Rational® Testing Products

Session recording and Script Generation Extensibility Reference

Version: 2002.05.00

Part Number: 800-025134-000

Rational®
the software development company

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

IMPORTANT NOTICE

Part Number: 800-025134-000

Version Number: 2002.05.00

Contents

Preface

About This Manual

This manual describes the APIs that you use to extend the Rational Robot session recording and script generation operations.

Audience

This manual is intended for protocol experts who want to develop session recorder, filter, and script generator adapters supported by the Rational Robot extensibility framework.

Other Resources

- This product contains online documentation. To access it, click **Session Recording Extensibility** in the following default installation path (*ProductName* is the name of the Rational® product you installed, such as Rational TestStudio®).

  Start > Programs > Rational *ProductName* > Rational Test > API

- All manuals for this product are available online in PDF format. These manuals are on the *Rational Solutions for Windows* Online Documentation CD.
- For information about training opportunities, see the Rational University Web site: http://www.rational.com/university.

Contacting Rational Technical Publications

To send feedback about documentation for Rational products, please send e-mail to our technical publications department at techpubs@rational.com.

Contacting Rational Technical Support

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

Your Location	Telephone	Facsimile	E-mail
North America	(800) 433-5444 (toll free) (408) 863-4000 Cupertino, CA	(781) 676-2460 Lexington, MA	support@rational.com
Europe, Middle East, Africa	+31 (0) 20-4546-200 Netherlands	+31 (0) 20-4545-201 Netherlands	support@europe.rational.com
Asia Pacific	+61-2-9419-0111 Australia	+61-2-9419-0123 Australia	support@apac.rational.com

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

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

Introduction to the Robot Extensibility Framework 1

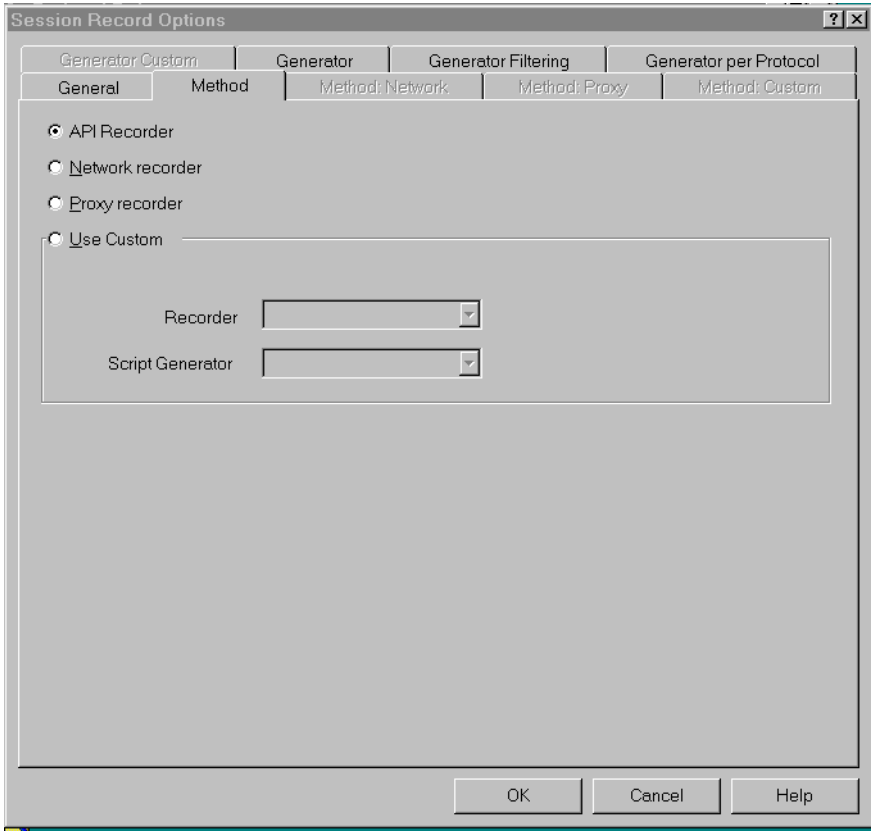
About Session Recording and Script Generation

This chapter explains how to extend the Rational® Robot session recording and script generation capabilities. It is organized as follows:

This Section	Describes
Overview of the Extensibility Framework on page 1	The choices provided by the extensibility framework.
Extending API Recording and Script Generation on page 3	The API recording and script generation framework, and the three types of adapter you develop to extend it.
Implementing Custom Recording and Script Generation on page 11	The two types of adapter you develop to implement the custom recording method. Rational's WebLogic adapters.
Installing Adapters on page 13	Where to put adapters such that Robot finds them on startup.
Limitations on page 15	Limitations of the extensibility framework.

Overview of the Extensibility Framework

Rational Robot supports four recording methods. These are presented to the Robot user on the Method tab of the Session Record Options dialog box, shown below.



The extensibility framework affects only the first and last choices, **API recorder** and **Use Custom**. On startup, Robot looks for installed adapters:

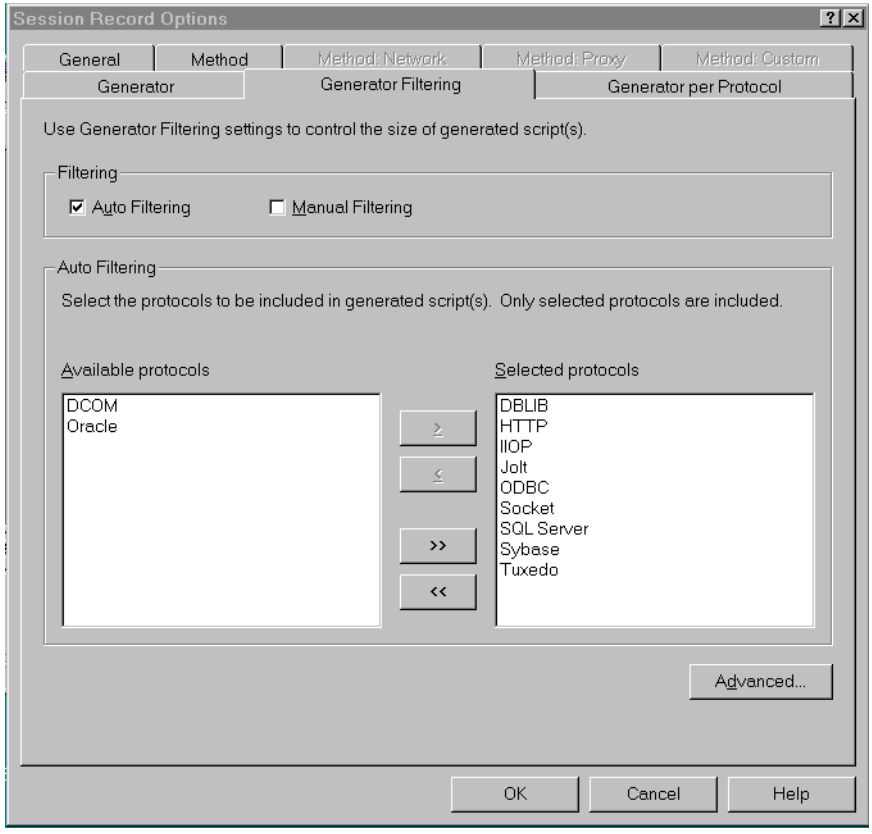
- If API adapters are detected, API recording and script generation are extended to one or more additional protocols that the Robot user selects from the Generator Filtering tab.
- If custom adapters are detected, the custom recording selection is enabled and one or more selectable adapters are listed in the **Recorder** and **Script Generator** boxes.

As the name implies, custom adapters work independently of Robot recording and script generation operations. They may be written in any language that can interface with C and recorder adapters can record in any format. By contrast, API adapters work inside the framework of the Robot session recording and script generation methodology. API adapters must be developed in C (or C++) and recorder adapters must use service functions provided with Robot for recording.

Extending API Recording and Script Generation

An API recording is a capture of information exchanged between an executing application-under-test (AUT) and a *target* dynamic-link library (DLL) used by the AUT. The AUT is an application front-end (client); a target is a client-side library implementing functions used by the application front-end to communicate with its server, which might run on a remote host. An example of an AUT is Internet Explorer. An example of an interesting target is winInet.dll, a well-documented dynamic-link library used by Internet Explorer. An example of an interesting function in winInet.dll is `InternetConnect`, which becomes active whenever an Internet Explorer user connects with a remote Web server.

During script generation, the total recorded data in a session file passes through one or more filtering programs, or *protocols*. This term describes a set of functions in a session recording that a script generator may find useful. Protocols are listed on the Generator Filtering tab of the Session Record Options dialog box.



By selecting a protocol, such as HTTP, the Robot user says in effect: if the session file includes any recorded calls to functions of this type, use the recorded data when generating test scripts.

The following table lists the API recording targets that are supported by Robot out-of-the-box. These DLLs need not (and cannot) be targeted by an extension adapter.

Descriptive Category	**Name**
Microsoft core	KERNEL32.DLL OLE32.DLL OLEAUT32.DLL
Microsoft communication	WSOCK32.DLL WS2_32.DLL WININET.DLL
Microsoft cryptographic	ADVAPI32.DLL CRYPT32.DLL RSABASE.DLL
Oracle (OCI)	ntt80.dll (dynamically loads WS2_32.DLL) ociw32.dll ora7nt.dll orant71.dll ora72.dll ora73.dll ora803.dll ora804.dll ora805.dll oci.dll orageneric8.dll oraclient8.dll
ODBC	odbc32.dll
DBLIB	ntwdblib.dll

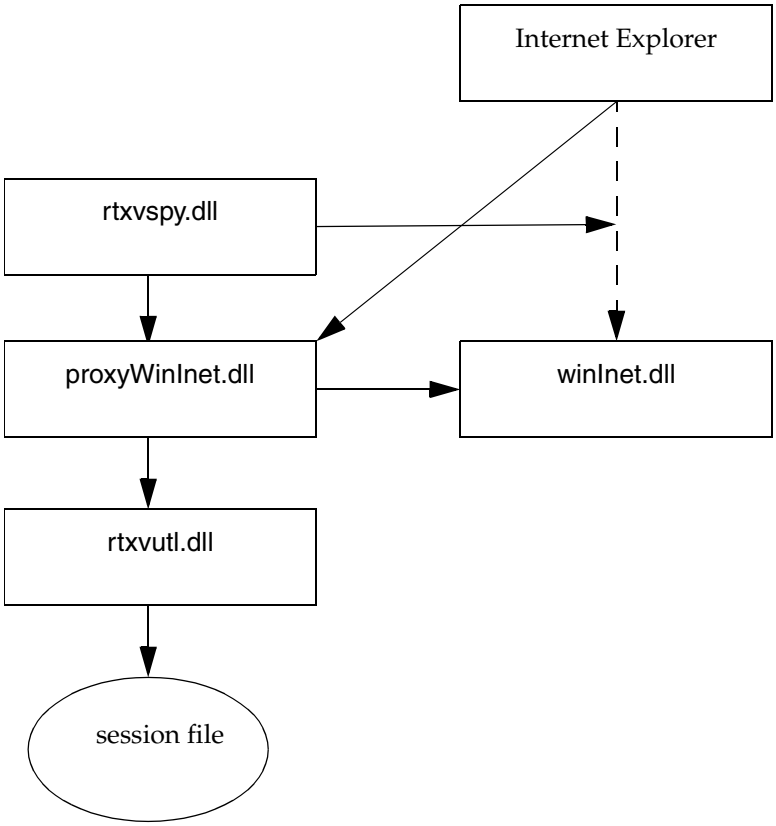
The API extensibility framework allows you to add to this list.

API Recording and Script Generation — Standard

Suppose that no API adapters are installed and that a Robot user:

- Selects **API record** from the Method tab.
- Selects **HTTP**, **IIOP**, and **Sybase** from the Generator Filtering tab.
- Clicks the Record Session icon and, when prompted for the AUT, starts Internet Explorer.

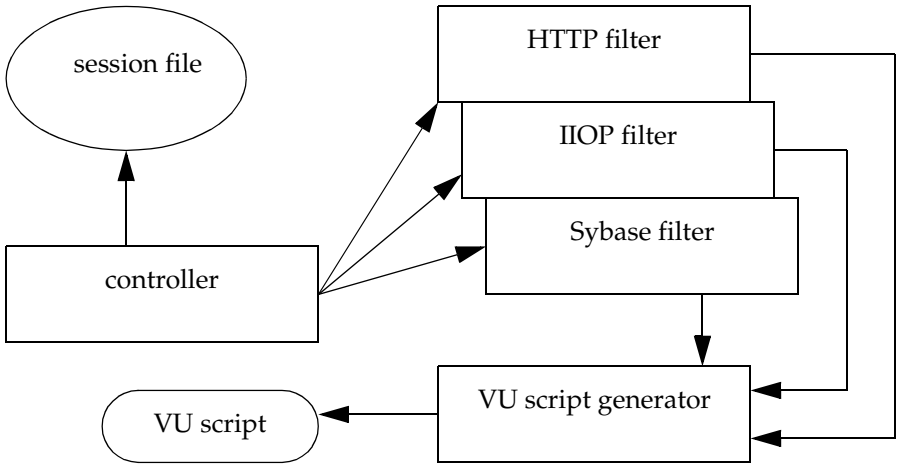
The diagram below illustrates what happens during the recording phase.



The targeted library in this example is winInet.dll, which is used by the AUT (Internet Explorer) to communicate with Web servers. When Internet Explorer starts, rtxvspy.dll attaches to its process space and waits for calls to functions defined in winInet.dll. Whenever a winInet.dll call occurs, the call is directed by rtxvspy.dll to a corresponding call in a proxy library (proxyWinInit.dll in the diagram).

The proxy library is the component responsible for API recording. For example, a call to the interesting function `InternetConnect` might be redirected to a proxy call in proxyWinInit.dll named proxyInternetConnect. The role of a proxy function is to decide what data should be captured and to send this to the Rational utility that maintains the proprietary session file, rtxvutl.dll. The functions needed to do this are documented in Chapter 4.

The diagram below illustrates what happens during the script generation phase.



The Rational recorder controller makes one or more passes through the session file, presenting each captured call in the session file to a *filter* corresponding to each user-selected protocol on the Generator Filtering tab. A filter inspects data sent to it and instructs the controller to forward relevant data to the VU script generator. (In actuality, this is a collection of script generators.)

API Recording and Script Generation — Extended

You extend API recording and script generation by developing and installing three types of adapters: an API Recorder Adapter, a Generator Filter Adapter, and an API Script Generator Adapter.

An API Recorder Adapter identifies:

- A new target DLL to be loaded.

