

Customizing Rational Rose RealTime for Target Control and Observability

RATIONAL ROSE® REALTIME

VERSION: 2002.05.20

WINDOWS/UNIX

IMPORTANT NOTICE

COPYRIGHT

Copyright ©2002, Rational Software Corporation. All rights reserved.

Part Number: 800-025105-000

Version Number: 2002.05.20

PERMITTED USAGE

THIS DOCUMENT CONTAINS PROPRIETARY INFORMATION WHICH IS THE PROPERTY OF RATIONAL SOFTWARE CORPORATION (“RATIONAL”) AND IS FURNISHED FOR THE SOLE PURPOSE OF THE OPERATION AND THE MAINTENANCE OF PRODUCTS OF RATIONAL. NO PART OF THIS PUBLICATION IS TO BE USED FOR ANY OTHER PURPOSE, AND IS NOT TO BE REPRODUCED, COPIED, ADAPTED, DISCLOSED, DISTRIBUTED, TRANSMITTED, STORED IN A RETRIEVAL SYSTEM OR TRANSLATED INTO ANY HUMAN OR COMPUTER LANGUAGE, IN ANY FORM, BY ANY MEANS, IN WHOLE OR IN PART, WITHOUT THE PRIOR EXPRESS WRITTEN CONSENT OF RATIONAL.

TRADEMARKS

Rational, Rational Software Corporation, Rational the e-development company, ClearCase, ClearCase Attache, ClearCase MultiSite, ClearDDTS, ClearQuest, ClearQuest MultiSite, DDTS, Object Testing, Object-Oriented Recording, ObjecTime & Design, Objectory, PerformanceStudio, ProjectConsole, PureCoverage, PureDDTS, PureLink, Purify, Purify'd, Quantify, Rational, Rational Apex, Rational CRC, Rational Rose, Rational Suite, Rational Summit, Rational Visual Test, Requisite, RequisitePro, RUP, SiteCheck, SoDA, TestFactory, TestFoundation, TestMate, The Rational Watch, AnalystStudio, ClearGuide, ClearTrack, Connexis, e-Development Accelerators, ObjecTime, Rational Dashboard, Rational PerformanceArchitect, Rational Process Workbench, Rational Suite AnalystStudio, Rational Suite ContentStudio, Rational Suite Enterprise, Rational Suite ManagerStudio, Rational Unified Process, SiteLoad, TestStudio, VADS, among others, are either trademarks or registered trademarks of Rational Software Corporation in the United States and/or in other countries. All other names are used for identification purposes only, and are trademarks or registered trademarks of their respective companies.

Microsoft, the Microsoft logo, Active Accessibility, Active Channel, Active Client, Active Desktop, Active Directory, ActiveMovie, Active Platform, ActiveStore, ActiveSync, ActiveX, Ask Maxwell, Authenticode, AutoSum, BackOffice, the BackOffice logo, BizTalk, Bookshelf, Chromeffects, Clearlead, ClearType, CodeView, Computing Central, DataTips, Developer Studio, Direct3D, DirectAnimation, DirectDraw, DirectInput, DirectMusic, DirectPlay, DirectShow, DirectSound, DirectX, DirectXJ, DoubleSpace, DriveSpace, FoxPro, FrontPage, Funstone, IntelliEye, the

IntelliEye logo, IntelliMirror, IntelliSense, J/Direct, JScript, LineShare, Liquid Motion, the Microsoft eMbedded Visual Tools logo, the Microsoft Internet Explorer logo, the Microsoft Office Compatible logo, Microsoft Press, the Microsoft Press logo, Microsoft QuickBasic, MS-DOS, MSDN, Natural, NetMeeting, NetShow, the Office logo, One Thumb, OpenType, Outlook, PhotoDraw, PivotChart, PivotTable, PowerPoint, QuickAssembler, QuickShelf, Realmation, RelayOne, Rushmore, SourceSafe, TipWizard, TrueImage, TutorAssist, V-Chat, VideoFlash, Virtual Basic, the Virtual Basic logo, Visual C++, Visual FoxPro, Visual InterDev, Visual J++, Visual SourceSafe, Visual Studio, the Visual Studio logo, Vizact, WebBot, WebPIP, Win32, Win32s, Win64, Windows, the Windows CE logo, the Windows logo, Windows NT, the Windows Start logo, and XENIX are trademarks or registered trademarks of Microsoft Corporation in the United States and other countries.

