReferenceToPointer (long* pVal)
```

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
**isStruct**

**Read method**

**Description**

The `isStruct` method determines whether the current type is a `struct`.

**Visual Basic**

**Syntax**

`isStruct() As Long`

**Return Value**

1 if this type is a `struct`; otherwise 0

**C/C++ Prototype**

`HRESULT isStruct (long* pVal)`

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
**isTemplate**  
**Write method**

**Description**

The `isTemplate` method determines whether the current type is a template.

**Visual Basic**

**Syntax**

`isTemplate() As Long`

**Return Value**

1 if this type is a template; otherwise 0

**C/C++ Prototype**

`HRESULT isTemplate (long* pVal)`

**Return Value**

`HRESULT (0 for success, or a signed integer error code)`
isUnion

Write method

Description

The isUnion method determines whether the current type is a union.

Visual Basic

Syntax

isUnion() As Long

Return Value

1 if this type is a union; otherwise 0

C/C++ Prototype

HRESULT isUnion (long* pVal)

Return Value

HRESULT (0 for success, or a signed integer error code)
IRPUnit Interface

The IRPUnit interface represents all model elements that can be stored as units for configuration management (CM) purposes. It is an abstract interface that inherits from IRPModelElement.

VB Properties

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMHeader</td>
<td>String</td>
<td>RW</td>
<td>The CM header of this unit</td>
</tr>
<tr>
<td>currentDirectory</td>
<td>String</td>
<td>RO</td>
<td>The current directory</td>
</tr>
<tr>
<td>filename</td>
<td>String</td>
<td>RW</td>
<td>The name of the file that stores the unit</td>
</tr>
<tr>
<td>includeInNextLoad</td>
<td>Long</td>
<td>RW</td>
<td>Indicates whether or not the unit should be loaded the next time the model is loaded.</td>
</tr>
<tr>
<td>isStub</td>
<td>Long</td>
<td>RO</td>
<td>Specifies whether this is a stub</td>
</tr>
<tr>
<td>structureDiagrams</td>
<td>Collection of RPStructure Diagrams</td>
<td>RO</td>
<td>Collection of structure diagrams that can be stored as units</td>
</tr>
</tbody>
</table>

Method Summary

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isReadOnly</td>
<td>Determines whether the current unit is read-only</td>
</tr>
<tr>
<td>isReferenceUnit</td>
<td>Determines whether the current unit was added to the model as a reference</td>
</tr>
<tr>
<td>isSeparateSaveUnit</td>
<td>Determines whether the current unit is saved in its own (separate) file</td>
</tr>
<tr>
<td>load</td>
<td>Loads the specified unit</td>
</tr>
<tr>
<td>save</td>
<td>Saves the specified unit</td>
</tr>
<tr>
<td>setReadOnly</td>
<td>Specifies whether the current unit is read-only</td>
</tr>
<tr>
<td>setSeparateSaveUnit</td>
<td>Sets a unit to be stored to its own file</td>
</tr>
</tbody>
</table>
isReadOnly

Read method

Description

The isReadOnly method determines whether the current unit is read-only.

Visual Basic

Syntax

isReadOnly() As Long

Return Value

1 if this unit is read-only; otherwise 0

C/C++ Prototype

HRESULT isReadOnly (long* pVal)

Return Value

HRESULT (0 for success, or a signed integer error code)
isReferenceUnit
Read method
Description
The isReferenceUnit method determines whether the current unit was added to the model as a reference.
Visual Basic
Syntax
isReferenceUnit() As Long
Return Value
1 if this unit was added to the model as a reference; otherwise 0
C/C++ Prototype
HRESULT isReferenceUnit(long* val)
Return Value
HRESULT (0 for success, or a signed integer error code)
isSeparateSaveUnit
Read method
Description
The isSeparateSaveUnit method determines whether the current unit is saved in its own (separate) file.
Visual Basic
Syntax
isSeparateSaveUnit() As Long
Return Value
1 if this unit is saved to its own file; otherwise 0
C/C++ Prototype
HRESULT isSeparateSaveUnit(long* pVal)
Return Value

HRESULT (0 for success, or a signed integer error code)

load

Write method

Description

The load method loads the specified unit.

Visual Basic

Syntax

load (withSubs As Long) As RPUnit

Argument

withSubs

Set this to 1 to load the unit’s subunits. Otherwise, set this to 0.

Return Value

The loaded unit

C/C++ Prototype

HRESULT load (long withSubs, IRPUnit** ret)

Return Value

HRESULT (0 for success, or a signed integer error code)
save
   Read method

Description
   The `save` method saves the current unit.

Visual Basic
   Syntax
   save (withSubs As Long)
   Argument
       withSubs
       Set this to 1 to load the unit’s subunits. Otherwise, set this to 0.

C/C++ Prototype
   HRESULT save (long withSubs)

Return Value
   HRESULT (0 for success, or a signed integer error code)

setReadOnly
   Write method

Description
   The `setReadOnly` method specifies whether the current unit is read-only.

Visual Basic
   Syntax
   setReadOnly (pVal As Long)
   Arguments
       pVal
       Set this argument to 1 to make the unit read-only; set it to 0 to make
       the unit read/write.

C/C++ Prototype
   HRESULT setReadOnly (long pVal)
**Return Value**

HRESULT (0 for success, or a signed integer error code)

**setSeparateSaveUnit**

**Write method**

**Description**

The `setSeparateSaveUnit` method sets a unit to be stored to its own file.

**Visual Basic**

**Syntax**

```vbnet
setSeparateSaveUnit (pVal As Long)
```

**Arguments**

- **pVal**
  
  Set this argument to 1 to have the unit stored to its own file. Otherwise, set it to 0.

**C/C++ Prototype**

```c
HRESULT setSeparateSaveUnit (long pVal)
```

**Return Value**

HRESULT (0 for success, or a signed integer error code)
The `IRPUseCase` interface represents a Rhapsody use case. It inherits from `IRPClassifier`.

**VB Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>describingDiagrams</td>
<td>Collection of RPSequenceDiagram</td>
<td>RO</td>
<td>A collection of sequence diagrams that describe this use case</td>
</tr>
<tr>
<td>entryPoints</td>
<td>Collection of strings</td>
<td>RO</td>
<td>A collection of entry points into this use case</td>
</tr>
<tr>
<td>extensionPoints</td>
<td>RPCollection</td>
<td>RO</td>
<td>A collection of extension points</td>
</tr>
</tbody>
</table>

**Method Summary**

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addDescribingDiagram</td>
<td>Adds a describing diagram for the current use case</td>
</tr>
<tr>
<td>addExtensionPoint</td>
<td>Adds an extension point to the current use case</td>
</tr>
<tr>
<td>deleteDescribingDiagram</td>
<td>Deletes the describing use case or sequence diagram for the current use case</td>
</tr>
<tr>
<td>deleteEntryPoint</td>
<td>Deletes the entry point of the current use case</td>
</tr>
<tr>
<td>deleteExtensionPoint</td>
<td>Deletes the specified extension point</td>
</tr>
<tr>
<td>findEntryPoint</td>
<td>Deletes the specified entry point</td>
</tr>
<tr>
<td>findExtensionPoint</td>
<td>Retrieves the extension point, given the generalization</td>
</tr>
<tr>
<td>getDescribingDiagram</td>
<td>Retrieves the use case diagram or sequence diagram linked to the current use case</td>
</tr>
</tbody>
</table>
addDescribingDiagram

Write method

Description

The `addDescribingDiagram` method adds a describing diagram for the current use case.

Visual Basic

Syntax

`addDescribingDiagram (diagram As RPDiagram)`

Arguments

`diagram`

The name for the new, describing diagram

C/C++ Prototype

`HRESULT addDescribingDiagram (IRPDiagram* diagram)`

Return Value

`HRESULT (0 for success, or a signed integer error code)`

addExtensionPoint

Write method

Description

The `addExtensionPoint` method adds an extension point to the current use case.