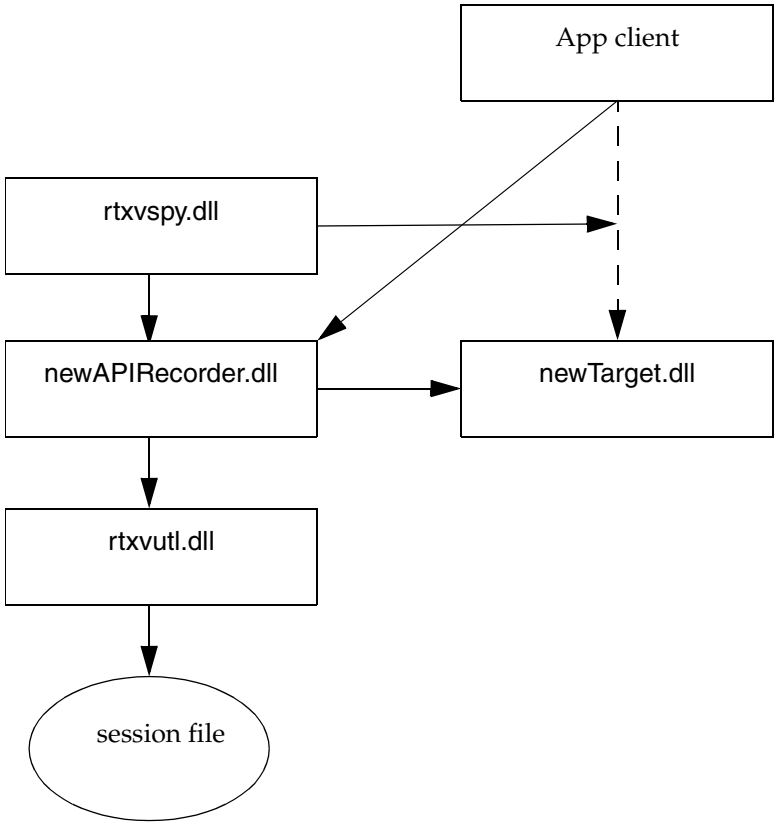
An API Recorder Adapter DLL includes a recording component (proxy) corresponding to the target. The AUT calls the proxy recorder instead of the target library, which is called by the proxy as needed. In this fashion, all traffic between the AUT and the target passes through the proxy. The functions used by the proxy to record functions are documented in Chapter 4.

- A list of functions implemented by the target DLL to be recorded.
- A corresponding list of *wrapper* (*proxy*) function names implemented in the proxy recorder component. Each real function of interest in the target DLL is implemented under the wrapped name in the proxy component.

A Generator Filter Adapter screens recorded calls for relevance to script generation.

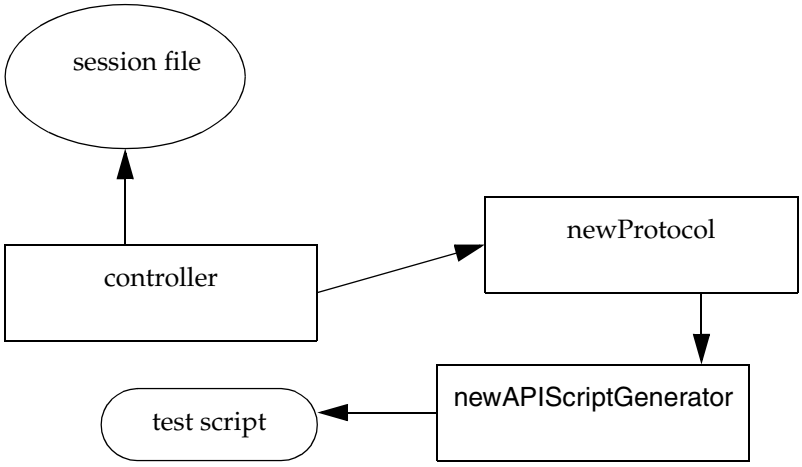
An API Script Generator Adapter generates a script from the data earmarked for it by its associated Generator Filter Adapter.

Suppose that you develop and install three API adapters named newAPIRecorder.dll, newFilter.dll, and newAPIScriptGenerator.dll. Now, when the Robot user initiates API recording, newAPIRecorder.dll is interjected into the AUT's process space by rtxvspy.dll such that client calls to newTarget.dll are redirected to the proxy recording functions in newAPIRecorer.dll.



The proxy functions in newAPIRecorder.dll decide what needs to be recorded and use the functions documented in Chapter 4, "Recording and Script Generation Services," to send this information to rtxvutl.dll, which adds it to the session file.

The script generation phase is modified as illustrated below. From the Generator Filtering tab, the Robot user selects the protocol name associated with newFilter.dll, which directs scriptable packets to newAPIScriptGenerator.dll.



The generated script can be in one of the three supported languages (VU, Java, Visual Basic). Rational® TestManager can play back test scripts you generate in these languages provided they are syntactically correct.

The Rational Test\rtsdk\c\rsrext\samples\APIExtensionAdapters folder includes the following three subfolders:

Recorders
Filters
ScriptGenerators

These folders contain skeletal adapters that you can use as starting points for developing full versions.

API Example

Sample extensible API adapters are provided in the Robot installation directory under Rational Test\rtsdk\c\rsrext\samples\APIExtensionAdapters\adapterSamples. The following table lists and describes the subfolders containing the components of this example.

Project Folder	Description
APIRecAdapterSample	Contains API Recorder Adapter and components installed with it: ▪ Test App is the AUT, a version of "Hello, World." ▪ HelloWorldAPI is the targeted library used by the AUT. Implements two functions, `HelloWorld1()` and `HelloWorld2()`. ▪ SampleAPIWrapper contains the components of the API Recorder Adapter DLL: ▫ A proxy recording component (`sampleAPIWrapper.c`). This component corresponds to the targeted library, `HelloWorldAPI`. It implements `pHelloWorld1()` and `pHelloWorld2()` and uses the calls documented in *Recording and Script Generation Services* on page 69 ▫ An API Recorder Adapter component (`DllMain.c`), implementing the calls documented in *API Recorder Adapter API* on page 17.
APIFilteringAdapterSample	Contains components of the API Generator Filter Adapter DLL, which implements the calls documented in *Generator Filter Adapter API* on page 21.
APISGenAdapterSample	Contains components of the API Script Generator Adapter DLL, which implements the calls documented in *API Script Generator Adapter API* on page 34.

This C++ example was created with Visual C++ Version 6. Build the targeted library (`HelloWorldAPI`) first, the sample AUT (Test App) second, and the proxy library (`sampleAPIWrapper.c`) third. The adapters may be built in any order. Before building, adjust project settings to pick up the appropriate include and library files, and to redirect output to the installation locations described in *Installing Adapters* on page 13. Install Test App.exe in the same directory as the API Recorder Adapter DLL. The target library can be installed anywhere provided `Test App.exe` can find it.

Implementing Custom Recording and Script Generation

If you install a Custom Recorder Adapter named newCustomRecorder.dll and a corresponding Custom Script Generator Adapter named newCustomScriptGenerator.dll, the Method tab of the Session Record Options dialog box displays your adapters (under their display names) to the Robot user. If the user selects the pair and initiates a custom recording session, newCustomRecorder.dll starts. Its session file, which can have any desired format, is available as input to newCustomScriptGenerator.dll.

The custom method is thus unrestricted: the two installed adapters can interact with one another, and possibly with other provided ancillary programs, in any way that you devise.

The Rational Test\\rtsdk\c\rsrext\samples\CustomAdapters folder includes the following subfolders:

Recorders
ScriptGenerators

These folders contain skeletal adapters that you can use as starting points for developing full versions.

Example — BEA WebLogic Adapters

BEA WebLogic is a popular application development and deployment environment based on the Java 2 Enterprise Edition (J2EE) specification and Enterprise Java Beans (EJB). Rational has developed custom adapters (written in Java) for this environment. If you install Robot and TestManager on a host on which the BEA WebLogic Server is installed, these adapters are ready for use.

The BEA WebLogic recorder adapter, `rtweblogicejb`, records method calls to a specific EJB in an XML-format session file. From the recording, the BEA WebLogic script generator adapter, `rtweblogicejb`, generates a Java test script representing the method calls and data sent to the EJB during the recorded session. The generated test script can be used to test the WebLogic server under varying load conditions.

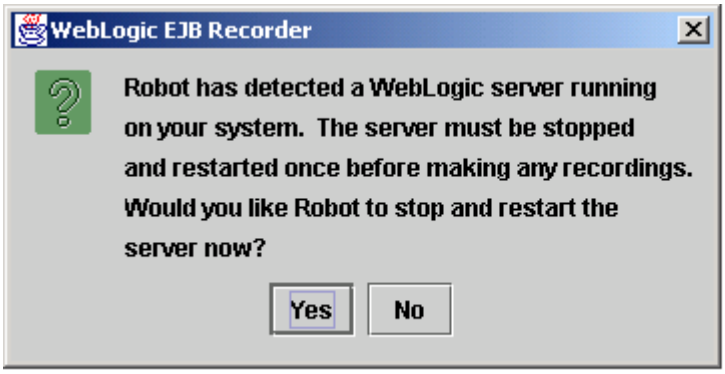
Requirements, Options, and Restrictions

General:

- The WebLogic recorder supports WebLogic Server version 6 or later, running on Windows NT or Windows 2000.
- The WebLogic recorder performs server-side recording. The WebLogic Server and Robot must be running on the same host. (Client EJBs can reside anywhere.)

Recording:

- To perform a recording, the WebLogic EJB probe and recorder must be running.
- At the start of recording, Robot must start the WebLogic server with the EJB recording probe attached. The commands for starting the server and inserting the probe are stored in the command file, startweblogic.cmd.)
- If the WebLogic server is already running at the time that recording is initiated, this prompt appears:



  Answer **Yes** to initiate recording, or **No** to abandon the recording request.

  If the WebLogic server is not currently running, it is started with the EJB probe attached.

  When the EJB probe is not recording, it remains idle.

- If you reboot your system, Robot starts the server and inserts the EJB probe again as noted above.

Options:

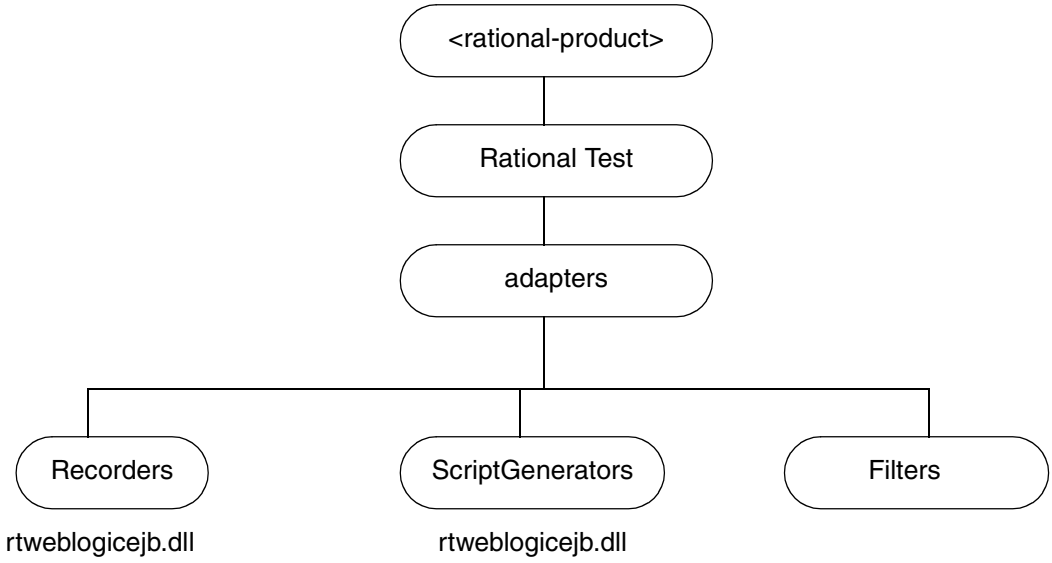
- By default, the probe records all traffic. Optionally, you can filter recorded traffic based on the WebLogic username (also called the principle). This username cannot be a Windows NT or 2000 username. Filtering works only if the EJB specifically supports the WebLogic username.
- The default server location is %WL_HOME%\config\mydomain. If your server is not located in mydomain, you can specify the location on the Method:Custom tab of the Session Record Options dialog.

Script generation:

- In the current release, generated Java test scripts include only the basic method and data required to test the EJB. Using an installed Java IDE, you can embellish these test scripts by manually adding Test Script Services (TSS) method calls for timing, logging, and other services. These services are documented in *Rational Test Script Services for Java*. (An HTML version of this manual is available from the Start menu at Rational Test\API\Rational TSS for Java.)
- You can play back Java test scripts from Rational TestManager. You can also play back Java test scripts on an agent provided the BEA WebLogic Server is installed on the agent.

Installing Adapters

The ability of Robot to detect adapters on startup is dependent on their being installed in known locations. The following diagram shows the relevant branches of the installation tree.



Several Rational products include Robot. The top of the tree, <rational-product>, is the name of the product. By default, this product is located under Program Files. The other folders are created by the installation directory. Locate:

- API Recorder Adapters and Custom Recorder Adapters in the recorders folder.
- API Script Generator Adapters and Custom Script Generator Adapters in the ScriptGenerators folder.
- Generator Filter Adapters in the Filters folder.

If your adapters use ancillary files, create a subfolder for them under recorders, ScriptGenerators, or filters. For example, under the recorders and ScriptGenerators folders, there is a subfolder named rtweblogicejb containing ancillary files used by Rational's adapters for the BEA WebLogic environment.

Header Files

Adapters are implemented as dynamic-link libraries (DLLs). The names of the header files required by each adapter type are listed in the following table. (Only Rational-supplied header files are listed, not headers that may be required by your development environment). These headers are located in the Robot installation directory under Rational Test\rtsdk\c\rsrext\include.

Header File	For
APIRecAdapter.h	API Recorder Adapter.
APISGenAdapter.h	API Script Generator Adapter.
CustomConfigurationUI.h	Custom Recorder Adapter, Custom Script Generator Adapter, Generator Filter Adapter.
CustRecAdapter.h	Custom Recorder Adapter.
CustSGenAdapter.h	Custom Script Generator Adapter.
ExtDefs.h	All adapters (included by adapter headers).
genfilteringadapter.h	Generator Filter Adapter.
proxhdr.h	Proxy recording services.
rtrsrutil.h	GetAnnotations().

Build Files

Files needed to build adapters are located in Rational Test\rtsdk\c\rsrext\lib.

Limitations

Generated VU scripts can reflect multiple protocols: from the Generator Filtering tab, a Robot user can select multiple protocols, and the resulting scripts are based on all relevant recorded data. By contrast, for a given API recording session, only a single adapter-supplied protocol name can be selected from the Generator Filtering tab.

For a given Custom recording session, a selected Custom Recorder Adapter works with a single Custom Script Generator Adapter. The Generator Filtering tab is not relevant for the Custom recording method.

Generated Java test scripts for BEA WebLogic do not support a number of features listed on the Generator tab and on the Session Insert tool bar: unsupported features appear dimmed.

Playback of generated BEA WebLogic test scripts from TestManager produces minimal performance analysis and reporting. This occurs because the generated scripts lack timing and logging method statements. To enhance these test scripts, edit them prior to playback and add the desired services as explained in Chapter 3 of *Rational Test Script Services for Java*.

API Adapter Reference 2

About API Adapters

This chapter documents the three APIs you use to develop adapters that extend the API recording and script generation method. These adapters are started when a Robot user selects **API recording** from the Method tab of the Session Record Options dialog box. These APIs are described in the following table:

API	Description
API Recorder Adapter API on page 17	Extends Robot's API recording to a new target and a new type of traffic. An API Recorder Adapter includes a recording component that uses the calls documented in Chapter 4, and requires a Generator Filter Adapter and a corresponding API Script Generator Adapter.
Generator Filter Adapter API on page 21	Directs scriptable functions recorded by an API Recorder Adapter to a corresponding API Script Generator Adapter.
API Script Generator Adapter API on page 34	Generates a test script(s) from recorded functions in a session file.

API Recorder Adapter API

An API Recorder Adapter is the interface between the Robot recording extensibility framework and a recording component (proxy library) that captures traffic between a designated target DLL and an application-under-test (AUT). The API Recorder Adapter and proxy library components must be in the same DLL. The recording functions used by the proxy library are documented in Chapter 4.