Visual Basic

Syntax

`addExtensionPoint (entryPoint As String)`

Arguments

`entryPoint`

The name of the new entry point

C/C++ Prototype

`HRESULT addExtensionPoint (BSTR entryPoint)`
Return Value

HRESULT (0 for success, or a signed integer error code)

deleteDescribingDiagram

Write method

Description

The deleteDescribingDiagram method deletes the describing use case or sequence diagram for the current use case.

Visual Basic

Syntax

deleteDescribingDiagram (diagram As RPDiagram)

Arguments

diagram

The use case or sequence diagram that describes the current use case

C/C++ Prototype

HRESULT deleteDescribingDiagram (IRPDiagram* diagram)

Return Value

HRESULT (0 for success, or a signed integer error code)
**deleteEntryPoint**

Write method

Description

The `deleteEntryPoint` method deletes the entry point of the current use case.

Visual Basic

Syntax

deleteEntryPoint (entryPoint As String)

Arguments

- `entryPoint`  
The name of the entry point to delete

C/C++ Prototype

HRESULT deleteEntryPoint (BSTR entryPoint)

Return Value

HRESULT (0 for success, or a signed integer error code)

**deleteExtensionPoint**

Write method

Description

The `deleteExtensionPoint` method deletes the specified extension point.

Visual Basic

Syntax

deleteExtensionPoint (point As String)

Arguments

- `point`  
The extension point to delete

C/C++ Prototype

HRESULT deleteExtensionPoint (BSTR entrypoint)
Return Value

HRESULT (0 for success, or a signed integer error code)

**findEntryPoint**

**Read method**

The **findEntryPoint** method returns the specified entry point of the current use case, given the generalization.

**Visual Basic**

**Syntax**

```vba
findEntryPoint (gen As RPGeneralization) As String
```

**Arguments**

- `gen`  
  The generalization

**Return Value**

The entry point

**C/C++ Prototype**

```c
HRESULT findEntryPoint (IRPGeneralization* gen, 
BSTR *pVal)
```

**Return Value**

HRESULT (0 for success, or a signed integer error code)
findExtensionPoint

Read method

The findExtensionPoint method returns the specified extension point of the current use case, given the generalization.

Visual Basic

Syntax

findExtensionPoint (gen As RPGeneralization) As String

Arguments

gen
The generalization

Return Value

The extension point

C/C++ Prototype

HRESULT findExtensionPoint (IRPGeneralization* gen,
BSTR *pVal)

Return Value

HRESULT (0 for success, or a signed integer error code)

getDescribingDiagram

Read method

Description

The getDescribingDiagram method retrieves the use case diagram or sequence diagram linked to the current use case.

Visual Basic

Syntax

getDescribingDiagram (name As String) As RPDiagram

Arguments

name
The name of the use case diagram or sequence diagram that is linked (for descriptive purposes) to the current use case
Return Value

The diagram of the specified use case

C/C++ Prototype

HRESULT getDescribingDiagram (BSTR name, IRPDiagram** diagram)

Return Value

HRESULT (0 for success, or a signed integer error code)
**IRPUseCaseDiagram Interface**

The IRPUseCaseDiagram interface represents a use case diagram. It inherits from IRPDiagram.

Currently, it does not expose additional functionality to IRPDiagram.

**IRPInternalOEMPlugin**

This interface is used for internal purposes only.

**IRPVariable Interface**

The IRPVariable interface represents a variable in a Rhapsody 6.1 model. It represents the UML TypedElement.

IRPVariable inherits from IRPModelElement.

**VB Properties**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Access</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>declaration</td>
<td>String</td>
<td>RW</td>
<td>The declaration statement for the variable</td>
</tr>
<tr>
<td>defaultValue</td>
<td>String</td>
<td>RW</td>
<td>The default value for the variable</td>
</tr>
<tr>
<td>type</td>
<td>RPClassifier</td>
<td>RW</td>
<td>The data type of the variable</td>
</tr>
</tbody>
</table>

**Method Summary**

*setTypeDeclaration* Updates the type declaration for the current attribute
**setTypeDeclaration**

Write method

**Description**

The `setTypeDeclaration` method updates the type declaration for the current attribute.

**Visual Basic**

**Syntax**

```vbnet
setTypeDeclaration (newVal As String)
```

**Arguments**

- **newVal**
  
  The type declaration for this attribute

**C/C++ Prototype**

```c
HRESULT setTypeDeclaration (BSTR newVal)
```

**Return Value**

- HRESULT (0 for success, or a signed integer error code)
The Callback API consists of a number of methods that can be used to respond to events that occur in Rhapsody. This response can consist of actions taken by an external application and/or preventing Rhapsody from proceeding with a specific action.

Callback API Introduction

The Callback API is implemented as a number of COM connection point interfaces. These callback methods can be used by:

- client applications using the Rhapsody COM or Java APIs, in the following languages:
  - VB
  - VBA
  - C++
  - Java
- client plug-ins to Rhapsody

For the methods that have boolean return values, the client application can return a value of True in order to prevent Rhapsody from proceeding with the action connected to the event, for example, preventing a diagram from being opened.

Clients can receive event notification by registering the corresponding COM connection point interface using the standard COM mechanism.

Multiple clients can register for any given callback, however, there is no guarantee that the clients will be notified in a specific order.

In cases where multiple clients have registered, if one client responds by cancelling the associated Rhapsody action, the remaining clients will not be notified of the event.

Rhapsody can log all callbacks invoked. For Rhapsody actions that can be cancelled by clients, it also logs the action taken. For details on enabling logging, see Callback Logging.
The Callback API

Callback notification can be disabled completely, or for specific interfaces by adding appropriate entries to the rhapsody.ini file. For details on complete or partial disabling of callback notification, see Disabling Callback Notification.

When callback notification is enabled, you have the option of disabling the ability of a client application to prevent Rhapsody from proceeding with an action. This can be done for all cancellable actions or just for specific cancellable actions. For details, see Disabling Cancellable Actions.

Events with Callback Methods

The Rhapsody API includes callback methods for the following Rhapsody events:

- project about to be closed
- project closed
- feature dialog about to be opened
- diagram about to be opened
- Rhapsody about to perform roundtrip
- code generation completed

**Note**

These events can only be responded to by using the Rhapsody API. They are not available as triggers in the Helpers dialog (Tools > Customize).
API Details

This section lists individual Rhapsody APIs with the API format and a description of its uses.

IRPApplicationListener

The IRPApplicationListener API is called before and after Rhapsody closes a project.

BeforeProjectClose

BOOL BeforeProjectClose(IRPProject Project)

This is called before a project is closed. The argument is the project that is to be closed.

If a client returns True, then the project will not be closed.

Points to take into consideration:

- If a client returns True to prevent the closing of the project, other clients that have registered will not be notified of the event.
- When multiple projects are to be closed, the method is called separately for each project. If a client prevents the closing of a specific project, this does not affect the calling of the method for the remaining projects.

AfterProjectClose

void AfterProjectClose(BSTR ProjectName)

This is called after Rhapsody closes a project. The argument is the name of the project that was closed.

Points to take into consideration:

- When multiple projects are closed, the method is called separately for each project.
- This method is not available for VBA clients. (This is because the VBA application is part of the Rhapsody project so it cannot be run after the project is closed.)

OnDiagramOpen

BOOL OnDiagramOpen(IRPDiagram Diagram)

This is called before a diagram is opened. The argument is the diagram that Rhapsody is about to open.

If a client returns True, then the diagram will not be opened.
Points to take into consideration:

- If a client returns *True* to prevent the opening of the diagram, other clients that have registered will not be notified of the event.
- The method is only called when a diagram is explicitly opened using the Rhapsody GUI or the Rhapsody API. It is not called when a diagram is opened as part of the restoration of the previous Rhapsody workspace.

**OnFeaturesOpen**

`BOOL OnFeaturesOpen(IRPModelElement ModelElement)`

This is called before the Features dialog is opened for a given element. The argument is the model element for which the Features dialog is going to be opened.

If a client returns *True*, then the Features dialog will not be opened for the element.

Points to take into consideration:

- If a client returns *True* to prevent the opening of the Features dialog, other clients that have registered will not be notified of the event.
- The method is only called when the Features dialog is explicitly opened using the Rhapsody GUI or the Rhapsody API. It is not called when the Features dialog is opened as part of the restoration of the previous Rhapsody workspace.
**IRPRoundTripListener**

The IRPRoundTripListener is called before the source code files are roundtripped into the model.

**BeforeRoundtrip**

```java
void BeforeRoundtrip(IRPCollection fileNames)
```

This is called before source code files are roundtripped into the model.

The argument consists of the files that are going to be roundtripped into the model.

**IRPCodeGeneratorListener**

The IRPCodeGeneratorListener is called after code generation.

**CodeGenerationCompleted**

```java
void CodeGenerationCompleted()
```

This is called after code generation has been completed.

Points to take into consideration:

- Clients should not modify generated code files in the framework of the callback method. This will result in timestamp inconsistency in the model-generated code, creating potential problems.
The Callback API

Callback Logging

By default, Rhapsody does not maintain a log file of callback events. To enable logging of callback events and cancellable actions, add the entry EnableCallbackLogging to a section called [Callback] in the rhapsody.ini file and set it to TRUE.

If you enable logging, the events and actions will be logged to a file called callback_log.txt in the system temporary directory.

Disabling Callback Notification

Callback functionality can be disabled completely, or for specific interfaces by adding one or more of the following entries to a section called [Callback] in the rhapsody.ini file:

To disable the callback mechanism for project closing, opening diagrams, and opening the Features dialog, add the entry EnableApplicationEventListening and set it to FALSE. Rhapsody will not notify registered clients of these events.

To disable the callback mechanism for roundtripping, add the entry EnableRoundTripEventListening and set it to FALSE. Rhapsody will not notify registered clients of roundtripping events.

To disable the callback mechanism for code generation, add the entry EnableCodeGenerationEventListening and set it to FALSE. Rhapsody will not notify registered clients of code generation events.

To disable the callback mechanism completely, add the entry EnableEventListening and set it to FALSE. Rhapsody will not notify registered clients of any of the callback events.

Disabling Cancellable Actions

When callback notification is enabled, you can disable the ability of a client application to prevent Rhapsody from proceeding with an action by adding one or more of the following entries to a section called [Callback] in the rhapsody.ini file:

To disable the ability to prevent Rhapsody from closing a project, add the entry CanCancelProjectClose and set it to FALSE.

To disable the ability to prevent Rhapsody from opening a diagram, add the entry CanCancelOpenDiagram and set it to FALSE.

To disable the ability to prevent Rhapsody from opening the Features dialog for an element, add the entry CanCancelOpenFeaturesDialog and set it to FALSE.
If you don’t want to allow clients to prevent any of the cancellable actions, add the entry `CanCancelAction` and set it to `FALSE`.

**Sample Client Applications**

Sample client applications that use the callback API to respond to Rhapsody events can be found in the Rhapsody samples directory (`[installation directory]\Samples\ExtensibilitySamples\CallbackAPISamples`).

The samples provided are written in a number of different languages.
Quick Reference

This section lists the Rhapsody API methods and provides a brief description of each. For ease of use, the methods are presented in alphabetical order.

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abort</td>
<td>Is invoked when the user selects the Abort option during code generation</td>
</tr>
<tr>
<td>activeProject</td>
<td>Returns a pointer to the active (open) project</td>
</tr>
<tr>
<td>addActivityDiagram</td>
<td>Adds an activity diagram to the current class</td>
</tr>
<tr>
<td>addActor</td>
<td>Adds the specified actor to the current package</td>
</tr>
<tr>
<td>addAnchor</td>
<td>Adds an anchor from the annotation to the specified model element</td>
</tr>
<tr>
<td>addArgument</td>
<td>Adds an argument for the operation to the end of its argument list</td>
</tr>
<tr>
<td>addArgumentBeforePosition</td>
<td>Adds an argument for the operation at the specified position in its argument list</td>
</tr>
<tr>
<td>addAttribute</td>
<td>Adds an attribute to the current class</td>
</tr>
<tr>
<td>addBlock</td>
<td>Adds a block to the current package</td>
</tr>
<tr>
<td>addClass</td>
<td>Adds a class to the current class</td>
</tr>
<tr>
<td>addClassifierRole</td>
<td>Adds a classifier role</td>
</tr>
<tr>
<td>addClassifierRoleByName</td>
<td>Adds a classifier role, given its name</td>
</tr>
<tr>
<td>addCollaborationDiagram</td>
<td>Adds a collaboration diagram to the current package</td>
</tr>
<tr>
<td>addComponent</td>
<td>Adds the specified component to the current project</td>
</tr>
<tr>
<td>addComponentDiagram</td>
<td>Adds a component diagram to the current package</td>
</tr>
<tr>
<td>addComponentInstance</td>
<td>Adds a new component instance</td>
</tr>
<tr>
<td>addConfiguration</td>
<td>Adds a configuration to this component</td>
</tr>
<tr>
<td>addConnector</td>
<td>Adds a connector to the statechart</td>
</tr>
<tr>
<td>addConstructor</td>
<td>Adds a constructor to the current class</td>
</tr>
<tr>
<td>Method Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>addConveyed</td>
<td>Adds an information element to the conveyed collection</td>
</tr>
<tr>
<td>addCtor</td>
<td>Adds a constructor</td>
</tr>
<tr>
<td>addDependency</td>
<td>Adds a dependency relationship to the specified object</td>
</tr>
<tr>
<td>addDependencyTo</td>
<td>Creates a new dependency between two objects</td>
</tr>
<tr>
<td>addDeploymentDiagram</td>
<td>Adds the specified deployment diagram to the current package</td>
</tr>
<tr>
<td>addDescribingDiagram</td>
<td>Adds a describing diagram for the current use case</td>
</tr>
<tr>
<td>addDestructor</td>
<td>Adds a destructor to the current class</td>
</tr>
<tr>
<td>addDtor</td>
<td>Adds a destructor</td>
</tr>
<tr>
<td>addElement</td>
<td>Adds an element to the current file</td>
</tr>
<tr>
<td>addEnumerationLiteral</td>
<td>Creates an enumeration literal</td>
</tr>
<tr>
<td>addEvent</td>
<td>Adds the specified event to the current package</td>
</tr>
<tr>
<td>addEventReception</td>
<td>Adds an event reception to the current class</td>
</tr>
<tr>
<td>addExtensionPoint</td>
<td>Adds an extension point to the current use case</td>
</tr>
<tr>
<td>addFile</td>
<td>Adds an empty file to the current component</td>
</tr>
<tr>
<td>addFlowItems</td>
<td>Adds the specified flowItem to the collection of flowItems</td>
</tr>
<tr>
<td>addFlows</td>
<td>Adds the specified flow to the collection of flows</td>
</tr>
<tr>
<td>addFolder</td>
<td>Adds an empty folder to the current component</td>
</tr>
<tr>
<td>addGeneralization</td>
<td>Adds a generalization to the current class</td>
</tr>
<tr>
<td>addGlobalFunction</td>
<td>Adds the specified global function to this package</td>
</tr>
<tr>
<td>addGlobalObject</td>
<td>Adds a global object (instance) to the current package</td>
</tr>
<tr>
<td>addGlobalVariable</td>
<td>Adds the specified global variable to the current package</td>
</tr>
<tr>
<td>addInitialInstance</td>
<td>Adds an instance to the list of initial instances for the current configuration</td>
</tr>
<tr>
<td>addItem</td>
<td>Adds an item to the collection</td>
</tr>
<tr>
<td>addMessage</td>
<td>Adds a message</td>
</tr>
<tr>
<td>addNestedComponent</td>
<td>Adds a component to the current component</td>
</tr>
<tr>
<td>addNestedPackage</td>
<td>Adds a nested package to the current package</td>
</tr>
<tr>
<td>addNewAggr</td>