API Recorder Adapters implement the following calls:

Function	Description
`IsAPIRecorderAdapter()`	Identifies an API Recorder Adapter.
`GetAPIRecAdapterInfo()`	Returns information about associated adapters and protocol.
`GetApiAndWrapperNamePairs()`	Returns names of calls to be intercepted and their wrapper names.

IsAPIRecorderAdapter()

Identifies an API Recorder Adapter.

Syntax

```
BOOL IsAPIRecorderAdapter();
```

Comments

Return TRUE to indicate that this is an API Recorder Adapter. Any other response disables the adapter.

Example

```
extern "C"
BOOL LINKDLL IsAPIRecorderAdapter()
{
   return TRUE;
}
```

See Also

`IsAPIGenFiltExtAdapter(),IsAPISgenAdapter()`

GetAPIRecAdapterInfo()

Returns information about associated adapters and protocol.

Syntax

```
int GetAPIRecAdapterInfo (TCHAR *info, size_t infoSize);
```

Element	Description
info	Pointer to a container for adapter information. Copy to this location a string containing the following comma-separated items: ▪ The name of the DLL implementing this API recorder adapter. ▪ The name of the target DLL. The recording component (proxy) records traffic between this target and the application-under-test. The calls to be intercepted and their corresponding wrapper names are specified by `GetApiAndWrapperNamePairs()`. ▪ The name of the Generator Filter Adapter DLL to be used with this adapter. ▪ The name of the DLL implementing the API Script Generator Adapter to be used with this recorder adapter. ▪ The protocol name. This is the public name (listed on the Generator Filtering tab of the Session Record Options dialog box) of the Generator Filter Adapter DLL, returned by *GetDisplayName()* on page 24. ▪ A description of this adapter, or "". ▪ The version of this adapter, or "".
infoSize	INPUT. The size allocated for *info*, which must not exceed this size.

Comments

Return `RSR_SUCCESS` to indicate that the call is complete.

Example

An adapter responds to this call as illustrated below. In the terms of this example, the following happens: if a Robot user selects **API Recording** from the Method tab and **MyProtocol** from the Generator Filtering tab and activates recording, traffic between the AUT and `HelloWorld.dll` specified by `GetApiAndWrapperNamePairs()` is routed to `MyAPIRecorder.dll`.

```
extern "C"
int LINKDLL GetApiRecAdapterInfo (TCHAR *info, size_t infoSize)
{
   static TCHAR buffer[] =
      "MyAPIRecorder.dll,"
      "HellowWorld.dll,"
      "MyGFA.dll,"
      "MySGA.dll,"
      "MyProtocol,"
      "This is an example,”
      "Version 0.1";

   if (_tcslen(buffer) >= infoSize)
```

```
      return RSE_BUFFER_TOO_SHORT;
   else
   {
      _tcscpy(info, buffer);
      return RSR_SUCCESS;
   }
}
```

See Also

`IsAPIRecorderAdapter(),GetApiAndWrapperNamePairs()`

GetApiAndWrapperNamePairs()

Returns names of calls to be intercepted and their wrapper names.

Syntax

LPSTR **`GetApiAndWrapperNamePairs`** (int *iNumPairs);

Element	Description
`iNumPairs`	The number of comma-separated pairs that are returned.

Comments

Your response to this call gives the name of each function call to be intercepted in the dialog between the target DLL and application-under-test, and the function's corresponding wrapper name. Calls to targeted functions are redirected to corresponding functions in the recording component (proxy library) of the API Recorder Adapter DLL.

Example

This example returns two API/wrapper pairs:

```
extern "C"
LPSTR GetApiAndWrapperNamePairs (int *iNumPairs)
{
   static TCHAR szPairs[] =
      "HelloWorld1, pHelloWorld1, HelloWorld2, pHelloWorld2";
   *iNumPairs = 2;
   return szPairs;
}
```

See Also

`IsAPIRecorderAdapter(),GetAPIRecAdapterInfo()`

Generator Filter Adapter API

If you extend API recording, you must provide a Generator Filter Adapter (GFA). GFAs are listed on the Generator Filtering tab of the Session Record Options dialog box, by the name you specify with `GetDisplayName()`. Each GFA is associated with a single Script Generator Adapter (SGA), whose name is returned by `GetScriptgenDllName()`. When a user selects a GFA and starts the AUT, recording of traffic between it and the target DLL commences. The role of the GFA is to direct recorded information packets that are relevant to script generation to the SGA.

Generator Filter Adapters implement the following calls. The shaded rows list functions that other adapter types also implement.

Function	Description
IsAPIGenFiltExtAdapter()	Identifies a Generator Filter Adapter.
GetScriptgenDllName()	Returns the name of the associated API Script Generator Adapter.
CheckAPIPacket()	Checks for scriptable packets.
GetDisplayName()	Returns the public name of this adapter.
GetOptions()	Returns configuration options in effect for this adapter.
SetOptions()	Sets user-specified configuration options for this adapter.
DisplayCustomConfigGUI()	Provides a custom GUI for adapter-defined configuration options.

IsAPIGenFiltExtAdapter()

Identifies a Generator Filter Adapter.

Syntax

```
BOOL IsAPIGenFiltExtAdapter();
```

Comments

Return TRUE to indicate that this is a Generator Filter Adapter. Any other response disables the adapter.

Example

This response identifies an adapter as a Generator Filter Adapter:

```
extern "C"
BOOL LINKDLL IsAPIGenFiltExtAdapter()
{
   return TRUE;
}
```

See Also

`IsAPIRecorderAdapter()`, `IsAPISgenAdapter()`

GetScriptgenDllName()

Returns the name of the associated API Script Generator Adapter.

Syntax

```
int GetScriptgenDllName (TCHAR *name, size_t nameSize);
```

Element	Description
name	Pointer to a container for the name of the API Script Generator Adapter DLL that is used with this adapter. Copy the adapter's name, including suffix, to this location.
nameSize	INPUT. The size allocated for *name*, which cannot exceed this size.

Comments

When a user selects this Generator Filter Adapter from the Generator Filtering tab, the API Script Generator Adapter named with this call is also selected.

Example

This example specifies that the API Script Generator Adapter to be used with this adapter is named `APISGenAdapterSample.dll`.

```
extern "C"
int LINKDLL GetScriptgenDllName (TCHAR *name, size_t nameSize)
{
   TCHAR buf[] = "APISGenAdapterSample.dll";
   if (_tcslen(buf) >= nameSize
      return RSR_BUFFER_TOO_SHORT;
   else
   {
      _tcscpy(name, buf);
      return RSR_SUCCESS;
   }
}
```

See also

`GetDisplayName()`

CheckAPIPacket()

Checks for scriptable packets.

Syntax

```
int CheckAPIPacket (void *packet, int IDNum, int *NextConn, int
   *clientIP, int *clientPort, int *serverIP, int *serverPort);
```

Element	Description
packet	Pointer to a location containing the name of an API packet in the session file.
IDNum	INPUT. The API packet ID assigned by the session controller.
NextConn	INPUT/OUTPUT. A packet ID assigned by the session controller that, if changed, earmarks a packet for the associated API Script Generator Adapter.
clientIP	INPUT. Pointer to the client IP address (hexadecimal notation).
clientPort	INPUT. Pointer to the port used by the client.
serverIP	INPUT. Pointer to the server's IP address.
serverPort	INPUT. Pointer to the port used by the server.

Comments

The session controller presents all recorded functions in the session file to all Generator Filter Adapters. Your adapter inspects the packets and determines which are relevant for script generation. You can make multiple passes through the session file.

Every packet of data received by your adapter has a controller-assigned *NextConn* identifier. If, upon inspection, you determine that a packet is relevant to script generation, change *NextConn* to a designated number. The session controller keeps track of IDs you change and, at the conclusion of filtering, sends data you have earmarked in this fashion to the associated API Script Generator Adapter.

Example

See *Sample API Generator Filter Adapter* on page 29.

See Also

`GetScriptgenDllName(), PassComplete(), ProcessAPIPacket()`

GetDisplayName()

Returns the public name of this adapter.

Syntax

```
int GetDisplayName (TCHAR *name, size_t nameSize);
```

Element	Description
name	Pointer to a container for the adapter's display name. Copy the name to this location.
nameSize	INPUT. The size allocated for *name*. The adapter's display name cannot exceed this size.

Comments

Specify the GUI name for this adapter. This is the name (protocol) that is displayed on the Generator Filtering dialog.

Example

This example specifies that the protocol name associated with this adapter is `MyAPIProtocol`.

```
extern "C"
int LINKDLL GetDisplayName (TCHAR *name, size_t nameSize)
{
   TCHAR buf[] = "MyAPIProtocol";
   if (_tcslen(buf) > nameSize)
      return RSR_BUFFER_TOO_SHORT;

   _tcscpy(name, buf);
   return RSR_SUCCESS;
}
```

GetOptions()

Returns configuration options in effect for this adapter.

Syntax

```
int GetOptions (TCHAR *options, size_t optionsSize);
```

Element	Description
options	Pointer to a container for this adapter's options. Copy supported options, separated by semicolons, to this location. Robot-defined options have the format: `argument[,setting]` where *argument* is one of the strings described in the options table shown below, and *setting* can be a value for the argument. Adapter-defined options have the format: *name*,*value*,*description*[,*value1*, *value2*, ...*valuen*] Adapter-defined options that you return in response to this call appear on the Generator per Protocol tab of the Robot Session Record Options dialog. They can also appear on an adapter-supplied GUI.
optionsSize	INPUT. The size allocated by Robot for *options*, which must not exceed this size.

Comments

As illustrated in the example, options supported by an adapter should be entered from a saved, local file. Otherwise, they do not persist between sessions.

Your adapter can define a custom format for options and provide a custom GUI for displaying and editing them and code to communicate the user's choices to the adapter. Do not include custom-format options in your response to this call.

The following table describes the Robot-defined configuration option arguments that a Generator Filter Adapter can support. See *Adapter Configuration* on page 81 for a mapping of these options to the Robot GUI.

Configuration Option	Description
`CONFIGURATION,USE_CUSTOM_UI`	Specifies that the adapter provides a custom GUI for displaying and selecting adapter-defined configuration options.
`CONFIGURATION,`*`name`*`,`*`value`*`,`*`description`*`[,`*`value1`*`,`*`value2`*` ...]`	The adapter supports a configuration option of the specified ***name*** and ***value***, which works as indicated by the ***description***. Adapter-defined options may be entered either from the Generator per Protocoltab of the Session Record Options dialog or from a supplied custom GUI. If a configuration triplet includes [, value1, value2, ...] , the supplied values are implemented by a ***value*** pull-down menu on the grid.

Example

The following response indicates that this adapter supports a single custom configuration triplet:

```
extern "C"
int LINKDLL GetOptions(TCHAR* options, size_t optionsSize)
{
   TCHAR buf[RSR_MAX_OPTIONS];
   _tcscpy(buf, _T(""));
   _tcscat(buf, CONFIGURATION);
   _tcscat(buf, _T(",Password,system,Enter server password"));
   _tcscat(buf, _T(";"));

   if (_tcslen(buf) > optionsSize)
      return RSR_BUFFER_TOO_SHORT;
   _tcscpy(options, buf);
   return RSR_SUCCESS;
}
```

See Also

SetOptions()

SetOptions()

Sets user-specified configuration options for this adapter.

Syntax

```
int SetOptions (TCHAR *options, size_t optionsSize);
```

Element	Description
options	INPUT. Pointer to a read-only location containing the Robot user's selections.
optionsSize	INPUT. The size of *options*.

Comments

When the user selects a Robot-defined option or edits an adapter-defined option, this call communicates the user's choice to your adapter.

This call also returns a user's choices for adapter-defined options, in the triplet format, that were selected from a Robot-provided dialog. However, if you use a custom GUI for displaying and editing custom options, you are responsible for reading the dialog, conveying the user's choices to the adapter, parsing, validation, and sending an appropriate error message for invalid user specifications.

Example

This example checks to see whether a Robot user selected the **Think maximum (ms)** option.

```
extern "C"
int LINKDLL SetOptions(TCHAR* options, size_t optionsSize)
{
   /* CStringArray declared for parsed sub-strings */
   CStringArray OptionsArray;

   /* parse the original string with semi-colon delimeter.*/
   ParseString(options,';',&OptionsArray);

   for(int i = 0;i<OptionsArray.GetSize();i++)
   {
      /* for every sub-string, create another CStringArray*/
      CStringArray SubArray;
      if(!OptionsArray.GetAt(i).IsEmpty())
      {
         /*parse the substrings with comma delimeters */
         ParseString (OptionsArray.GetAt(i).GetBuffer
            (OptionsArray.GetAt(i).GetLength()),',',&SubArray);

         /* deal with each sub-string set */
         if(SubArray[0]==GENERATOR_THINK)
         {
            if(SubArray[1] == 0)
            {
            int think_min = SubArray[2];
         }
```

```
            else
            {
               /* Unrecognized option -- error may be thrown*/
            }
         }
      }
   }
```

See Also

`GetOptions()`

DisplayCustomConfigGUI()

Provides a custom GUI for adapter-defined configuration options.

Syntax

```
int DisplayCustomConfigGUI (TCHAR *errorMessage, size_t
   errorMessageSize);
```

Element	Description
errorMessage	Pointer to a location for a message to be displayed in the event of error. Copy the message to this location.
errorMessageSize	INPUT. The size allocated for *errorMessage*, which cannot exceed this size.

Comments

If your adapter specifies the option `CONFIGURATION,USE_CUSTOM_UI`, a **Configure** button on the Generator per Protocol tab of the Session Record Options dialog is enabled. If a user clicks this button, Robot issues this call. In response, your adapter should display a custom GUI for entering or editing custom configuration options.

If you provide a custom GUI:

- The format of custom configuration options is entirely up to you: it is not necessary to include custom options with your response to `GetOptions()`, and the options need not adhere to the triplet format defined by Robot for custom options.
- If you do adhere to the triplet format for custom options and include them in your response to `GetOptions()`, the Robot user can use a provided triplet grid as well as your custom GUI.

- If you do not adhere to the triplet format for custom options, do not include them in your response to `GetOptions()`. Doing so causes an error because the format is not understood.
- Custom options chosen or edited using the Robot-supplied triplet grid are conveyed to an adapter by `SetOptions()`. With a custom GUI, you are responsible for reading and persisting user choices.
- If the GUI you provide includes online Help, it works as implemented. In any case, you are responsible for providing any documentation that users require.