<td>Used to add a new element to the current element, for example, adding a new class to a package</td>
</tr>
<tr>
<td>addNode</td>
<td>Adds the specified node to the current package</td>
</tr>
<tr>
<td>Method Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>addObjectModelDiagram</td>
<td>Adds the specified OMD to the current package</td>
</tr>
<tr>
<td>addOperation</td>
<td>Adds an operation to the current class</td>
</tr>
<tr>
<td>addPackage</td>
<td>Adds the specified package to the current project</td>
</tr>
<tr>
<td>addPackageToInstrumentationScope</td>
<td>Adds the specified package to the instrumentation scope, including all its aggregated classes, actors, and nested packages</td>
</tr>
<tr>
<td>addPackageToScope</td>
<td>Adds the specified package to the scope of the file or folder</td>
</tr>
<tr>
<td>addProperty</td>
<td>Adds a new property/value pair for the current element</td>
</tr>
<tr>
<td>addProvidedInterface</td>
<td>Adds the specified interface to the collection of provided interfaces</td>
</tr>
<tr>
<td>addReferenceActivity</td>
<td>Adds a reference activity to the activity diagram</td>
</tr>
<tr>
<td>addRelation</td>
<td>Adds a symmetric relation between the current class and another one</td>
</tr>
<tr>
<td>addRepresented</td>
<td>Adds a flowItem to the represented collection</td>
</tr>
<tr>
<td>addRequiredInterface</td>
<td>Adds the specified interface to the collection of required interfaces</td>
</tr>
<tr>
<td>addScopeElement</td>
<td>Places a model element within the scope of the current component</td>
</tr>
<tr>
<td>addSequenceDiagram</td>
<td>Adds the specified sequence diagram to the current package</td>
</tr>
<tr>
<td>addState</td>
<td>Adds a state to the statechart</td>
</tr>
<tr>
<td>addStatechart</td>
<td>Adds a statechart to the current class</td>
</tr>
<tr>
<td>addStaticReaction</td>
<td>Adds a static reaction to the statechart</td>
</tr>
<tr>
<td>addStereotype</td>
<td>Adds a stereotype relationship to the specified object</td>
</tr>
<tr>
<td>addSuperclass</td>
<td>Adds a superclass to the current class</td>
</tr>
<tr>
<td>addSwimlane</td>
<td>Adds a swimlane to the activity diagram</td>
</tr>
<tr>
<td>addSystemBorder</td>
<td>Adds a system border to the collaboration diagram</td>
</tr>
<tr>
<td>addTerminationState</td>
<td>Adds a termination state to the statechart</td>
</tr>
<tr>
<td>addTextElement</td>
<td>Adds text to the file</td>
</tr>
<tr>
<td>addTimeInterval</td>
<td>Adds a time interval to the diagram</td>
</tr>
<tr>
<td>addTimeout</td>
<td>Adds a timeout</td>
</tr>
<tr>
<td>addToInstrumentationScope</td>
<td>Adds explicit initial instances to the instrumentation scope</td>
</tr>
</tbody>
</table>
### Method Name | Description
--- | ---
`addToModel` | Adds a Rhapsody unit located in the specified file to the current model with or without descendant elements
`addToModelFromURL` | Adds a Rhapsody unit located at the specified URL to the current model
`addToScope` | Places the specified file, classes, and packages within the scope of the current component
`addTransition` | Creates a transition
`addTriggeredOperation` | Adds a triggered operation to the current class
`addType` | Adds a type to the current class
`addUnidirectionalRelation` | Adds a directional relation from the current class to another class
`addUseCase` | Adds the specified use case to the current package
`addUseCaseDiagram` | Adds the specified UCD to the current package
`allElementsInScope` | Places all model elements within the scope of the current component
`arcCheckOut` | Checks out files from the CM archive into the model
`becomeTemplateInstantiationOf` | Creates a template instantiation of another template (of another template class)
`build` | Builds the application
`checkEventsBaseIdsSolveCollisions` | Checks the values of the events base IDs for all packages in the model, detects collisions between the IDs, and resolves any incorrect values and collisions
`checkIn` | Checks in the specified unit within the model into the CM archive you have already connected to (using `connectToArchive`)
`checkModel` | Checks the current model
`checkOut` | Refreshes a unit in the model by checking it out from the CM archive
`clone` | Clones the element, names it, and adds it to the new owner
`close` | Closes a file or project
`connectToArchive` | Connects the Rhapsody project to the specified CM archive
`createDefaultTransition` | Creates a default transition in the statechart
`createGraphics` | Creates graphics in the Rhapsody statechart
`createNestedStatechart` | Creates a nested statechart
<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>createNewProject</code></td>
<td>Creates a new project named <code>&lt;projectName&gt;</code> in <code>&lt;projectLocation&gt;</code></td>
</tr>
<tr>
<td><code>deleteActivityDiagram</code></td>
<td>Deletes the specified activity diagram from the current class</td>
</tr>
<tr>
<td><code>deleteActor</code></td>
<td>Deletes the specified actor from the current package</td>
</tr>
<tr>
<td><code>deleteArgument</code></td>
<td>Deletes an argument from the current operation</td>
</tr>
<tr>
<td><code>deleteAttribute</code></td>
<td>Deletes the specified attribute from the current class</td>
</tr>
<tr>
<td><code>deleteClass</code></td>
<td>Deletes a class from the current class</td>
</tr>
<tr>
<td><code>deleteCollaborationDiagram</code></td>
<td>Deletes the specified collaboration diagram from the current package</td>
</tr>
<tr>
<td><code>deleteComponent</code></td>
<td>Deletes the specified component from the current project</td>
</tr>
<tr>
<td><code>deleteComponentDiagram</code></td>
<td>Deletes the specified component diagram from the current package</td>
</tr>
<tr>
<td><code>deleteComponentInstance</code></td>
<td>Deletes the specified component instance</td>
</tr>
<tr>
<td><code>deleteConfiguration</code></td>
<td>Deletes the specified configuration from the current component</td>
</tr>
<tr>
<td><code>deleteConnector</code></td>
<td>Deletes the specified connector from the statechart</td>
</tr>
<tr>
<td><code>deleteConstructor</code></td>
<td>Deletes a constructor from the current class</td>
</tr>
<tr>
<td><code>deleteDependency</code></td>
<td>Deletes a dependency</td>
</tr>
<tr>
<td><code>deleteDeploymentDiagram</code></td>
<td>Deletes the specified deployment diagram from the current package</td>
</tr>
<tr>
<td><code>deleteDescribingDiagram</code></td>
<td>Deletes the describing use case or sequence diagram for the current use case</td>
</tr>
<tr>
<td><code>deleteDestructor</code></td>
<td>Deletes a destructor from the current class</td>
</tr>
<tr>
<td><code>deleteEntryPoint</code></td>
<td>Deletes the entry point of the current use case</td>
</tr>
<tr>
<td><code>deleteEvent</code></td>
<td>Deletes the specified event from the current package</td>
</tr>
<tr>
<td><code>deleteEventReception</code></td>
<td>Deletes the specified event reception from the current class</td>
</tr>
<tr>
<td><code>deleteExtensionPoint</code></td>
<td>Deletes the specified extension point</td>
</tr>
<tr>
<td><code>deleteFile</code></td>
<td>Deletes the specified file from the current component</td>
</tr>
<tr>
<td><code>deleteFlowchart</code></td>
<td>Deletes an activity diagram from the current operation</td>
</tr>
<tr>
<td><code>deleteFlowItems</code></td>
<td>Deletes the specified flowItem from the collection of flowItems</td>
</tr>
<tr>
<td><code>deleteFlows</code></td>
<td>Deletes the specified flow from the collection of flows</td>
</tr>
<tr>
<td>Method Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>deleteFromProject</td>
<td>Deletes the current model element from the project open in Rhapsody</td>
</tr>
<tr>
<td>deleteGeneralization</td>
<td>Deletes the specified generalization from the current class</td>
</tr>
<tr>
<td>deleteGlobalFunction</td>
<td>Deletes the specified global function from the current package</td>
</tr>
<tr>
<td>deleteGlobalObject</td>
<td>Deletes the specified global object from the current package</td>