Example

```
extern "C"
int LINKDLL DisplayCustomConfigGUI (TCHAR *errorMessage, size_t
errorMessageSize)
{
   //display custom GUI and gather user input
   if (successful)
      return RSR_SUCCESS;
   else
   {
      TCHAR buf[] = "Custom GUI failed to start";
      if (_tcslen(buf) > errorMessageSize)
         return RSR_BUFFER_TOO_SHORT;

      _tcscpy(errorMessage, buf);
      return RSR_FAILURE;
   }
}
```

See Also

`GetOptions(),SetOptions()`

Sample API Generator Filter Adapter

The following sample is part of the extended example described in *API Example* on page 9.

```
////////////////////////////////////////////////////////////////////////
// Copyright (c) Rational Software Corporation. 2001
// All Rights Reserved.
//
// FILENAME:APIFilteringAdapterSample.c
// DESCRIPTION:implementation of Sample "APIFilteringAdapterSample"
//
// REVISION HISTORY
// PROGRAMMERDATEREVISION
//
========================================================================
```

```
//
// DJM        05/25/01initial version
//
//////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////
// Includes
//////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////

#include <windows.h>
#include <tchar.h>

// Rational headers
#include "ExtDefs.h"
#include "proxhdr.h"
#include "psystem.h"
#include "plibdefs.h"
// local
#include "APIFilteringAdapterSample.h"

static unsigned int uiLibID = 0;

char buf2;
#define RSR_EXT_STUB_SILENT "RSR_EXT_STUB_SILENT"

//////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////
// Function Definitions
//////////////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////////////


BOOL IsAPIGenFiltExtAdapter()
{
   return TRUE;
}

int GetDisplayName(TCHAR* name, size_t nameSize)
{
   TCHAR buf[] = "MyAPIProtocol";

   if (_tcslen(buf) <  nameSize)
   {
      _tcscpy(name, buf);

      if(!GetEnvironmentVariable(RSR_EXT_STUB_SILENT,&buf2,0))
         MessageBox(NULL, "get Display name called and succeeded", "DJM
filter adapter", MB_OK);

      return RSR_SUCCESS;
   }
   else
```

```
   {
      return RSR_ERROR;
   }

}

int GetScriptgenDllName(TCHAR* name, size_t nameSize)
{
   TCHAR buf[] = "APISGenAdapterSample.dll";

   //if (_tcslen(buf) <  nameSize)
   //{
      _tcscpy(name, buf);
   if(!GetEnvironmentVariable(RSR_EXT_STUB_SILENT,&buf2,0))
      MessageBox(NULL, name, "filter, DJM getscriptgenname", MB_OK);

      return RSR_SUCCESS;
   /*}
   else
   {
      return RSR_ERROR;
   }*/


}

int GetOptions(TCHAR* options, size_t optionsSize)
{

     TCHAR opVar[4096];
     _tcscpy(options, _T(""));

    if (GetEnvironmentVariable(_T("RSR_FILTER_OPTIONS"), opVar, 4096)
!= 0)
    {
     _tcscpy(options, opVar);
        return RSR_SUCCESS;
    }

    _tcscat(options, "options;options");

    if (_tcslen(options) >= optionsSize)
        return RSR_BUFFER_TOO_SHORT;
    else
        return RSR_SUCCESS;
}

int SetOptions(TCHAR* options, size_t optionsSize, TCHAR* errorMsg,
size_t errorMsgSize)
{
   return RSR_SUCCESS;
}

int DisplayCustomConfigGUI(TCHAR* errorMessage,
```

```
                          size_t errorMessageSize)
{
   return RSR_SUCCESS;
}

int CheckAPIPacket(void* packet,
                          int  IDNum,
                          int* NextConn,
                          int* clientIP,
                          int* clientPort,
                          int* serverIP,
                          int* serverPort)
{

   major_wch_header *majorhdr;
   char buf[MAX_PATH];

   majorhdr = packet;
   // this example doesn't pass any packet type besides API_SINGLE
   // on to the script generator
   if (majorhdr->major_type == API_SINGLE)
   {
      minor_wch_header *minorhdr = (minor_wch_header *)((char*)packet +
sizeof(major_wch_header));
      // I'm interested in the following packet types
      switch (IDNum)
      {
         // add filters for other libraries here
         case PROXY_LIB_WINDOWS_SYSTEM:
               // check all the customer adapter records
               // to see if it is ours
               if (minorhdr->API_id != P_CUSTOMADAPTER_A)
               {
                  *NextConn = 0;
               }
               else
               {  // custom adapter record, let's
                  // see if it is ours
                  S_CUSTOMADAPTER *custadapter = (S_CUSTOMADAPTER
*)((char*)packet + sizeof(major_wch_header)
                     + sizeof(minor_wch_header) +
sizeof(stamp2_wch_header));
                  void *p = (void *)((char*)packet +
sizeof(major_wch_header)
                     + sizeof(minor_wch_header)  +
sizeof(stamp2_wch_header) + custadapter->iOffszLibraryName);
                  char *szLibName = (char *)p;
                  /*
                  wsprintf(buf, "sizeofmajor=%d sizeofminor=%d",
sizeof(major_wch_header),
                     sizeof(minor_wch_header));
                  MessageBox(NULL, buf, "DJM Filter", MB_OK);
                  wsprintf(buf, "szOffset=%d",
custadapter->iOffszLibraryName);
```

```
                MessageBox(NULL, buf, "DJM Filter", MB_OK);
                MessageBox(NULL, szLibName, "DJM Filter", MB_OK);
                MessageBox(NULL, "custadapterpacket found", "DJM
filter adapter", MB_OK);
                */
                if (stricmp(szLibName, "SampleAPIWrapper.dll") == 0)
                {
                   uiLibID = custadapter->uiLibID;
                   MessageBox(NULL, szLibName, "Identified System
library record",  MB_OK);
                   //wsprintf(buf, "lib ID=%d", uiLibID);
                   //MessageBox(NULL, buf, "DJM lib id", MB_OK);
                }
                else
                   *NextConn = 0;

             }
             break;
        default:
             // check for any records that are to be thrown away
             //wsprintf(buf,"default processing libID=%d looking for
uiLibID=%d",IDNum, uiLibID);
             //MessageBox(NULL,buf,"DJM non sys packet", MB_OK);

             if (IDNum != (int)uiLibID)
                *NextConn = 0;
             if (uiLibID == 0)
                *NextConn = 0;
             if (IDNum == (int)uiLibID)
             {
                wsprintf(buf,"processing with good data packet conn
number=%d looking for uiLibID=%d", *NextConn, uiLibID);
                MessageBox(NULL,buf,"DJM Filter", MB_OK);
                //MessageBox(NULL, "found packet to process", "DJM
Filter", MB_OK);


             }
             break;

     }

  }
  else
  {
     *NextConn = 0;
  }

  return RSR_SUCCESS;
}
```

API Script Generator Adapter API

Implement this API in a script generator adapter to be used with an API Recorder Adapter. Implement the *Custom Script Generator Adapter API* on page 55 in a script generator adapter to be used with the custom recording method.

API Script Generator Adapters implement the following calls. The shaded rows list functions that other adapter types also implement.

Function	Description
IsAPISgenAdapter()	Identifies an API Script Generator Adapter.
InitializeScriptgen()	Performs initialization procedures.
ProcessAPIPacket()	Receives an API packet for processing.
PassComplete()	Returns the pass completion status.
GetStatus()	Returns the progress status.
GetOptions()	Returns configuration options in effect for this adapter.
SetOptions()	Sets user-specified configuration options for this adapter.

IsAPISgenAdapter()

Identifies an API Script Generator Adapter.

Syntax

```
BOOL IsAPISgenAdapter();
```

Comments

Return TRUE to indicate that this is an API Script Generator Adapter. Any other response disables the adapter.

This adapter is associated with a single Generator Filter Adapter. When a user selects the associated Generator Filter Adapter from the Generator Filtering tab and starts an API recording session, this adapter receives any scriptable packets that are exchanged between the AUT and the target.

Example

A C adapter should respond to this call as illustrated below.

```
extern "C"
BOOL LINKDLL IsAPISgenAdapter()
{
   return TRUE;
}
```

See Also

`IsAPIRecorderAdapter()`, IsAPIGenFiltExtAdapter()

InitializeScriptgen()

Performs initialization procedures.

Syntax

```
int InitializeScriptgen (TCHAR *scriptPathname);
```

Element	Description
scriptPathname	Pointer to a location containing the path name of the script file (stored inside the Rational datastore). The script filename base is supplied by the Robot user. If split scripts are supported, their names are generated from this base.

Comments

An initialization procedure sets up the output path for scripts and can perform other startup functions.

Example

This example illustrates a successful response.

```
extern "C"
int LINKDLL InitializeScriptgen (TCHAR *scriptPathname)
{
   MessageBox(NULL, "InitializeScriptgen Called","Hello from APISgen
Adapter Stub!", MB_OK);
   return RSR_SUCCESS;
}
```

ProcessAPIPacket()

Receives an API packet for processing.

Syntax

```
int ProcessAPIPacket (void *packet, int IDNum);
```

Element	Description
packet	INPUT. Pointer to a container for the name of an API packet in the session file.
IDNum	INPUT. The API packet ID assigned by the session controller.

Comments

The associated Generator Filter Adapter designated this packet as relevant for script generation.

Example

This example illustrates a successful response.

```
extern "C"
int LINKDLL ProcessAPIPacket (void *packet, int IDNum)
{
   MessageBox(NULL, "ProcessAPIPacket Called","Hello from APISgen
Adapter Stub!", MB_OK);
   return RSR_SUCCESS;
{
```

See Also

```
CheckAPIPacket()
```

PassComplete()

Returns the pass completion status.

Syntax

```
int PassComplete (int passNumber);
```

Element	Description
passNumber	INPUT. The ID of the pass through the session file.

Comments

The packets in a session file are read by the session controller and presented to the Generator Filter Adapter, which designates packets that are relevant to script generation. There are usually multiple passes through the session file.

Return RSR_SUCCESS to indicate that script generation is progressing normally. Otherwise, return a nonzero integer and supply an appropriate error message.

Example

This example illustrates a successful response.

```
extern "C"
int LINKDLL PassComplete (int passNumber)
{
   MessageBox(NULL, "PassComplete Called","Hello from APISgen Adapter
Stub!", MB_OK);
   return RSR_SUCCESS;
}
```

See Also

`CheckAPIPacket(),ProcessAPIPacket()`

GetStatus()

Returns the progress status.

Syntax

```
int GetStatus (TCHAr *msg, size_t msgSize);
```

Element	Description
msg	Pointer to a container for the status message to be displayed. Copy the message to this location.
msgSize	INPUT. The size allocated for *msg*, which cannot exceed this size.

Comments

Return RSR_SUCCESS to indicate that script generation is progressing normally. Otherwise, return a nonzero integer and supply an appropriate error message.

Example

This example illustrates a successful response.

```
extern "C"
int LINKDLL GetStatus (TCHAR *msg, size_t msgSize)
{
   MessageBox(NULL, "GetStatus Called","Hello from APISgen Adapter
Stub!", MB_OK);
   return RSR_SUCCESS;
}
```

GetOptions()

Returns configuration options in effect for this adapter.

Syntax

```
int GetOptions (TCHAR *options, size_t optionsSize);
```

Element	Description
options	Pointer to a container for this adapter's options. Copy supported options, separated by semicolons, to this location. Robot-defined options have the format: `argument`[,`setting`] where *argument* is one of the strings described in the options table shown below, and *setting* can be a value for the argument. Generator:Custom
optionsSize	INPUT. The size allocated by Robot for *options*, which must not exceed this size.

Comments

As illustrated in the example, options supported by an adapter should be entered from a saved, local file. Otherwise, they do not persist between sessions.

The following table describes the Robot-defined configuration option arguments that a Script Generator Adapter can support. See *Adapter Configuration* on page 81 for a mapping of these options to the Robot GUI.

Configuration Option	Description
GENERATOR_BIND_VU_VARS	On the Generator tab of the Session Record Options dialog, the **Bind output parameters to VU variables** check box is enabled.
GENERATOR_COMMAND_ID	On the Generator tab of the Session Record Options dialog, the **Command ID prefix** box is enabled.

Configuration Option	Description
GENERATOR_CPU_THRESH	On the Generator tab of the Session Record Options dialog, the **CPU/user threshold (ms)** check box is enabled.

GENERATOR_DISPLAY_ROWS	On the Generator tab of the Session Record Options dialog, the **Display recorded rows** boxes are enabled. The Robot user's choices are communicated by `SetOptions()` in the format:
	`GENERATOR_DISPLAY_ROWS, choice, rows`
	where:
	▪ *choice* is one of these values corresponding to the user's selection of **None**, **First**, **Last**, or **All**:
	`RSR_DISPLAY_RECORDED_ROWS_NONE`
	`RSR_DISPLAY_RECORDED_ROWS_FIRST`
	`RSR_DISPLAY_RECORDED_ROWS_LAST`
	`RSR_DISPLAY_RECORDED_ROWS_ALL`
	▪ *rows* is 0 (for All or None) or the specified number of rows.

GENERATOR_PLAYBACK_PACING	On the Generator tab of the Session Record Options dialog, the **Playback pacing** radio boxes are enabled. The Robot user's choice of **per command**, **per script**, or **none** is communicated by `SetOptions()` as:
	`RSR_GENERATOR_PLAYBACK_PACING_COMMAND`
	`RSR_GENERATOR_PLAYBACK_PACING_SCRIPT`
	`RSR_GENERATOR_PLAYBACK_PACING_NONE`

GENERATOR_THINK	On the Generator tab of the Session Record Options dialog, the **Think maximum (ms)** check box is enabled.
GENERATOR_USE_DATAPOOLS	On the Generator tab of the Session Record Options dialog, the **Use datapools** check box is enabled.
GENERATOR_VERIFY_RETURN_CODES	On the Generator tab of the Session Record Options dialog, the **Verify playback return codes** check box is enabled.
GENERATOR_VERIFY_ROW_COUNTS	On the Generator tab of the Session Record Options dialog, the **Verify playback row counts** check box is enabled.

Configuration Option	Description
TEST_SCRIPT_TYPE, *type*	Specifies the type of test script generated by this Script Generator Adapter; ***type*** may be one of the following indicating, respectively, Java, Visual Basic, or VU: RSR_SCRIPT_TYPE_JAVA RSR_SCRIPT_TYPE_VISUAL_BASIC RSR_SCRIPT_TYPE_VU

Example

The following response indicates that this adapter:

- Generates Java test scripts.
- Uses datapools.

```
extern "C"
int LINKDLL GetOptions(TCHAR* options, size_t optionsSize)
{
   TCHAR buf[RSR_MAX_OPTIONS];
   _tcscpy(buf, _T(""));
   _tcscat(buf, TEST_SCRIPT_TYPE);
   _tcscat(buf, _T(",”));
   _tcscat(buf, _T(RSR_SCRIPT_TYPE_JAVA));
   _tcscat(buf, _T(";"));
   _tcscat(buf, GENERATOR_USE_DATAPOOLS);
   _tcscat(buf, _T(";"));

   if (_tcslen(buf) > optionsSize)
      return RSR_BUFFER_TOO_SHORT;
   _tcscpy(options, buf);
   return RSR_SUCCESS;
}
```

See Also

SetOptions()

SetOptions()

Sets user-specified configuration options for this adapter.

Syntax

```
int SetOptions (TCHAR *options, size_t optionsSize);
```

Element	Description
options	INPUT. Pointer to a read-only location containing the Robot user's selections.
optionsSize	INPUT. The size of *options*.

Comments

When the user selects or specifies a value for a Robot-defined option, this call communicates the user's choice to your adapter.

For Robot-defined options pertaining to script generation (those specified on the Generator tab of the Session Record Options dialog), this call communicates the user's choices using this format:

option[,choice][,*value*]

where:

- *option* is one of the option strings in column 1 of the options table: see GetOptions().
- If the option includes a check box, choice is 0 (not checked) or 1 (checked).
- If there is a data entry box(es), the entered *value*(s) appears after a preceding comma.

For example, if your adapter supports option GENERATOR_THINK and a user checks this option and specifies a maximum of 5 milliseconds, the *options* argument of SetOptions() contains this value: GENERATOR_THINK,1,5. If the user does not check this option, SetOptions() returns this value: GENERATOR_THINK,0,0. Options are separated from one another by semicolons.