</tr>
<tr>
<td>deleteGlobalVariable</td>
<td>Deletes the specified global variable from the current package</td>
</tr>
<tr>
<td>deleteInitialInstance</td>
<td>Deletes an instance from the list of build instances for the current</td>
</tr>
<tr>
<td></td>
<td>configuration</td>
</tr>
<tr>
<td>deleteNode</td>
<td>Deletes the specified node from the current package</td>
</tr>
<tr>
<td>deleteObjectModelDiagram</td>
<td>Deletes the specified OMD from the current package</td>
</tr>
<tr>
<td>deleteOperation</td>
<td>Deletes the specified operation from the current class</td>
</tr>
<tr>
<td>deletePackage</td>
<td>Deletes the current package</td>
</tr>
<tr>
<td>deleteRelation</td>
<td>Deletes the specified relation from the current class</td>
</tr>
<tr>
<td>deleteSequenceDiagram</td>
<td>Deletes the specified sequence diagram from the current package</td>
</tr>
<tr>
<td>deleteState</td>
<td>Deletes the specified state from the Rhapsody statechart</td>
</tr>
<tr>
<td>deleteStatechart</td>
<td>Deletes the specified statechart from the current class</td>
</tr>
<tr>
<td>deleteStaticReaction</td>
<td>Deletes the specified static reaction from the statechart</td>
</tr>
<tr>
<td>deleteSuperclass</td>
<td>Deletes a superclass from the current class</td>
</tr>
<tr>
<td>deleteTransition</td>
<td>Deletes a transition</td>
</tr>
<tr>
<td>deleteType</td>
<td>Deletes a type from the current class</td>
</tr>
<tr>
<td>deleteUseCase</td>
<td>Deletes the specified use case from the current package</td>
</tr>
<tr>
<td>deleteUseCaseDiagram</td>
<td>Deletes the specified use case diagram from the current package</td>
</tr>
<tr>
<td>enterAnimationCommand</td>
<td>Specifies the command to begin animation</td>
</tr>
<tr>
<td>errorMessage</td>
<td>Returns the most recent error message</td>
</tr>
<tr>
<td>Exit</td>
<td>Is invoked before Rhapsody exits</td>
</tr>
<tr>
<td>findActor</td>
<td>Retrieves the specified actor, if it belongs to the current package</td>
</tr>
<tr>
<td>findAllByName</td>
<td>Searches all the elements and finds the first element of the specified</td>
</tr>
<tr>
<td></td>
<td>name and metaclass in the current package</td>
</tr>
<tr>
<td>Method Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>findAttribute</td>
<td>Retrieves the specified attribute of the classifier</td>
</tr>
<tr>
<td>findBaseClassifier</td>
<td>Retrieves a base (parent) classifier of a classifier</td>
</tr>
<tr>
<td>findClass</td>
<td>Retrieves the specified class, if it belongs to the current package</td>
</tr>
<tr>
<td>findComponent</td>
<td>Retrieves the specified component from the current project</td>
</tr>
<tr>
<td>findComponentInstance</td>
<td>Retrieves the specified component instance</td>
</tr>
<tr>
<td>findConfiguration</td>
<td>Retrieves the specified configuration in the current component</td>
</tr>
<tr>
<td>findDerivedClassifier</td>
<td>Retrieves the specified derived classifier of a classifier</td>
</tr>
<tr>
<td>findElementsByFullName</td>
<td>Searches for the specified element</td>
</tr>
<tr>
<td>findEntryPoint</td>
<td>Deletes the specified entry point</td>
</tr>
<tr>
<td>findEvent</td>
<td>Retrieves the specified event, if it belongs to the current package</td>
</tr>
<tr>
<td>findExtensionPoint</td>
<td>Retrieves the extension point, given the generalization</td>
</tr>
<tr>
<td>findGeneralization</td>
<td>Retrieves the specified generalization of a classifier</td>
</tr>
<tr>
<td>findGlobalFunction</td>
<td>Retrieves the specified global function, if it belongs to the current package</td>
</tr>
<tr>
<td>findGlobalObject</td>
<td>Retrieves the specified global object, if it belongs to the current package</td>
</tr>
<tr>
<td>findGlobalVariable</td>
<td>Retrieves the specified global variable, if it belongs to the current package</td>
</tr>
<tr>
<td>findInterfaceItem</td>
<td>Retrieves an operation or event reception of the given signature that belongs to a classifier</td>
</tr>
<tr>
<td>findNestedClassifier</td>
<td>Retrieves the specified classifier defined within this object</td>
</tr>
<tr>
<td>findNestedClassifierRecursive</td>
<td>Retrieves the specified classifier defined in this object and in objects defined within this object</td>
</tr>
<tr>
<td>findNestedElement</td>
<td>Retrieves the specified element nested in a model element</td>
</tr>
<tr>
<td>findNestedElementRecursive</td>
<td>Retrieves the specified element from a given model element at any level of nesting within that element</td>
</tr>
<tr>
<td>findNode</td>
<td>Retrieves the specified node, if it belongs to the current package</td>
</tr>
<tr>
<td>findRelation</td>
<td>Retrieves the specified relation that belongs to the current classifier</td>
</tr>
<tr>
<td>findTrigger</td>
<td>Retrieves the specified trigger in the statechart of the current class</td>
</tr>
<tr>
<td>Method Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>findType</code></td>
<td>Retrieves the specified data type, if it belongs to the current package</td>
</tr>
<tr>
<td><code>findUsage</code></td>
<td>Retrieves the usage of the specified element in the current package</td>
</tr>
<tr>
<td><code>findUseCase</code></td>
<td>Retrieves the specified use case, if it belongs to the current package</td>
</tr>
<tr>
<td><code>forceRoundtrip</code></td>
<td>Forcers a roundtrip of the code back into the Rhapsody model, and vice versa</td>
</tr>
<tr>
<td><code>generate</code></td>
<td>Generates code for the active configuration of the active component</td>
</tr>
<tr>
<td><code>generateSequence</code></td>
<td>Generates the specified sequence diagram</td>
</tr>
<tr>
<td><code>getConcurrentGroup</code></td>
<td>Retrieves the activation messages</td>
</tr>
<tr>
<td><code>getAllGraphicalProperties</code></td>
<td>Returns the list of graphical properties for the current diagram</td>
</tr>
<tr>
<td><code>getAllTriggers</code></td>
<td>Returns a collection of all the triggers for the current statechart</td>
</tr>
<tr>
<td><code>getAttributesIncludingBases</code></td>
<td>Retrieves the attributes defined for this class and the ones inherited from its superclasses</td>
</tr>
<tr>
<td><code>getClassifierRole</code></td>
<td>Retrieves the classifier role for this message point</td>
</tr>
<tr>
<td><code>getClassifierRoles</code></td>
<td>Returns a collection of <code>IRPClassifierRoles</code> linked by the current association role</td>
</tr>
<tr>
<td><code>getConcurrentGroup</code></td>
<td>Retrieves all the messages concurrent with the input message, including the input message itself</td>
</tr>
<tr>
<td><code>getDerivedInEdges</code></td>
<td>Retrieves the incoming transitions for the connector</td>
</tr>
<tr>
<td><code>getDerivedOutEdge</code></td>
<td>Retrieves the incoming transitions for the connector</td>
</tr>
<tr>
<td><code>getDescribingDiagram</code></td>
<td>Retrieves the use case diagram or sequence diagram linked to the current use case</td>
</tr>
<tr>
<td><code>getDiagramOfSelectedElement</code></td>
<td>Retrieves the diagram of the current element</td>
</tr>
<tr>
<td><code>getDirectory</code></td>
<td>Retrieves the build directory specified for the current configuration</td>
</tr>
<tr>
<td><code>getElementsInDiagram</code></td>
<td>Returns a collection of all the model elements in the current diagram</td>
</tr>
<tr>
<td><code>getErrorMessage</code></td>
<td>Returns the most recent error message</td>
</tr>
<tr>
<td><code>getEvent</code></td>
<td>Returns the event for the current event reception that serves as part of the interface for a class</td>
</tr>
<tr>
<td><code>getFile</code></td>
<td>Returns the file in which the specified classifier will be generated</td>
</tr>
<tr>
<td>Method Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>getFileName</td>
<td>Retrieves the name of the file to which the specified classifier will be generated in this component</td>
</tr>
<tr>
<td>getFormalRelations</td>
<td>Returns a collection of <code>IRPRelations</code> for the current association role</td>
</tr>