This function returns the session file name, *sessionfile*, in this format:

GENERATOR_SESSION_NAME,sessionfile

You need this name in order to use the service call *GetAnnotations()* on page 70.

Example

This example checks to see whether a Robot user selected the **Think maximum (ms)** option.

```
extern "C"
int LINKDLL SetOptions(TCHAR* options, size_t optionsSize)
{
   /* CStringArray declared for parsed sub-strings */
   CStringArray OptionsArray;
```

```
   /* parse the original string with semi-colon delimeter.*/
   ParseString(options,';',&OptionsArray);

   for(int i = 0;i<OptionsArray.GetSize();i++)
   {
      /* for every sub-string, create another CStringArray*/
      CStringArray SubArray;
      if(!OptionsArray.GetAt(i).IsEmpty())
      {
         /*parse the substrings with comma delimeters */
         ParseString (OptionsArray.GetAt(i).GetBuffer
            (OptionsArray.GetAt(i).GetLength()),',',&SubArray);

         /* deal with each sub-string set */
         if(SubArray[0]==GENERATOR_THINK)
         {
            if(SubArray[1] == 0)
            {
            int think_min = SubArray[2];
         }
         else
         {
            /* Unrecognized option -- error may be thrown*/
         }
      }
   }
}
```

See Also

GetOptions()

Custom Adapter Reference 3

About Custom Adapters

This chapter documents the two APIs you use to implement custom recording and script generation adapters. These APIs are described in the following table:

API	Description
Custom Recorder Adapter API on page 43	A Custom Recorder Adapter is started when a user selects **Use Custom** from the Method tab on the Session Record Options dialog box. Requires a corresponding Custom Script Generator Adapter.
Custom Script Generator Adapter API on page 55	A Custom Script Generator Adapter generates a test script(s) from a session file recorded by a corresponding Custom Recorder Adapter.

Design Recommendation

A Custom Recorder Adapter can run as part of the multithreaded Robot session recorder process or as a separate process. If your recorder adapter is implemented in a language other than C, run it in a separate process. This design simplifies communication between the recorder adapter, script generator adapter, and other optional adapter components.

Custom Recorder Adapter API

Custom Recorder Adapters implement the following calls. The shaded rows list functions that other adapter types also implement.

Function	Description
`IsCustomRecorderAdapter()`	Identifies a Custom Recorder Adapter.
`InitializeRecorder()`	Performs initialization procedures.

Function	Description
`StartRecording()`	Starts recording a session.
`StopRecording()`	Concludes a recording session.
`GetDisplayName()`	Returns the public name of this adapter.
`GetOptions()`	Returns configuration options in effect for this adapter.
`SetOptions()`	Sets user-specified configuration options for this adapter.
`DisplayCustomConfigGUI()`	Provides a custom GUI for adapter-defined configuration options.

IsCustomRecorderAdapter()

Identifies a Custom Recorder Adapter.

Syntax

```
BOOL IsCustomRecorderAdapter();
```

Comments

Return TRUE to indicate that this is a Custom Recorder Adapter. Any other response disables the adapter.

Example

A C adapter should respond to this call as illustrated below.

```
extern "C"
BOOL LINKDLL IsCustomRecorderAdapter()
{
   return TRUE;
}
```

See Also

`IsCustomScriptgenAdapter()`

InitializeRecorder()

Performs initialization procedures.

Syntax

```
int InitializeRecorder (TCHAR *errorMessage, size_t
   errorMessageSize);
```

Element	Description
errorMessage	Pointer to a container for a message to be displayed in case of an initialization error. Copy the message to this location.
errorMessageSize	INPUT. The size allocated for *errorMessage*, which cannot exceed this size.

Comments

Initialization procedures are optional and adapter-defined. A return of RSR_SUCCESS indicates that the adapter is prepared to begin recording a session on request.

Example

```
extern "C"
int LINKDLL InitializeRecorder (TCHAR *errorMessage, size_t
errorMessageSize)
{
   //perform initialization procedures
   if (successful)
      return RSR_SUCCESS
   else
   {
      TCHAR buf[] = "Recorder failed to initialize";
      if (_tcslen(buf) >= errorMessageSize)
         return RSR_BUFFER_TOO_SHORT;

      _tcscpy(errorMessage, buf);
      return RSR_FAILURE;
   }
}
```

See Also

StartRecording(), StopRecording()

StartRecording()

Starts recording a session.

Syntax

```
int StartRecording (TCHAR *sessionPath, size_t sessionPathSize,
   StatusCallbackPtr fPtr, TCHAR *errorMessage, size_t
   errorMessageSize);
```

Element	Description
sessionPath	INPUT. Pointer to a read-only location containing the session file's path name, without extension.
sessionPathSize	INPUT. The size allocated of *sessionPath*.

fPtr	INPUT. Pointer to a Robot-defined callback function that communicates the adapter's status. The function has this signature:
	`fPtr (msgType, "status message");`
	msgType may be one of the following:
	`RSR_CALLBACK_PROGRESS`
	`RSR_CALLBACK_STOP`
	`RSR_CALLBACK_ERROR`.

errorMessage	Pointer to a container for a message to be displayed if recording fails to start. Copy the message to this location.
errorMessageSize	INPUT. The size allocated for *errorMessage*, which cannot exceed this size.

Example

This example starts recording into an XML session file.

```
extern "C"
int LINKDLL StartRecording(TCHAR* pathname,
                           size_t pathnameSize,
                           StatusCallbackPtr fPtr,
                           TCHAR* errorMessage,
                           size_t errorMessageSize)
{
    TCHAR buf[1024];
    _tcscpy (buf, "Start recording to: ");
    _tcscat(buf, pathname);
    MessageBox(NULL, buf,"", MB_OK);

    TCHAR fileName[1024];
    _tcscpy(fileName, pathname);
    _tcscat(fileName, ".xml");
    ofstream of (fileName);
    if (of)
        of << "<?xml version=\"1.0\" ?>\n<Sample>\n </Sample>" << endl;

    fPtr(RSR_CALLBACK_PROGRESS, "I am doing OK");
    return RSR_SUCCESS;
}
```

See Also

InitializeRecorder(), `StopRecording()`

StopRecording()

Concludes a recording session.

Syntax

```
int StopRecording (TCHAR *errorMessage, size_t
   errorMessageSize);
```

Element	Description
errorMessage	Pointer to a container for a message to be displayed in case there is a cleanup error. Copy the message to this location.
errorMessageSize	INPUT. The size allocated for *errorMessage*, which cannot exceed this size.

Comments

This call is made when a Robot user ends a recording session. Stop recording and return `RSR_SUCCESS`.

If you return `RSR_BUFFER_TOO_SHORT` to indicate that the initial *errorMessageSize* is too small, Robot loops until *errorMessageSize* is large enough to contain *errorMessage*.

Example

```
extern "C"
int LINKDLL StopRecording(TCHAR* errorMessage,
                          size_t errorMessageSize)
{
   //stop recording
   if (successful)
      return RSR_SUCCESS
   else
   {
      TCHAR buf[] = "Couldn't stop!!!";
      if (_tcslen(buf) >= errorMessageSize)
         return RSR_BUFFER_TOO_SHORT;

      _tcscpy(errorMessage, buf);
      return RSR_FAILURE;
   }
}
```

See Also

InitializeRecorder(), `StartRecording()`

GetDisplayName()

Returns the public name of this adapter.

Syntax

```
int GetDisplayName (TCHAR *name, size_t nameSize);
```

Element	Description
name	Pointer to a container for the adapter's display name. Copy the name to this location.
nameSize	INPUT. The size allocated for *name*. The adapter's display name cannot exceed this size.

Comments

Specify the GUI name for this adapter. This name is presented to the Robot user (in the **Recorder** list box beside the **Use Custom** radio box) on the Method tab of the Session Record Options dialog.

Example

This example specifies that the GUI name of this adapter is `rtweblogicEJB`.

```
extern "C"
int LINKDLL GetDisplayName (TCHAR *name, size_t nameSize)
{
   TCHAR buf[] = "rtweblogicEJB";
   if (_tcslen(buf) > nameSize)
      return RSR_BUFFER_TOO_SHORT;

   _tcscpy(name, buf);
   return RSR_SUCCESS;
}
```

GetOptions()

Returns configuration options in effect for this adapter.

Syntax

```
int GetOptions (TCHAR *options, size_t optionsSize);
```

Element	Description
options	Pointer to a container for this adapter's options. Copy supported options, separated by semicolons, to this location. Robot-defined options have the format: argument[,setting] where *argument* is one of the strings described in the options table shown below, and *setting* can be a value for the argument. Adapter-defined options have the format: *name*,*value*,*description*[,*value1*, *value2*, ...*valuen*] Adapter-defined options that you return in response to this call appear on the Method:Custom tab of the Robot Session Record Options dialog. They can also appear on an adapter-supplied GUI.
optionsSize	INPUT. The size allocated by Robot for *options*, which must not exceed this size.

Comments

As illustrated in the example, options supported by an adapter should be entered from a saved, local file. Otherwise, they do not persist between sessions.

Your adapter can define a custom format for options and provide a custom GUI for displaying and editing them and code to communicate the user's choices to the adapter. Do not include custom-format options in your response to this call.

The following table describes the Robot-defined configuration option arguments that a Custom Recorder Adapter can support. See *Adapter Configuration* on page 81 for a mapping of these options to the Robot GUI.

Configuration Option	Description
CONFIGURATION, USE_CUSTOM_UI	Specifies that the adapter provides a custom GUI for displaying and selecting adapter-defined configuration options.
CONFIGURATION, *name*, *value*, *description*[,*value1*, *value2* ...]	The adapter supports a configuration option of the specified ***name*** and ***value***, which works as indicated by the ***description***. Adapter-defined options may be entered either from the Method:Custom tab of the Session Record Options dialog or from a supplied custom GUI. If a configuration triplet includes [, value1, value2, ...] , the supplied values are implemented by a ***value*** pull-down menu on the grid.

Configuration Option	Description
DEFAULT_SCRIPT_GENERATOR, *sga*	Specifies the name of the companion Custom Script Generator Adapter for this Custom Recorder Adapter. Enter the adapter's display name as specified with `GetDisplayName()`.
GENERAL_START_APP_PROMPT	On the General tab of the Session Record Options dialog, the **Prompt for application name on start recording** check box is enabled.
RECORD_BLOCK	On the Session Insert dialog and the **Insert** menu, the **Start Block** and **Stop Block** options are enabled.
RECORD_COMMENT	On the Session Insert dialog and the **Insert** menu, the **Comment** option is enabled.
RECORD_SPLITS	On the Session Record dialog, the **Split Script** icon is enabled.
RECORD_SYNC_PT	On the Session Insert dialog and the **Insert** menu, the **Sync point** option is enabled.
RECORD_TIMER	On the Session Insert dialog and the **Insert** menu, the **Start timer** and **Stop timer** options are enabled.
RECORD_START_APP	On the Session Insert dialog and the **Insert** menu, the **Start Application** option is enabled.
SESSION_FILES, *format*	Specifies the file format(s) and extension(s) of the session file(s), which can be one or more of the following: ▪ RSR_SESSION_FILE_EXT — user-defined type, extension .ext. ▪ RSR_SESSION_FILE_EXT — XML format, extension .xml. (The BEA WebLogic recorder uses this format). ▪ RSR_SESSION_FILE_EXT — Rational's proprietary trace file format (called a watch file), extension .wch. API Recorder Adapters use this format.

Example

The following response indicates that this adapter:

- Creates an XML trace file.
- Is used with a Script Generator Adapter named mySGA.

- Supports split scripts.

```
extern "C"
int LINKDLL GetOptions(TCHAR* options, size_t optionsSize)
{
   TCHAR buf[RSR_MAX_OPTIONS];
   _tcscpy(buf, _T(""));
   _tcscat(buf, SESSION_FILES);
   _tcscat(buf, _T(",xml"));
   _tcscat(buf, _T(";"));
   _tcscat(buf, DEFAULT_SCRIPT_GENERATOR);
   _tcscat(buf, _T(",mySGA"));
   _tcscat(buf, _T(";"));
   _tcscat(buf, RECORD_SPLITS);
   _tcscat(buf, _T(";"));

   if (_tcslen(buf) > optionsSize)
      return RSR_BUFFER_TOO_SHORT;
   _tcscpy(options, buf);
   return RSR_SUCCESS;
}
```

See Also

SetOptions()

SetOptions()

Sets user-specified configuration options for this adapter.

Syntax

```
int SetOptions (TCHAR *options, size_t optionsSize);
```

Element	Description
options	INPUT. Pointer to a read-only location containing the Robot user's selections.
optionsSize	INPUT. The size of *options*.

Comments

When the user selects or specifies a value for a Robot-defined option or edits an adapter-defined option, this call communicates the user's choice to your adapter.

This call also returns a user's choices for adapter-defined options, in the triplet format, that were selected from a Robot-provided dialog. However, if you use a custom GUI for displaying and editing custom options, you are responsible for reading the dialog, conveying the user's choices to the adapter, parsing, validation, and sending an appropriate error message for invalid user specifications.

Example

This example checks to see whether a Robot user selected the **Think maximum (ms)** option.

```
extern "C"
int LINKDLL SetOptions(TCHAR* options, size_t optionsSize)
{
   /* CStringArray declared for parsed sub-strings */
   CStringArray OptionsArray;

   /* parse the original string with semi-colon delimeter.*/
   ParseString(options,';',&OptionsArray);

   for(int i = 0;i<OptionsArray.GetSize();i++)
   {
      /* for every sub-string, create another CStringArray*/
      CStringArray SubArray;
      if(!OptionsArray.GetAt(i).IsEmpty())
      {
         /*parse the substrings with comma delimeters */
         ParseString (OptionsArray.GetAt(i).GetBuffer
            (OptionsArray.GetAt(i).GetLength()),',',&SubArray);

         /* deal with each sub-string set */
         if(SubArray[0]==GENERATOR_THINK)
         {
            if(SubArray[1] == 0)
            {
            int think_min = SubArray[2];
         }
         else
         {
            /* Unrecognized option -- error may be thrown*/
         }
      }
   }
}
```

See Also

`GetOptions()`

DisplayCustomConfigGUI()

Provides a custom GUI for adapter-defined configuration options.

Syntax

```
int DisplayCustomConfigGUI (TCHAR *errorMessage, size_t
   errorMessageSize);
```

Element	Description
errorMessage	Pointer to a location for a message to be displayed in the event of error. Copy the message to this location.
errorMessageSize	INPUT. The size allocated for *errorMessage*, which cannot exceed this size.

Comments

If your adapter specifies the option CONFIGURATION,USE_CUSTOM_UI, a **Configure** button on the Method:Custom tab of the Session Record Options dialog is enabled. If a user clicks this button, Robot issues this call. In response, your adapter should display a custom GUI for entering or editing custom configuration options.

If you provide a custom GUI:

- The format of custom configuration options is entirely up to you: it is not necessary to include custom options with your response to GetOptions(), and the options need not adhere to the triplet format defined by Robot for custom options.
- If you do adhere to the triplet format for custom options and include them in your response to GetOptions(), the Robot user can use a provided triplet grid as well as your custom GUI.
- If you do not adhere to the triplet format for custom options, do not include them in your response to GetOptions(). Doing so causes an error because the format is not understood.
- Custom options chosen or edited using the Robot-supplied triplet grid are conveyed to an adapter by SetOptions(). With a custom GUI, you are responsible for reading and persisting user choices.
- If the GUI you provide includes online Help, it works as implemented. In any case, you are responsible for providing any documentation that users require.