<tr>
<td>getFullNameInStatechart</td>
<td>Returns the full text name of this state within its statecharts</td>
</tr>
<tr>
<td>getFullPathName</td>
<td>Retrieves the full path name of a model element as a string</td>
</tr>
<tr>
<td>getFullPathNameIn</td>
<td>Retrieves the full path name of a model element as a string</td>
</tr>
<tr>
<td>getGraphicalProperty</td>
<td>Returns the specified graphical property for the current diagram</td>
</tr>
<tr>
<td>getImpName</td>
<td>Retrieves the name of the current file's implementation file, including its extension and, if specified, its relative path</td>
</tr>
<tr>
<td>getInheritsFrom</td>
<td>Returns the base state from which the current state inherits</td>
</tr>
<tr>
<td>getInLinks</td>
<td>Returns the list of links for which the instance is the target instance (identified by the &quot;to&quot; property of the link)</td>
</tr>
<tr>
<td>getInterfaceItemsIncludingBases</td>
<td>Retrieves the operations and event receptions defined for this class and the ones it inherited from its superclasses</td>
</tr>
<tr>
<td>getInTransitions</td>
<td>Returns a collection of transitions that are directed into the current state or connector</td>
</tr>
<tr>
<td>getItsAction</td>
<td>Returns the action code of the current transition</td>
</tr>
<tr>
<td>getItsComponent</td>
<td>Retrieves the component to which the current configuration belongs</td>
</tr>
<tr>
<td>getItsGuard</td>
<td>Returns the guard condition of the current transition</td>
</tr>
<tr>
<td>getItsOperation</td>
<td>Returns the event or triggered operation of the current trigger</td>
</tr>
<tr>
<td>getItsTrigger</td>
<td>Returns the trigger (event or triggered operation) of the current transition</td>
</tr>
<tr>
<td>getListOfFactoryProperties</td>
<td>Returns the list of properties in the <code>&lt;lang&gt;_factory.prp</code> file</td>
</tr>
<tr>
<td>getListOfInitializerArguments</td>
<td>Returns the list of arguments for the initializer, as defined by the user in the instance features dialog box</td>
</tr>
<tr>
<td>getListOfSelectedElements</td>
<td>Returns the collection of model elements</td>
</tr>
<tr>
<td>getListOfSiteProperties</td>
<td>Returns the list of properties in the <code>&lt;lang&gt;_site.prp</code> file</td>
</tr>
<tr>
<td>getLogicalCollaboration</td>
<td>Retrieves the logic behind the collaboration diagram</td>
</tr>
<tr>
<td>Method Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>getLogicalStates</code></td>
<td>Retrieves the list of logical states</td>
</tr>
<tr>
<td><code>GetMainFileName</code></td>
<td>Is invoked when Rhapsody needs the main file name and path for a configuration</td>
</tr>
<tr>
<td><code>getName</code></td>
<td>Retrieves the name of the file where the <code>main()</code> routine for the current configuration resides</td>
</tr>
<tr>
<td><code>getMakefileName</code></td>
<td>Retrieves the name of the makefile generated for the current configuration</td>
</tr>
<tr>
<td><code>getMessagePoints</code></td>
<td>Returns an ordered collection of all messagepoints occurring on this classifier</td>
</tr>
<tr>
<td><code>getModelElementFileName</code></td>
<td>Gets the file name of the specified model element</td>
</tr>
<tr>
<td><code>getNestedElements</code></td>
<td>Retrieves the elements defined in the current object</td>
</tr>
<tr>
<td><code>getNestedElementsRecursive</code></td>
<td>Recursively retrieves the elements defined in the model element for the object and for objects defined in it</td>
</tr>
<tr>
<td><code>getNewCollaboration</code></td>
<td>Retrieves the new collaboration for the current project</td>
</tr>
<tr>
<td><code>getOfState</code></td>
<td>Returns the state connected to the current connector if it is a history connector</td>
</tr>
<tr>
<td><code>getOutLinks</code></td>
<td>Returns the list of links for which the instance is the source instance (identified by the “from” property of the link)</td>
</tr>
<tr>
<td><code>getOutTransitions</code></td>
<td>Returns a collection of transitions that are directed out of the current state or connector</td>
</tr>
<tr>
<td><code>getOverriddenProperties</code></td>
<td>Retrieves the list of properties whose default values have been overridden</td>
</tr>
<tr>
<td><code>getPackageFile</code></td>
<td>Returns the package file</td>
</tr>
<tr>
<td><code>getPicture</code></td>
<td>Renders this diagram into the specified extended metafile</td>
</tr>
<tr>
<td><code>getPictureAsDividedMetafiles</code></td>
<td>Enables you to split a large diagram into several metafiles when you export it</td>
</tr>
<tr>
<td><code>getPredecessor</code></td>
<td>Retrieves the message that precedes the specified message</td>
</tr>
<tr>
<td><code>getPropertyValue</code></td>
<td>Returns the value associated with the specified key value</td>
</tr>
<tr>
<td><code>getPropertyValueExplicit</code></td>
<td>Returns an explicit value if it has been assigned to the metamodel</td>
</tr>
<tr>
<td><code>getRelationsIncludingBases</code></td>
<td>Retrieves the relations defined for this class and the ones it inherited from its superclasses</td>
</tr>
<tr>
<td><code>getRelatedUseCases</code></td>
<td>Retrieves use cases related to the current sequence diagram</td>
</tr>
<tr>
<td><code>getSelectedElement</code></td>
<td>Retrieves the current model element</td>
</tr>
<tr>
<td>Method Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>getSignature</td>
<td>Retrieves the prototype of the IRPMessage</td>
</tr>
<tr>
<td>getSignatureNoArgNames</td>
<td>Retrieves the signature of the current class interface element without argument names</td>
</tr>
<tr>
<td>getSignatureNoArgTypes</td>
<td>Retrieves the signature of the current class interface element without argument types</td>
</tr>
<tr>
<td>getSpecName</td>
<td>Retrieves the name of the current file’s specification file, including its extension and, if specified, its relative path</td>
</tr>
<tr>
<td>getStaticReactions</td>
<td>Returns a collection of static reaction transitions originating from the current state</td>
</tr>
<tr>
<td>getSubStates</td>
<td>Returns a collection of substates belonging to the current state</td>
</tr>
<tr>
<td>getSuccessor</td>
<td>Retrieves the message that follows the specified message</td>
</tr>
<tr>
<td>GetTargetfileName</td>
<td>Is invoked when Rhapsody needs the target name and path for a configuration</td>
</tr>
<tr>
<td>getTargetName</td>
<td>Retrieves the build name of the file to be generated for the current configuration</td>
</tr>
<tr>
<td>getTheExternalCodeGeneratorInvoker</td>
<td>Returns the invoker for the external code generator</td>
</tr>
<tr>
<td>highlightByHandle</td>
<td>Highlights an element, given its handle</td>
</tr>
<tr>
<td>highLightElement</td>
<td>Highlights the specified element</td>
</tr>
<tr>
<td>importClasses</td>
<td>Imports classes according to the reverse engineering setting stored in the current configuration</td>
</tr>
<tr>
<td>importPackageFromRose</td>
<td>Imports the specified package from Rational Rose</td>
</tr>
<tr>
<td>importProjectFromRose</td>
<td>Imports the specified project from Rational Rose</td>
</tr>
<tr>
<td>isAnd</td>
<td>Determines whether this state is an And state</td>
</tr>
<tr>
<td>isArray</td>
<td>Determines whether the current type is an array</td>
</tr>
<tr>
<td>isCompound</td>
<td>Determines whether the current state is a compound state</td>
</tr>
<tr>
<td>isConditionConnector</td>
<td>Determines whether the current connector is a condition connector</td>
</tr>
<tr>
<td>isDefaultTransition</td>
<td>Determines whether the current transition is a default transition</td>
</tr>
<tr>
<td>isDiagramConnector</td>
<td>Determines whether the current connector is a diagram connector</td>
</tr>
<tr>
<td>isEmpty</td>
<td>Determines whether the current file is empty</td>
</tr>
<tr>
<td>isEnum</td>
<td>Determines whether the current type is an enumerated type</td>
</tr>
<tr>
<td>Method Name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>isEqualTo</td>
<td>Tests for equality between the type of the type and the type itself</td>
</tr>
<tr>
<td>isForkConnector</td>
<td>Determines whether the current connector is a fork synch bar connector</td>
</tr>
<tr>
<td>isHistoryConnector</td>
<td>Determines whether the current connector is a history connector</td>
</tr>
<tr>
<td>isImplicit</td>
<td>Determines whether the type is an implicit type</td>
</tr>
<tr>
<td>isJoinConnector</td>
<td>Determines whether the current connector is a join synch bar connector</td>
</tr>
<tr>
<td>isJunctionConnector</td>
<td>Determines whether the current connector is a junction connector</td>
</tr>
<tr>
<td>isKindEnumeration</td>
<td>Determines whether the type is an enumeration</td>
</tr>
<tr>
<td>isKindLanguage</td>
<td>Determines whether the type is a language declaration type</td>
</tr>
<tr>
<td>isKindStructure</td>
<td>Determines whether the type is a structure</td>
</tr>
<tr>
<td>isKindTypedef</td>
<td>Determines whether the type is a typedef</td>
</tr>
<tr>
<td>isKindUnion</td>
<td>Determines whether the type is a union</td>
</tr>
<tr>
<td>isLeaf</td>
<td>Determines whether the current state is a leaf state</td>
</tr>
<tr>
<td>isOperation</td>
<td>Determines whether the current trigger is an operation (event or triggered operation)</td>
</tr>
<tr>
<td>isPointer</td>
<td>Determines whether the current type is a pointer</td>
</tr>
<tr>
<td>isPointerToPointer</td>
<td>Determines whether the current type is a pointer to another pointer</td>
</tr>
<tr>
<td>isReadOnly</td>
<td>Determines whether the current unit is read-only</td>
</tr>
<tr>
<td>isReference</td>
<td>Determines whether the current type is a reference</td>
</tr>
<tr>
<td>isReferenceToPointer</td>
<td>Determines whether the current type is a reference to a pointer</td>
</tr>
<tr>
<td>isRoot</td>
<td>Determines whether the current state is a root state</td>
</tr>
<tr>
<td>isSeparateSaveUnit</td>
<td>Determines whether the current unit is saved in its own (separate) file</td>
</tr>
<tr>
<td>isStaticReaction</td>
<td>Determines whether this is a static reaction</td>
</tr>
<tr>
<td>isStruct</td>
<td>Determines whether the current type is a struct</td>
</tr>
<tr>
<td>isStubConnector</td>
<td>Determines whether the current connector is a stub connector</td>
</tr>
<tr>
<td>isTemplate</td>
<td>Determines whether the current type is a template</td>
</tr>
<tr>
<td>Method Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>isTerminationConnector</td>
<td>Determines whether the current connector is a termination connector</td>
</tr>
<tr>
<td>isTimeout</td>
<td>Determines whether the current trigger is a timeout</td>
</tr>
<tr>
<td>isTypelessObject</td>
<td>Tests an object to see if it is defined explicitly or implicitly</td>
</tr>
<tr>
<td>isUnion</td>
<td>Determines whether the current type is a union</td>
</tr>
<tr>
<td>itsCompoundSource</td>
<td>Returns a collection of states that act as multiple sources for this single transition</td>
</tr>
<tr>
<td>load</td>
<td>Loads the specified unit</td>
</tr>
<tr>
<td>make</td>
<td>Builds the current component following the current configuration</td>
</tr>
<tr>
<td>makeUnidirect</td>
<td>Changes the current relation from a unidirectional (symmetric) one to one that is directional from the me of this relation to me's inverse</td>
</tr>
<tr>
<td>matchOnSignature</td>
<td>Determines whether the signature of the current class interface element matches that of another IRPInterfaceItem</td>
</tr>
<tr>
<td>notifyGenerationDone</td>
<td>Is called by the external code generator after a generation session invoked by the generate event is done</td>
</tr>
<tr>
<td>open</td>
<td>Opens a file</td>
</tr>
<tr>
<td>openProject</td>
<td>Opens a Rhapsody project</td>
</tr>
<tr>
<td>openProjectFromURL</td>
<td>Opens the Rhapsody product at the specified URL</td>
</tr>
<tr>
<td>openProjectWithLastSession</td>
<td>Opens the project using the settings from the previous Rhapsody session</td>
</tr>
<tr>
<td>openProjectWithoutSubUnits</td>
<td>Opens the Rhapsody project without subunits</td>
</tr>
<tr>
<td>overrideInheritance</td>
<td>Overrides inheritance for the current state</td>
</tr>
<tr>
<td>quit</td>
<td>Closes the active Rhapsody project</td>
</tr>
<tr>
<td>rebuild</td>
<td>Rebuilds the application</td>
</tr>
<tr>
<td>recalculateEventsBaseId</td>
<td>Recalculates the events base ID of the package or project</td>
</tr>
<tr>
<td>refreshAllViews</td>
<td>Refreshes all the views</td>
</tr>
<tr>
<td>regenerate</td>
<td>Regenerates the active configuration of the active component</td>
</tr>
<tr>
<td>removeConveyed</td>
<td>Removes an information element from the conveyed collection</td>
</tr>
<tr>
<td>removeFromInstrumentationScope</td>
<td>Removes the classifier from the instrumentation scope</td>
</tr>
<tr>
<td>Method Name</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>removePackageFromInstrumentationScope</td>
<td>Removes the specified package from the instrumentation scope, including all its aggregated classes, actors, and nested packages</td>
</tr>
<tr>
<td>removeProperty</td>
<td>Removes the property from the model element</td>
</tr>
<tr>
<td>removeProvidedInterface</td>
<td>Removes the specified interface from the collection of required interfaces</td>
</tr>
<tr>
<td>removeRepresented</td>
<td>Removes a flowItem from the represented collection</td>
</tr>
<tr>
<td>removeRequiredInterface</td>
<td>Removes the specified interface from the collection of required interfaces</td>
</tr>
<tr>
<td>removeScopeElement</td>
<td>Deletes a scope element</td>
</tr>
<tr>
<td>removeStereotype</td>
<td>Removes the stereotype from the model element</td>
</tr>
<tr>
<td>report</td>
<td>Generates a report in ASCII or RTF into the specified file</td>
</tr>
<tr>
<td>resetEntryActionInheritance</td>
<td>Resets the inheritance of the entry action of the current state</td>
</tr>
<tr>
<td>resetExitActionInheritance</td>
<td>Resets the inheritance of the exit action of the current state</td>
</tr>
<tr>
<td>resetLabelInheritance</td>
<td>Resets the label inheritance</td>
</tr>
<tr>
<td>roundtrip</td>
<td>Roundtrips code changes back into the open model</td>
</tr>
<tr>
<td>save</td>
<td>Saves the current project</td>
</tr>
<tr>
<td>saveAs</td>
<td>Saves the current project to the specified file name and location</td>
</tr>
<tr>
<td>setActiveComponent</td>
<td>Sets the active configuration for the current project</td>
</tr>
<tr>
<td>setActiveConfiguration</td>
<td>Sets the active configuration for the current project</td>
</tr>
<tr>
<td>setClassType</td>
<td>Sets or changes the current template parameter to a class type parameter</td>
</tr>
<tr>
<td>setComponent</td>
<td>Sets the current component for the open project</td>
</tr>
<tr>
<td>setConfiguration</td>
<td>Sets the current configuration for the open project</td>
</tr>
<tr>
<td>setDirectory</td>
<td>Sets the directory for the current configuration</td>
</tr>
<tr>
<td>setEnd1ViaPort</td>
<td>Connects end1 of the flow to the specified instance via the given port (defined by the instance class)</td>
</tr>
<tr>
<td>setEnd2ViaPort</td>
<td>Connects end2 of the flow to the specified instance via the given port (defined by the instance class)</td>
</tr>
<tr>
<td>setGraphicalProperty</td>
<td>Allows the setting of graphical properties for a diagram element.</td>
</tr>
<tr>
<td>setInverse</td>
<td>Adds or updates the inverse relation</td>
</tr>
<tr>
<td>setItsAction</td>
<td>Updates the current transition with a new action</td>
</tr>
<tr>
<td>Method Name</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>setItsComponent</code></td>
<td>Sets the owning component for the current configuration</td>
</tr>
<tr>
<td><code>setItsGuard</code></td>
<td>Updates the current transition with a new guard</td>
</tr>
<tr>
<td><code>setItsLabel</code></td>
<td>Updates this transition with a new label (trigger[guard]/action)</td>
</tr>
<tr>
<td><code>setItsTrigger</code></td>
<td>Updates the current transition with a new trigger</td>