Example

```
extern "C"
int LINKDLL DisplayCustomConfigGUI (TCHAR *errorMessage, size_t
errorMessageSize)
{
   //display custom GUI and gather user input
   if (successful)
      return RSR_SUCCESS;
   else
   {
      TCHAR buf[] = "Custom GUI failed to start";
      if (_tcslen(buf) > errorMessageSize)
         return RSR_BUFFER_TOO_SHORT;

      _tcscpy(errorMessage, buf);
      return RSR_FAILURE;
   }
}
```

See Also

GetOptions(),SetOptions()

Custom Script Generator Adapter API

Use this API to develop a script generator adapter to be used with a Custom Recorder Adapter. Use *API Script Generator Adapter API* on page 34 to develop a script generator adapter to be used with the API recording method.

Custom Script Generator Adapters implement the following calls. The shaded rows list functions that other adapter types also implement.

Function	Description
IsCustomScriptgenAdapter()	Identifies a Custom Script Generator Adapter.
InitializeScriptgen()	Performs initialization procedures.
StartScriptgen()	Starts a script generation session.
CancelScriptgen()	Cancels a script generation request.
GetDisplayName()	Returns the public name of this adapter.
GetOptions()	Returns configuration options in effect for this adapter.
SetOptions()	Sets user-specified configuration options for this adapter.
DisplayCustomConfigGUI()	Provides a custom GUI for adapter-defined configuration options.

IsCustomScriptgenAdapter()

Identifies a Custom Script Generator Adapter.

Syntax

```
BOOL IsCustomScriptgenAdapter();
```

Comments

Return TRUE to indicate that this is a Custom Script Generator Adapter. Any other response disables the adapter.

Example

A C adapter should respond to this call as illustrated below.

```
extern "C"
BOOL LINKDLL IsCustomScriptgenAdapter()
{
```

```
   return TRUE;
}
```

See Also

`IsCustomRecorderAdapter()`

InitializeScriptgen()

Performs initialization procedures.

Syntax

```
int InitializeScriptgen (TCHAR *errorMessage, size_t
   errorMessageSize);
```

Element	Description
errorMessage	Pointer to a container for a message to be displayed in case there is an initialization error. Copy the message to this location.
errorMessageSize	INPUT. The size allocated for *errorMessage*, which cannot exceed this size.

Comments

In response to this call, perform any needed initialization procedures and return `RSR_SUCCESS`.

Example

```
extern "C"
int LINKDLL InitializeScriptgen (TCHAR *errorMessage, size_t
errorMessageSize)
{
   //perform initialization procedures
   if (successful)
      return RSR_SUCCESS
   else
   {
      TCHAR buf[] = "Recorder failed to initialize";
      if (_tcslen(buf) >= errorMessageSize)
         return RSR_BUFFER_TOO_SHORT;

      _tcscpy(errorMessage, buf);
      return RSR_FAILURE;
   }
}
```

See Also

`CancelScriptgen(),InitializeRecorder(),StartScriptgen()`

StartScriptgen()

Starts a script generation session.

Syntax

```
int StartScriptgen (TCHAR *sessionPath, size_t sessionPathSize,
   TCHAR *scriptPath, size_t scriptPathSize, StatusCallbackPtr
   fPtr, TCHAR *errorMessage, size_t errorMessageSize);
```

Element	Description
sessionPath	INPUT. Pointer to a location containing the session file's path name, without extension.
sessionPathSize	INPUT. The size allocated for *sessionPath*, which cannot exceed this size.
scriptPath	INPUT. Pointer to a location for the script file's path name, without extension.
scriptPathSize	INPUT. The size allocated for *scriptPath*, which cannot exceed this size.

fPtr	Pointer to a Robot-defined callback function for communication of completion status. The function has this signature:
	`fPtr (msgType,"status message");`
	msgType can be one of the following:
	`RSR_CALLBACK_DETAILS`
	`RSR_CALLBACK_ERROR`
	`RSR_CALLBACK_FINISHED`
	`RSR_CALLBACK_PROGRESS`
	`RSR_CALLBACK_STOP`
	As shown in the example, with *msgType* RSR_CALLBACK_PROGRESS, the status message is a number between 0 and 100, indicating percent completed, formatted as a string.

errorMessage	Pointer to a location for a message to be displayed in case of a startup error.
errorMessageSize	INPUT. The size allocated for *errorMessage*, which cannot exceed this size.

Comments

Respond to this call by starting script generation and returning `RSR_SUCCESS`.

Example

The following example provides progress information in 10% intervals and parses for multiple script names.

```
extern "C"
int LINKDLL StartScriptgen(TCHAR* pathname,
                           size_t pathnameSize,
                           TCHAR* scriptFilePathnames,
                           size_t scriptFilePathnamesSize,
                           StatusCallbackPtr fPtr,
                           TCHAR* errorMessage,
                           size_t errorMessageSize)
{


   CStringArray ScriptNames;

   for (int i=1; i<=10; i++)
     {
         TCHAR progress[5];
      sprintf(progress,"%d",i*10);
      fPtr(RSR_CALLBACK_PROGRESS, progress);
         Sleep(1000);
     }


   ParseString(scriptFilePathnames,';',&ScriptNames);


   for(i = 0;i<ScriptNames.GetSize();i++)
   {
      CopyFile("c:\\seed.java",ScriptNames[i]+".java",FALSE);
   }
     fPtr(RSR_CALLBACK_FINISHED, "Scriptgen successful!!!");

     return RSR_SUCCESS;
}
```

See Also

`CancelScriptgen(),InitializeScriptgen(),StartRecording()`

CancelScriptgen()

Cancels a script generation request.

Syntax

```
int CancelScriptgen (TCHAR *errorMessage, size_t
   errorMessageSize);
```

Element	Description
errorMessage	Pointer to a container for a message to be displayed in case of a cleanup error. Copy the message to this location.
errorMessageSize	INPUT. The size allocated for *errorMessage*, which cannot exceed this size.

Comments

This call is made when a Robot user cancels script generation, or in case of a system error. On receiving the call, stop script generation as soon as possible, perform cleanup operations, and return RSR_SUCCESS.

Example

```
extern "C"
int LINKDLL CancelScriptgen(TCHAR* errorMessage,
                            size_t errorMessageSize)
{
   //stop scriptgen and perform cleanup
   if (successful)
      return RSR_SUCCESS
   else
   {
      TCHAR msg[] = "Cancellation cleanup failed";
      if (_tcslen(buf) >= errorMessageSize)
         return RSR_BUFFER_TOO_SHORT;

      _tcscpy(errorMessage, buf);
      return RSR_FAILURE;
   }
}
```

See Also

InitializeScriptgen(),StartScriptgen()

GetDisplayName()

Returns the public name of this adapter.

Syntax

```
int GetDisplayName (TCHAR *name, size_t nameSize);
```

Element	Description
name	Pointer to a container for the adapter's display name. Copy the name to this location.
nameSize	INPUT. The size allocated for *name*. The adapter's display name cannot exceed this size.

Comments

Specify the GUI name for this adapter. This name is presented to the Robot user (in the **Script Generator** list box beside the **Use Custom** radio box) on the Method tab of the Session Record Options dialog.

Example

This example specifies that the GUI name of this adapter is `rtweblogicEJB`.

```
extern "C"
int LINKDLL GetDisplayName (TCHAR *name, size_t nameSize)
{
   TCHAR buf[] = "rtweblogicEJB";
   if (_tcslen(buf) > nameSize)
      return RSR_BUFFER_TOO_SHORT;

   _tcscpy(name, buf);
   return RSR_SUCCESS;
}
```

GetOptions()

Returns configuration options in effect for this adapter.

Syntax

```
int GetOptions (TCHAR *options, size_t optionsSize);
```

Element	Description
options	Pointer to a container for this adapter's options. Copy supported options, separated by semicolons, to this location. Robot-defined options have the format: `argument[,setting]` where *argument* is one of the strings described in the options table shown below, and *setting* can be a value for the argument. Adapter-defined options have the format: *name*,*value*,*description*[,*value1*, *value2*, ...*valuen*] Adapter-defined options that you return in response to this call appear on the Generator:Custom tab of the Robot Session Record Options dialog. They can also appear on an adapter-supplied GUI.
optionsSize	INPUT. The size allocated by Robot for *options*, which must not exceed this size.

Comments

As illustrated in the example, options supported by an adapter should be entered from a saved, local file. Otherwise, they do not persist between sessions.

Your adapter can define a custom format for options and provide a custom GUI for displaying and editing them and code to communicate the user's choices to the adapter. Do not include custom-format options in your response to this call.

The following table describes the Robot-defined configuration option arguments that a Script Generator Adapter can support. See *Adapter Configuration* on page 81 for a mapping of these options to the Robot GUI.

Configuration Option	Description
`CONFIGURATION,USE_CUSTOM_UI`	Specifies that the adapter provides a custom GUI for displaying and selecting adapter-defined configuration options.
`CONFIGURATION,` *name*, *value*, *description*`[,`*value1*, *value2* ...`]`	The adapter supports a configuration option of the specified ***name*** and ***value***, which works as indicated by the ***description***. Adapter-defined options may be entered either from the Generator:Customtab of the Session Record Options dialog or from a supplied custom GUI. If a configuration triplet includes [, value1, value2, ...] , the supplied values are implemented by a ***value*** pull-down menu on the grid.

Configuration Option	Description
GENERATOR_BIND_VU_VARS	On the Generator tab of the Session Record Options dialog, the **Bind output parameters to VU variables** check box is enabled.
GENERATOR_COMMAND_ID	On the Generator tab of the Session Record Options dialog, the **Command ID prefix** box is enabled.
GENERATOR_CPU_THRESH	On the Generator tab of the Session Record Options dialog, the **CPU/user threshold (ms)** check box is enabled.

GENERATOR_DISPLAY_ROWS	On the Generator tab of the Session Record Options dialog, the **Display recorded rows** boxes are enabled. The Robot user's choices are communicated by `SetOptions()` in the format:
	`GENERATOR_DISPLAY_ROWS, choice, rows`
	where:
	▪ *choice* is one of these values corresponding to the user's selection of **None**, **First**, **Last**, or **All**:
	`RSR_DISPLAY_RECORDED_ROWS_NONE`
	`RSR_DISPLAY_RECORDED_ROWS_FIRST`
	`RSR_DISPLAY_RECORDED_ROWS_LAST`
	`RSR_DISPLAY_RECORDED_ROWS_ALL`
	▪ *rows* is 0 (for All or None) or the specified number of rows.

GENERATOR_PLAYBACK_PACING	On the Generator tab of the Session Record Options dialog, the **Playback pacing** radio boxes are enabled. The Robot user's choice of **per command**, **per script**, or **none** is communicated by `SetOptions()` as:
	`RSR_GENERATOR_PLAYBACK_PACING_COMMAND`
	`RSR_GENERATOR_PLAYBACK_PACING_SCRIPT`
	`RSR_GENERATOR_PLAYBACK_PACING_NONE`

GENERATOR_THINK	On the Generator tab of the Session Record Options dialog, the **Think maximum (ms)** check box is enabled.
GENERATOR_USE_DATAPOOLS	On the Generator tab of the Session Record Options dialog, the **Use datapools** check box is enabled.

Configuration Option	Description
GENERATOR_USE_SCRIPTGEN_PROGRESS	Specifies that the adapter provides progress information to be displayed by a Robot progress dialog.
GENERATOR_VERIFY_RETURN_CODES	On the Generator tab of the Session Record Options dialog, the **Verify playback return codes** check box is enabled.
GENERATOR_VERIFY_ROW_COUNTS	On the Generator tab of the Session Record Options dialog, the **Verify playback row counts** check box is enabled.
TEST_SCRIPT_TYPE, *type*	Specifies the type of test script generated by this Script Generator Adapter; ***type*** may be one of the following indicating, respectively, Java, Visual Basic, or VU: RSR_SCRIPT_TYPE_JAVA RSR_SCRIPT_TYPE_VISUAL_BASIC RSR_SCRIPT_TYPE_VU

Example

The following response indicates that this adapter:

- Generates Java test scripts.
- Uses datapools.

```
extern "C"
int LINKDLL GetOptions(TCHAR* options, size_t optionsSize)
{
   TCHAR buf[RSR_MAX_OPTIONS];
   _tcscpy(buf, _T(""));
   _tcscat(buf, TEST_SCRIPT_TYPE);
   _tcscat(buf, _T(","));
   _tcscat(buf, _T(RSR_SCRIPT_TYPE_JAVA));
   _tcscat(buf, _T(";"));
   _tcscat(buf, GENERATOR_USE_DATAPOOLS);
   _tcscat(buf, _T(";"));

   if (_tcslen(buf) > optionsSize)
      return RSR_BUFFER_TOO_SHORT;
   _tcscpy(options, buf);
   return RSR_SUCCESS;
}
```

See Also

SetOptions()

SetOptions()

Sets user-specified configuration options for this adapter.

Syntax

```
int SetOptions (TCHAR *options, size_t optionsSize);
```

Element	Description
options	INPUT. Pointer to a read-only location containing the Robot user's selections.
optionsSize	INPUT. The size of *options*.

Comments

When the user selects or specifies a value for a Robot-defined option or edits an adapter-defined option, this call communicates the user's choice to your adapter.

For Robot-defined options pertaining to script generation (those specified on the Generator tab of the Session Record Options dialog), this call communicates the user's choices using this format:

option[,choice][,*value*]

where:

- *option* is one of the option strings in column 1 of the options table: see GetOptions().
- If the option includes a check box, choice is 0 (not checked) or 1 (checked).
- If there is a data entry box(es), the entered *value*(s) appears after a preceding comma.

For example, if your adapter supports option GENERATOR_THINK and a user checks this option and specifies a maximum of 5 milliseconds, the *options* argument of SetOptions() contains this value: GENERATOR_THINK,1,5. If the user does not check this option, SetOptions() returns this value: GENERATOR_THINK,0,0. Options are separated from one another by semicolons.

This call also returns a user's choices for adapter-defined options, in the triplet format, that were selected from a Robot-provided dialog. However, if you use a custom GUI for displaying and editing custom options, you are responsible for reading the dialog, conveying the user's choices to the adapter, parsing, validation, and sending an appropriate error message for invalid user specifications.

Example

This example checks to see whether a Robot user selected the **Think maximum (ms)** option.

```
extern "C"
int LINKDLL SetOptions(TCHAR* options, size_t optionsSize)
{
   /* CStringArray declared for parsed sub-strings */
   CStringArray OptionsArray;

   /* parse the original string with semi-colon delimeter.*/
   ParseString(options,';',&OptionsArray);

   for(int i = 0;i<OptionsArray.GetSize();i++)
   {
      /* for every sub-string, create another CStringArray*/
      CStringArray SubArray;
      if(!OptionsArray.GetAt(i).IsEmpty())
      {
         /*parse the substrings with comma delimeters */
         ParseString (OptionsArray.GetAt(i).GetBuffer
            (OptionsArray.GetAt(i).GetLength()),',',&SubArray);

         /* deal with each sub-string set */
         if(SubArray[0]==GENERATOR_THINK)
         {
            if(SubArray[1] == 0)
            {
            int think_min = SubArray[2];
         }
         else
         {
            /* Unrecognized option -- error may be thrown*/
         }
      }
   }
}
```

See Also

`GetOptions()`

DisplayCustomConfigGUI()

Provides a custom GUI for adapter-defined configuration options.