</tr>
<tr>
<td><code>setLog</code></td>
<td>Creates a log file that records all the information that is normally displayed in the Rhapsody output window</td>
</tr>
<tr>
<td><code>setOfState</code></td>
<td>Updates the source state of the current connector with a new state</td>
</tr>
<tr>
<td><code>setPath</code></td>
<td>Sets the path of the application built for this component</td>
</tr>
<tr>
<td><code>setPropertyValue</code></td>
<td>Modifies the value of the specified property</td>
</tr>
<tr>
<td><code>setReadOnly</code></td>
<td>Specifies whether the current unit is read-only</td>
</tr>
<tr>
<td><code>setReturnTypeDeclaration</code></td>
<td>Specifies a new value for the return type declaration</td>
</tr>
<tr>
<td><code>setSeparateSaveUnit</code></td>
<td>Sets a unit to be stored to its own file</td>
</tr>
<tr>
<td><code>setStaticReaction</code></td>
<td>Sets the static reaction for the current state</td>
</tr>
<tr>
<td><code>setTypeDeclaration</code></td>
<td>Sets the C++ type declaration for this argument</td>
</tr>
<tr>
<td><code>synchronizeTemplateInstantiation</code></td>
<td>Is used to synchronize between a template and a template instantiation parameter</td>
</tr>
<tr>
<td><code>unoverrideInheritance</code></td>
<td>Removes the override inheritance for the current state</td>
</tr>
<tr>
<td><code>version</code></td>
<td>Returns the version of Rhapsody that corresponds to the current COM API version</td>
</tr>
<tr>
<td><code>WhoAmI</code></td>
<td>Is invoked to identify the external code generator</td>
</tr>
<tr>
<td><code>write</code></td>
<td>Writes to the specified file</td>
</tr>
</tbody>
</table>
Index

A
Abort event 247
Action
entry 446
exit 447
Activities, reference 275
Activity diagram 138
Actors 350
add 353
delete 372
find 390
interface 54
Ada language external code generator 241
addSwimlane 276
Animation
enter command 57, 71
API 7
activeProject Method 38
available information 1
basic concepts 7
COM 35
conventions 50
creating applications 41
getNestedElementsRecursive method 40
hierarchy of classes 36
hierarchy of interfaces 2
interfaces 49
loading a project 38, 45
looping over packages 45
methods 529
openProject method 38
reference to application 38
reporting a project 39, 40
reporting on a project 39
Rhapsody reference 35
RHAPSODY.tlb file 35, 43
RPY/Explorer example 27
RPY/Reporter example 26
viewing Rhapsody objects 36
Application, creating VB applications 41
Attributes
delete 139
find 146

B
Base classifier 147
body property, IRPConstraint 232

C
C language 429
prototype 50
C++ language 49, 490
COM bindings 25
interfaces 49
isReference 106
prototype 50
setTypeDeclarations 98
visual 12, 13
Callback API 521
Class
accessing using VBA 322
find 392
Classifier
base 147
derived 148
Close 416
Code 31
Code generation, sample program 245
COM 35
API 7
API interfaces 49
API tools 7
Visual Basic API 7
COM bindings 25
Component
delete 417
find 418
Condition connector 224
Configuration
delete 194
find 196
Connectors
condition 224
diagram 225
fork 226
history 227
join 228
Junction 229
## Index

| stub | 230 |
| termination | 230 |
| constraintsByMe property | 232 |
| CountPackages macro | used in code example | 43 |
| Create | macro | 42 |
| project element | 18 |
| Create EMetaFile from the RPDiagram option | 28 |
| CreateObject | 38 |
| Custom helpers | 24 |

### D

| declaration property | IRPArgument | 98 |
| IRPAttribute | 106 |
| defaultValue property | IRPArgument | 98 |
| IRPAttribute | 106 |
| deferredAddToModel | 70 |
| Delete project element | 19 |
| Derived classifier | 148 |
| Diagrams | 28 |
| connector | 225 |
| storing | 28 |
| viewing | 28 |

### E

| Element | deleting | 19 |
| form | 31 |
| manipulating project | 18 |
| entryAction, method | 446 |
| Error codes | 23 |
| Error handling | 22, 242 |
| Events | abort | 247 |
| exit | 247 |
| Examples | findElementsByFullName | 322 |
| Radio | 42 |
| RPYReporter | 26 |
| VB program | 8 |
| Exit event | 247 |
| exitAction | 447 |

### F

| F8 key | 32 |
| File | delete | 195 |
| RHAPSODY.tlb | 35 |
| findElementsByFullName function | example | 322 |
| Flow items | 140 |
| Flows, delete | 141, 379 |
| Fork connector | 226 |
| Function, CreateObject | 38 |

### G

| Generalization | delete | 142 |
| find | 149 |
| getNestedElementsRecursive | used in sample | 40 |
| GraphElement | 286 |

### H

| Helpers | 24 |
| History connector | 227 |

### I

| Interfaces | 49, 50 |
| hierarchy of | 2 |
| Rhapsody | 7 |
| IRPCollection interface | using | 45 |
| VB sample | 39 |
| IRPModelElement interface | VB sample | 39 |

### J

| Java language | 344 |
| API | 5 |
| COM bindings | 25 |
| samples | 5 |
| Join connectors | 228 |
| Junction connector | 229 |

### K

| Keyboard icon | 42 |

### L

| Language property | 54 |
| Languages | COM API | 7 |
| Library, rhapsody.tlb | 7 |

### M

| Macros | CountPackages, used in a code example | 43 |
| creating sample | 42 |
| editing sample | 43 |
| running | 44 |
running sample 47
Methods 529
deferedAddToModel 70
entryAction 446
exitAction 447
parent 469
setTypeDeclaration for IRPArgument 99
stateType 456
Model, deferring 70
MS Word 42, 43

O
Object model diagram, delete 384
Object, type 32
Operation, delete 143

P
Package
  add 413
dele 385
parent method 469
Press new shortcut key option 42
Private keyword 33
Profile, add 414
Project
  deleting element 19
element, creating 18
elements, manipulating 18
  modifying an element 19
  open in VB 29
Properties 20, 31
  handling using the API 20
  manipulating 21
  propagation of default values 20
  VB 50

R
Radio example 42
Read from the Rhapsody API 13
Reference activity 275
Reference, definition 35
Relation, delete 144
Report, on API project 39
returnType property 345
Rhapsody 25
  .tlb file 35
  annotations 54
  API 35
  helpers 24
  project 13
  properties 20
  Radio example 42
  references 35
  Tools menu 28
Rhapsody API 7
  available information 1
callback 521
error handling 22
error handling codes 23
  handling properties 20
  hierarchy of interfaces 2
  interfaces 7
  manipulating project elements 18
  using with VB 7
  VBScript 9
  with Visual C++ 12
RHAPSODY.tlb file 35
rhapsody.tlb file 7
RPYReporter
  code summary, project loading 38
  code summary, project reporting 40
  example 26
Run Sub/UserForm option 44

S
Sample programs 26
  API 5
  code generator 245
  using VB 8
  using Visual C++ 13, 15
  VBScript 10
  Visual C++ reading project 13
Save 425
Save changes in field 42
Sequence diagram, delete 386
setTypeDeclaration
  IRPArgument 99
Solaris systems, VBScript 9
Start With Full Compile option 41
State, type 456
Statechart 136
dele 145
stateType, method 456
Store macro in field 42
Stub connector 230
Swimlane, add 276

T
Termination connector 230
Trigger, find 155
Type
delete 387
find 398
setTypeDeclaration for IRPArgument 99
state 456
type property
  IRPArgument 98
  IRPAttribute 106
Index

U
Usage, find 399
Use case diagram, delete 389

V
VB
catching an error condition 22
VB properties
  body for IRPConstraint 232
  constraintsByMe 232
  declaration for IRPArgument 98
  declaration for IRPAttribute 106
  defaultValue for IRPAttribute 106
  Language 54
  returnType 345
  type for IRPArgument 98
  type for IRPAttribute 106
VBScript
  running 9
  sample 10
  using 9
  writing files from 9
Visual Basic
  attributes 49
  code window 31
  compiling 41
  CreateObject function 38
  creating new projects 41
  forms 30, 31
  IDE 29
  loading a project 45
  making 41
  Menu File Editor option 32
  Object Browser option 36
  Open Project option 29
  Project Explorer window 29
  properties 50
  Properties window 30
  Reference dialog box 35
  sample program 8
  saving projects 41
  stepping through the code 32
  stopping execution 35
  using with the Rhapsody API 7
  Word VB IDE 42
Visual Basic Editor option 29
Visual C++
  and the Rhapsody API 12
  read sample 13
  write sample 15

W
Write
  files from VBScript 9
  to the Rhapsody API 15