Syntax

```
int DisplayCustomConfigGUI (TCHAR *errorMessage, size_t
   errorMessageSize);
```

Element	Description
errorMessage	Pointer to a location for a message to be displayed in the event of error. Copy the message to this location.
errorMessageSize	INPUT. The size allocated for *errorMessage*, which cannot exceed this size.

Comments

If your adapter specifies the option CONFIGURATION,USE_CUSTOM_UI, a **Configure** button on the Generator per Protocol tab of the Session Record Options dialog is enabled. If a user clicks this button, Robot issues this call. In response, your adapter should display a custom GUI for entering or editing custom configuration options.

If you provide a custom GUI:

- The format of custom configuration options is entirely up to you: it is not necessary to include custom options with your response to GetOptions(), and the options need not adhere to the triplet format defined by Robot for custom options.
- If you do adhere to the triplet format for custom options and include them in your response to GetOptions(), the Robot user can use a provided triplet grid as well as your custom GUI.
- If you do not adhere to the triplet format for custom options, do not include them in your response to GetOptions(). Doing so causes an error because the format is not understood.
- Custom options chosen or edited using the Robot-supplied triplet grid are conveyed to an adapter by SetOptions(). With a custom GUI, you are responsible for reading and persisting user choices.
- If the GUI you provide includes online Help, it works as implemented. In any case, you are responsible for providing any documentation that users require.

Example

```
extern "C"
int LINKDLL DisplayCustomConfigGUI (TCHAR *errorMessage, size_t
errorMessageSize)
{
   //display custom GUI and gather user input
   if (successful)
      return RSR_SUCCESS;
   else
   {
```

```
        TCHAR buf[] = "Custom GUI failed to start";
        if (_tcslen(buf) > errorMessageSize)
          return RSR_BUFFER_TOO_SHORT;

        _tcscpy(errorMessage, buf);
        return RSR_FAILURE;
      }
    }
```

See Also

GetOptions(),SetOptions()

Recording and Script Generation Services 4

About Recording and Script Generation Services

This chapter documents functions required by some types of adapters. It contains these sections:

- *Proxy Services Reference* on page 69. These functions are needed by API recorder DLLs.
- *The GetAnnotations() Service Function* on page 78. This function is needed by Custom Script Generator Adapters that support the features described in *Script Generation Options* on page 83, and by all API Script Generator Adapters (which are required to support these features).

Proxy Services Reference

Extending API recording requires the writing of a recording component (proxy library) corresponding to the targeted library that implements the functions to be recorded. Traffic between the AUT and target library that is to be recorded passes through the proxy library. The job of the functions in the proxy library is to decide what needs to be recorded and to pass this data to Rational's rtxvutl.dll, which adds the data to the session file. The proxy library and API Recorder Adapter components must reside in the same DLL.

This chapter documents the calls needed to pass data to rtvutl.dll. The API recording services are listed below, in the order in which they are used. See *Proxy Examples* on page 76 for examples illustrating these calls.

Function	Description
ProxyGetAssignedLibraryID()	Gets an ID for the specified target DLL.
ProxyGetTicket()	Gets an ID for a proxy function to be recorded.
ProxyGetTimeStamp()	Gets a time stamp for a proxy function to be recorded.
ProxyLockNew()	Locks the session file prior to a write operation.

Function	Description
ProxyWriteBlock()	Writes a block to the session file.
ProxyUnlock()	Unlocks the session file after a write operation.
ProxyExceptionHandler()	Returns the library ID and function associated with an error.

ProxyGetAssignedLibraryID()

Gets an ID for the specified target DLL.

Syntax

```
int ProxyGetAssignedLibraryID(char *DLLName)
```

Element	Description
DLLName	Specifies the name of the target DLL.

Return Value

On success, this function returns an ID for the named DLL.

Comments

Unlike the other proxy calls, which are called with each write operation, this call is made only during session initialization and so is likely to appear in a different DLL. The returned ID is the *libraryID* argument to *ProxyLockNew()* on page 72.

Example

This example illustrates the use of this call in an initialization block.

```
#include <windows.h>
#include <tchar.h>
#include "SampleAPIWrapper.h"
#include "proxutil.h"

// initialize global for library ID
int gbLibID = 0;

int WINAPI DllMain(HANDLE hInstance,
              ULONG ul_reason_being_called,
              LPVOID lpReserved)
{
   switch (ul_reason_being_called)
   {
```

```
      case DLL_PROCESS_ATTACH:
      {
         // MUST call the following in DLL_PROCESS_ATTACH
         // this function is provided by the Utility DLL
         // rtxutil.dll

         // the name passed-in must be the exact name of
         // your wrapper DLL
         gbLibID = ProxyGetAssignedLibraryID("SampleAPIWrapper.dll");

         // add whatever initialization you need here
         break;
      }
case DLL_PROCESS_DETACH:
      {
         // add whatever cleanup you may need here
         break;
      }
      default:
      {
         break;
      }
   }
   return 1;
}
```

ProxyGetTicket()

Gets an ID for a proxy function to be recorded.

Syntax

```
DWORD ProxyGetTicket(void)
```

Return Value

On success, this function returns the ID uniquely identifying a packet to be recorded. On failure, it returns `PROXY_INVALID_TICKET`.

Comments

This call begins a write operation inside a proxy function, which should proceed only if the call succeeds. The returned ID is an input argument to *ProxyLockNew()* on page 72. On failure, the correct behavior is to call the real function and return without recording anything.

Example

See *Proxy Examples* on page 76.

ProxyGetTimeStamp()

Gets a time stamp for a proxy function to be recorded.

Syntax

```
DWORD ProxyGetTimeStamp(void)
```

Return Value

On success, this function returns the number of milliseconds that have elapsed since the session file was initialized.

Comments

This value returned by this function is an argument of *ProxyLockNew()* on page 72.

Example

See *Proxy Examples* on page 76.

ProxyLockNew()

Locks the session file prior to a write operation.

Syntax

```
void *ProxyLockNew (int ticketID, DWORD msec, int libraryID,
   int proxyID, int PacketType, int packetSize)
```

Element	Description
ticketID	The ID of the packet to be written to the .wch file. Returned by ProxyGetTicket().
msec	The number of milliseconds elapsed since the session file was initialized. Returned by ProxyGetTimeStamp()
libraryID	The ID of the library associated with this proxy call. Returned by ProxyGetAssignedLibraryID()
proxyID	The ID of this proxy call.

Element	Description
packetType	One of the following: ▪ API_ENTRY. Begin a record in a badly written API that overwrites input data parameters. ▪ API_EXIT. End a record. ▪ API_SINGLE. Produce a single record for each proxy function. Normally specify API_SINGLE. If used, API_ENTRY and API_EXIT must both be used in order to get a complete record.
packetSize	The number of bytes to be written.

Comments

The *proxyID* is a unique identifier for each function that you record. Typically, you start with 1 and increment *proxyID* for each function to be recorded. This argument should be defined in a header file common to the API Recorder, Generator Filter, and API Script Generator adapters, such that the combination of *libraryID* and *proxyID* uniquely identifies a single proxy function and what it pertains to.

Example

See page 76 for an API_SINGLE example; see page 77 for an API_ENTRY/API_EXIT example.

ProxyWriteBlock()

Writes a block to the session file.

Syntax

```
int ProxyWriteBlock (LPVOID, int iSize)
```

element	Description
LpData	Pointer to the data to be written to the session file.
iSize	The number of bytes to write.

Return Value

On success, the function returns the offset from the most recent ProxyLockNew() call to the beginning of data for the current ProxyWriteBlock() call.

Example

See *Proxy Examples* on page 76.

ProxyUnlock()

Unlocks the session file after a write operation.

Syntax

```
void ProxyUnlock (DWORD msec)
```

Element	Description
msec	Specify as 0 for the default behavior, described below: ▪ API_SINGLE. ProxyGetTimestamp() is called and the result placed in the session file header. ▪ API_ENTRY and API_EXIT. The original time stamp passed with ProxyLock is placed in the session header. Alternatively, you can enter a number of milliseconds to place in the session header.

Example

See *Proxy Examples* on page 76.

ProxyExceptionHandler()

Returns the library ID and function associated with an error.

Syntax

```
void ProxyExceptionHandler (type proxyLib, type proxyFunction)
```

Element	Description
proxyLib	The proxy library ID.
proxyFunction	The proxy function ID.

Example

See *Proxy Examples* on page 76.

Proxy Data Types

To use the proxy utility functions, you must include \Suite\include\proxhdr.h, whose structures are as follows:

```
#define API_ENTRY1
#define API_EXIT  2
#define API_SINGLE3

/* All three have a common header: */

typedef struct {
  unsigned char  major_type;/* API_ENTRY, API_EXIT, API_SINGLE, WCH_V1
*/
  unsigned char  spare_uchar;/* Future */
  unsigned short spare_ushort;/* Future */
  unsigned int   ticket;/* Unique ticket for each API call */
  unsigned int   timestamp;/* Entry TS for all but API_EXIT (api
return) */
  unsigned int   total_length;/* Total length of packet including ALL
headers */
} major_wch_header;

/* API_EXIT has no secondary header */

/* API_ENTRY and API_SINGLE supply the following as a second header: */
typedef struct {
   unsigned int   process_id;/* Process ID of API_proxy */
   unsigned int   thread_id;/* Thread ID of API_proxy */
   unsigned int   library_id;/* Library identifier of proxied API */
   unsigned int   API_id;/* Unique id for each proxied API */
} minor_wch_header;

/* API_SINGLE will supply a third header following the second header:
*/

typedef struct {
  unsigned int   timestamp;/* api return timestamp for API_SINGLE */
  unsigned int   alignment_uint;/* To maintain double word alignment
*/
} stamp2_wch_header;

/*
Packets will be generated for an API_proxy as either a pair of
API_ENRTY
and API_EXIT sharing a common ticket number OR one API_SINGLE.
*/
```

The above structures, including the process ID, thread ID, and total length of the proxy record, are filled in for you automatically.

Proxy Examples

Following are examples illustrating API_SINGLE and API _ENTRY/API_EXIT. In this examples, `pMyProxyFunc()` is the name of the proxy function and `RealFunc()` is the name of the actual function in the target DLL that is being recorded.

The following is an `API_SINGLE` example.

```
typedef  struct{
int iInput;
int iInput2;
int dwResult;
}MYSTRUCT;

LRESULT pMyProxyFunc( int iInput, int iInput2, LPSTR szInString, LPSTR
szOutString)
{
   DWORD dwTicket, dwTimeStamp;
   LRESULT  dwResult;
   MYSTRUCT *pMyStruct;


   dwTicket = ProxyGetTicket();
   if  (dwTicket  == PROXY INVALID_TICKET)
   {
      return  RealFunc(  iInput,  iInput2,  szInString, szOutString)
   }
   else
   {
      dwTimeStamp = ProxyGetTimeStamp();

      dwResult =  RealFunc(  iInput,  iInput2,  szInString,
szOutString)

      pMyStruct = ProxyLockNew(dwTicket,
         dwTimeTimeStamp,
         PROXY_LIB_MY_PROXY,
         P_MYPROXY_ID,
         API_SINGLE,
         sizeof(MYSTRUCT);

      pMyStruct.iInput = iInput;
      pMyStructiInput2 = iInput2;
      pMyStruct.dwResult = dwResult;

      ProxyWriteBlock(szInString, strlen(szInString) + 1);
      ProxyWriteBlock(szOutString, strlen(szOutString) + 1);

      ProxyUnlock(0);

      return dwResult;
   }
}
```

```
typedef  struct{
int iInput;
int iInput2;
}MYSTRUCT_IN;

typedef  struct{
int dwResult;
}MYSTRUCT_OUT;
```

The following is an `API_ENTRY/API_EXIT` example.

```
LRESULT pMyProxyFunc( int iInput, int iInput2, LPSTR szInString, LPSTR
szOutString)
{
   DWORD dwTicket, dwTimeStamp;
   LRESULT  dwResult;
   MYSTRUCT_IN *pMyStructIn;
   MYSTRUCT_OUT *pMyStructOut;


   dwTicket = ProxyGetTicket();
   if  (dwTicket  == PROXY INVALID_TICKET)
   {
      return  RealFunc(  iInput,  iInput2,  szInString, szOutString)
   }
   else
   {
      dwTimeStamp = ProxyGetTimeStamp();

      pMyStructIn = ProxyLockNew(dwTicket,
         dwTime,
         PROXY_LIB_MY_PROXY,
         P_MYPROXY_ID,
         API_ENTRY,
         sizeof(MYSTRUCT_IN);

      ProxyWriteBlock(szInString, strlen(szInString) + 1);

      ProxyUnlock(0);

      dwResult =  RealFunc(  iInput,  iInput2,  szInString,
szOutString)

      pMyStruct = ProxyLockNew(dwTicket,
         dwTime,
         PROXY_LIB_MY_PROXY,
         P_MYPROXY_ID,
         API_EXIT,
         sizeof(MYSTRUCT);

      pMyStruct.dwResult = dwResult;

      ProxyWriteBlock(szOutString, strlen(szOutString) + 1);
```

```
ProxyUnlock(0);
return dwResult;
```

The GetAnnotations() Service Function

Script Generation Options on page 83 describes features that API Recorder Adapters must support and that Custom Recorder Adapters can support. These features allow Robot users to insert comments, synchronization points, start/end blocks, and start/end timers at specified locations inside generated test scripts. If a user specifies these options, they are automatically recorded in an annotations file. Your script generator adapter is then responsible for reading this file and placing the requested code in the test script. You use the `GetAnnotations()` call to access the entries in the annotations file.

GetAnnotations()

Reads the annotations file.

Syntax

```
int GetAnnotations(TCHAR sessionName[], int insertType, TCHAR
   *insertVal, int *insertValSize)
```

Element	Description
sessionName	Enter the name of the session file that the annotations file you are reading is associated with. For custom recordings, the session file name is an input argument included with *StartRecording()* on page 45. For API recordings, the session file name is returned to the API Script Generator Adapter by *SetOptions()* on page 40.
insertType	Enter one of the following values indicating the type of insert to be retrieved: ▪ `RSR_ANNOTATION_SPLITS` ▪ `RSR_ANNOTATION_COMMENTS` ▪ `RSR_ANNOTATION_TIMERS` ▪ `RSR_ANNOTATION_BLOCKS` ▪ `RSR_ANNOTATION_SYNC_PTS` If multiple inserts of this type are present, all are returned.
insertVal	OUTPUT. After a successful call, *insertVal* contains all insertions of the specified type in the annotations file.

Element	Description
insertValSize	Pointer to the size of *insertVal*. If this size is too small, the required size is returned.

Return Value

- RSR_SUCCESS. Success. The call found annotations of type *insertType* and returned them to *insertVal*.
- RSR_OBJECT_DOES_NOT_EXIST. No annotations of type *insertType* were found.
- RSR_BUFFER_TOO_SHORT. The local container *insertVal* is too small to contain all insertions of the specified type.
- RSR_FAILURE. The call failed, probably due to an internal error.

Comments

The first line of the annotations file contains an integer representing the base time at the creation of the session file. All other lines in the file contain time stamps that are offsets relative to the base time. A script generator adapter uses these offsets in order to determine placement of the insertion in the generated test script.

Following is a sample annotations file containing these insertion requests: two split-scripts, one comment, two synchronization points, two start/stop blocks, and two start/stop timers.

```
995471049
79334, script_1
86794, script_2
71792, comment 1
35000, sync point 1
61479, sync point 2
29712, 50763, block 1
55941, 79334, block 2
24035, 79294, timer 1
41699, 79334, timer 2
```

If called with argument RSR_ANNOTATION_SPLITS, GetAnnotations() returns the first three lines; with argument RSR_ANNOTATIONS_COMMENTS, lines 1 and 4; with argument RSR_ANNOTATIONS_TIMERS, the first and last two lines are returned — and so on.

Example

This example queries the annotations file for insertions of type RSR_ANNOTATION_SPLITS.

```
   //specify size of reply buffer
   int ReplySize = 1024;

   //dynamically allocate the reply buffer
   TCHAR *ReplyBuffer = new TCHAR[ReplySize];

   //call GetAnnotations and react according to its return flag.

switch(GetAnnotations(tchar(SessionName),RSR_ANNOTATION_SPLITS,ReplyBu
ffer,&ReplySize))
   {
   case RSR_SUCCESS:
      //you have your request in Reply buffer
      TRACE("\n");
      TRACE(ReplyBuffer);
      TRACE("\n\n");
      break;
   case RSR_OBJECT_DOES_NOT_EXIST:
      //no split-script insertion requests were present
      break;
   case RSR_BUFFER_TOO_SHORT:
      //your buffer was too short. Call again with needed size.
      delete ReplyBuffer;
      ReplyBuffer = new TCHAR[ReplySize];

switch(GetAnnotations(tchar(SessionName),RSR_ANNOTATION_SPLITS,ReplyBu
ffer,&ReplySize))
      {
      case RSR_SUCCESS:
         //you have what you asked for in ReplyBuffer
         TRACE("\n");
         TRACE(ReplyBuffer);
         TRACE("\n\n");
         break;
      default:
         ASSERT(FALSE);
      }
      break;
   case RSR_FAILURE:
   default:
      ASSERT(FALSE);
```

Adapter Configuration 5

About Adapter Configuration

The extensibility framework provides a way for Robot users to specify configuration options for the adapters you develop (except for API Recorder Adapters, which do not support configuration options). The same calls are used for all adapter types:

- `GetOptions()` returns to Robot all configuration options supported by an adapter. But while the call is the same, the supported option arguments are different for different types of adapters.
- `SetOptions()` passes a choice or parameter specified by a Robot user to an adapter, so that the adapter can implement the choice.
- `GetAnnotations()` is a service call that script generator adapters use in order to support scripting options specified by the user during recording.

Configuration Argument Format

Configuration options are passed by defined argument strings specified with `GetOptions()` and `SetOptions()`. Configuration options fall into two categories: those defined by Robot and those defined by an adapter. Robot-defined option arguments have the format

`option-argument`[,*`value`*]

where `option-argument` is a defined string and *`value`* is a setting (settings are relevant for some but not all options). A *`value`* setting is a string that cannot contain commas or semicolons (spaces are okay).

Adapter-defined options have the format

`name,value,description`

Neither of the three arguments can contain internal commas or semicolons.

The two types of arguments, which can be intermixed, are separated by semicolons. For example, the following argument returned in response to `GetOptions()` indicates that the adapter supports two Robot-defined options and one adapter-defined option:

```
RECORD_SPLIT;
TEST_SCRIPT_TYPE, RSR_SCRIPT_TYPE_JAVA;
SERVER_USERNAME,system,Username for privileged operations
```

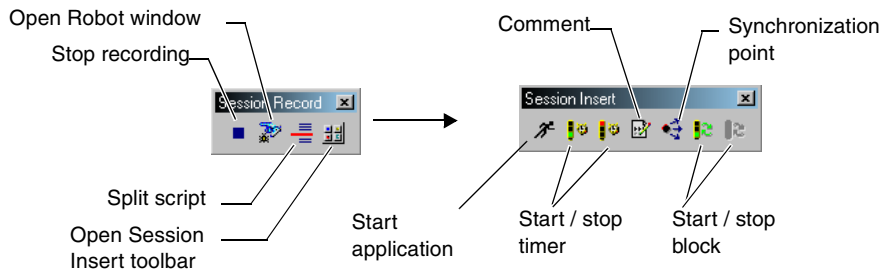
Robot-Defined Configuration Options

There are three categories of Robot-defined options:

- *Recording Options* on page 82.
- *Script Generation Options* on page 83.
- *Miscellaneous (non-GUI) Options* on page 84.

Recording Options

When the user starts recording, the Session Record floating tool bar appears (see below, left side), from which the Session Insert tool bar (right side) can be opened.



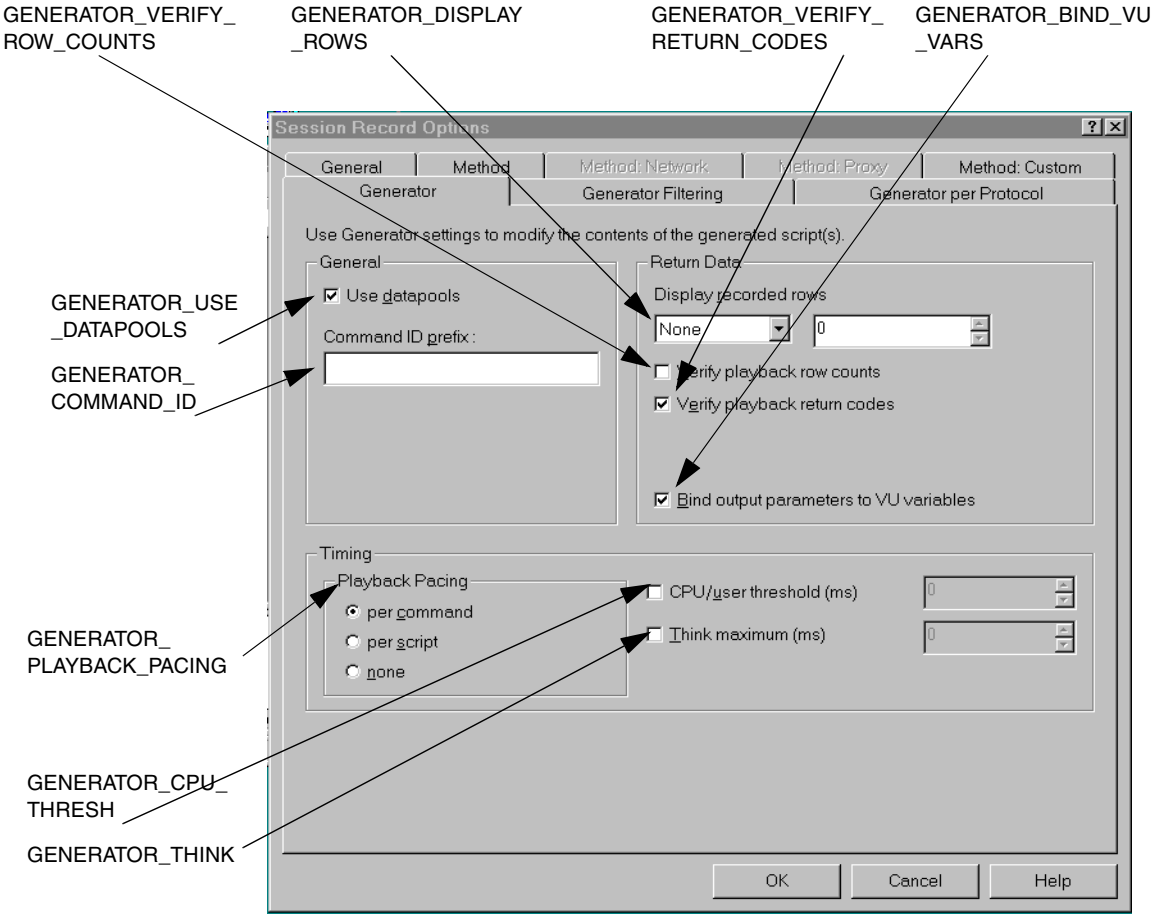
Options on these tool bars allow users to insert time-stamped data into an *annotations* file. If your Custom Recorder Adapter enables these options, the associated Custom Script Generator Adapter, in order to script the preferences, must interpret the annotations file entries and synchronize them with the session file. API Script Generator Adapters must support these options.

The following table lists the argument strings that enable the features on the Session Record and Session Insert dialog boxes. If your Custom Recorder Adapter does not include these argument strings in response to `GetOptions()`, the corresponding icon is dimmed. The icons are always enabled during API recording and so should be supported by API Script Generator Adapters.

Feature	Argument String
Start Application	`RECORD_START_APP`
Split Script	`RECORD_SPLIT`
Insert Timer start/stop	`RECORD_TIMER`
Insert Comment	`RECORD_COMMENT`
Insert Block start/stop	`RECORD_BLOCK`

Script Generation Options

The Generator tab of the Session Record Options dialog box displays script generation options that Custom Script Generator Adapters and API Script Generator Adapters can enable or disable. The following figure shows the GUI options and the argument strings you use to enable the options. If your script generator adapter does not include these argument strings in response to `GetOptions()`, the corresponding checkbox or entry box is shaded. See *GetOptions()* on page 60 (Custom Script Generator Adapters) or *GetOptions()* on page 38 (API Script Generator Adapters) for the exact configuration arguments to use in order to enable these options.



One other script generation option, relevant only for Custom Recorder Adapters, appears on the General tab. If **Prompt for application name on start recording** is checked, the user is prompted to name an application to be started at the outset of recording. The argument string that enables this checkbox is `GENERAL_START_APP_PROMPT`.

Miscellaneous (non-GUI) Options

The following table describes the argument strings for non-GUI configuration options and names the type of adapter that can specify the argument in response to `GetOptions()`. These option arguments describe how your adapters work internally or how adapters communicate with one another or the Robot user.

Argument String	Used by	Description
`CONFIGURATION, name,value, description [,value1, value2 ...]`	Custom Recorder Adapter, Custom Script Generator Adapter, Generator Filter Adapter, API Script Generator Adapter	Specifies an adapter-defined option. See *Adapter-Defined Configuration Options* on page 86.
`CONFIGURATION, USE_CUSTOM_UI`	Custom Recorder Adapter, Custom Script Generator Adapter, Generator Filter Adapter, API Script Generator Adapter	Specifies that the adapter supplies a custom GUI for displaying and editing of adapter-defined configuration settings. See *Using a Custom UI for Custom Options* on page 87.
`DEFAULT_SCRIPT _GENERATOR`	Custom Recorder Adapter	Specifies the display name of the corresponding Custom Script Generator Adapter. If not specified, the user must select the adapter from the Script Generator box on the Method tab. If a Custom Recorder Adapter can be used with only one Custom Script Generator Adapter, you can remove ambiguity for the user by assigning them the same display name. For example, Rational's recorder and script generator adapters for the BEA WebLogic environment have the same display name, **rtweblogicejb**.
`SESSION_FILES, type`	Custom Recorder Adapter	Specifies the internal format (Rational binary format, XML, or custom) of the session file(s). This option must be specified.
`TEST_SCRIPT_TYPE, type`	Custom Script Generator Adapter, Generator Filter Adapter	Specifies the language (Java, Visual Basic, or VU) of the generated script. This option must be specified.
`USE_SCRIPTGEN _PROGRESS`	Custom Script Generator Adapter	Specifies that an adapter supplies progress information so that the waiting user can see the status in a progress dialog box.

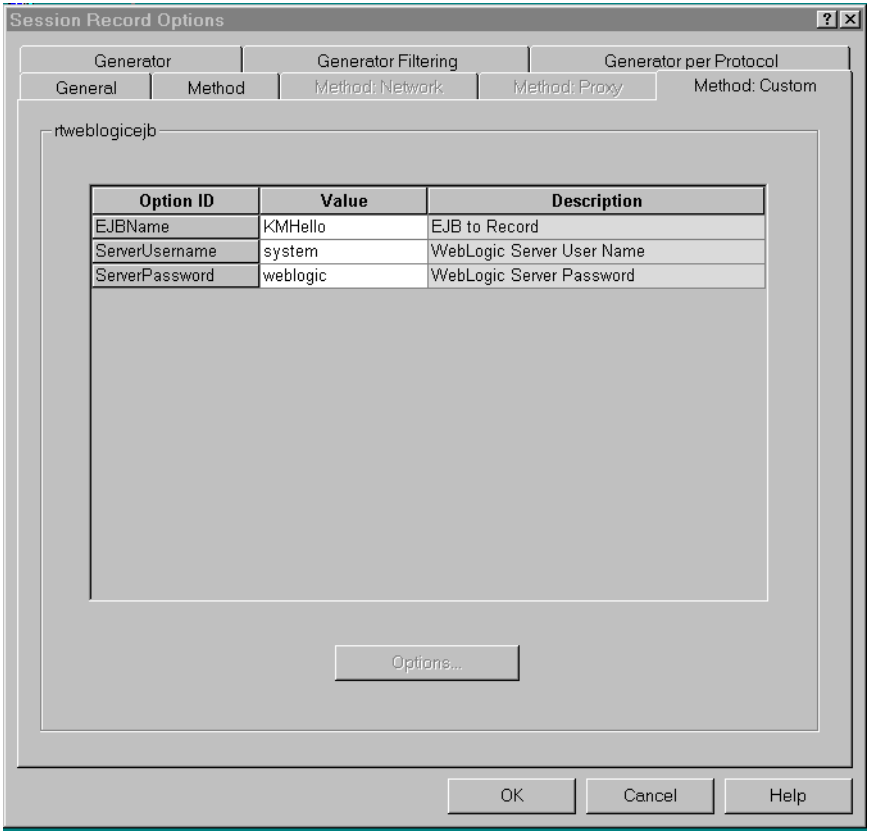
Adapter-Defined Configuration Options

Adapters may define custom configuration options (except for API Recorder Adapters, which do not support custom options). Adapter-defined options are comma-separated name-value-description triplets that are known only to the adapter. These options are defined with this argument string:

`CONFIGURATION,`*`name`*`,`*`value`*`,`*`description`*`[,`*`value1`*`,`*`value2`*` ...]`

Robot users may specify adapter-defined configuration options. If you supply optional value arguments to define all valid settings for an option, instead of having to enter a value, the user selects from a predefined list of values. This prevents typographical errors and eliminates the need to parse user selections.

By default, custom options returned from a Custom Recorder Adapter in response to the `GetOptions()` call appear on the Method: Custom tab (illustrated below).



The following fragment shows the response to `GetOptions()` by a Custom Recorder Adapter that would result in the display shown above.

```
CONFIGURATION,EJBName,KMHello,EJB to Record;
CONFIGURATION,ServerUsername,system,WebLogic Server User Name;
CONFIGURATION,ServerPassword,Weblogic,WebLogic Server Password
```

The triplet grid shown above appears on a different tab depending on the type of adapter:

- Custom Recorder Adapter options appear on the Method:Custom tab shown above.
- Custom Script Generator Adapter options appear on the Generator:Custom tab.
- API Script Generator Adapter options appear on the Generator per Protocol tab.

Using a Custom UI for Custom Options

Your adapter can provide a custom GUI for options it defines. This choice is controlled by the option:

```
CONFIGURATION,USE_CUSTOM_UI
```

If your adapter specifies this option, the **Configure** button at the bottom of the options grid (see the previous figure) is enabled. If the user clicks this button, you start the GUI.

If your adapter provides a custom GUI, your response to `GetOptions()` is not required to include custom options or to adhere to the Robot-defined triplet format for custom options. If your `GetOptions()` response does include custom options, the Robot user can use both the triplet grid and the custom GUI.

If your adapter provides a custom GUI, you are responsible for any necessary user documentation and for conveying a user's choices to the adapter.

Index

R

S

T

W