FLEXIm and GLOBEtrotter are trademarks or registered trademarks of GLOBEtrotter Software, Inc. Licensee shall not incorporate any GLOBEtrotter software (FLEXIm libraries and utilities) into any product or application the primary purpose of which is software license management.

Portions Copyright ©1992-2002, Summit Software Company. All rights reserved.

PATENT

U.S. Patent Nos. 5,193,180 and 5,335,344 and 5,535,329 and 5,835,701. Additional patents pending.

Purify is licensed under Sun Microsystems, Inc., U.S. Patent No. 5,404,499.

GOVERNMENT RIGHTS LEGEND

Use, duplication, or disclosure by the U.S. Government is subject to restrictions set forth in the applicable Rational Software Corporation license agreement and as provided in DFARS 277.7202-1(a) and 277.7202-3(a) (1995), DFARS 252.227-7013(c)(1)(ii) (Oct. 1988), FAR 12.212(a) (1995), FAR 52.227-19, or FAR 227-14, as applicable.

WARRANTY DISCLAIMER

This document and its associated software may be used as stated in the underlying license agreement. Rational Software Corporation expressly disclaims all other warranties, express or implied, with respect to the media and software product and its documentation, including without limitation, the warranties of merchantability or fitness for a particular purpose or arising from a course of dealing, usage, or trade practice.

Contents

1 Customizing for Target Control and Observability	7
Introduction	7
Model Compilation and Target Control	7
Intended Audience	8
Target Control	8
Target Control Modes	9
Manual Mode	9
Basic Mode	9
Debugger Mode	10
Target Control Scripts	10
Menu Commands	11
Reset	11
Load	12
Unload	13
Execute	14
Terminate	15
General Issues	16
Third-Party Source Code Debugger Integration	16
Registering Threads on Unix	17
Calling Sequence	17
Debugger DLL API	19
Get DLL Capabilities	19
Create Debug Session	20
Destroy Debug Session	21
Initialize Debugger	22
Cleanup Debugger	22
Start Debugger	23
Stop Debugger	24
Set Callback	24
Event Callback Function	25
Set Source Search Path	26

Set Breakpoint in File27
Set Breakpoint At Function27
Clear Breakpoint28
Set DllTrace29

Index	31
--------------------	-----------

Customizing for Target Control and Observability

1

Contents

This chapter is organized as follows:

- *Introduction* on page 7
- *Model Compilation and Target Control* on page 7
- *Target Control* on page 8
- *Menu Commands* on page 11
- *Third-Party Source Code Debugger Integration* on page 16

Introduction

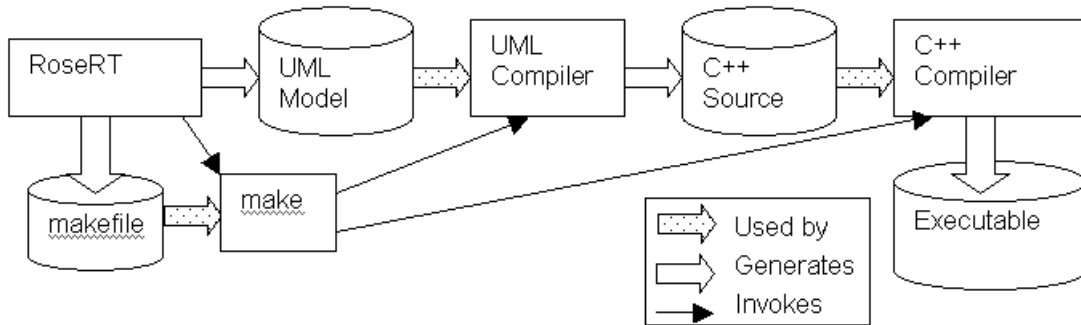
Rational Rose RealTime is a comprehensive visual modeling environment that delivers a powerful combination of notation, processes, and tools optimized to meet the challenges of real-time software development. The Rational Rose RealTime UML model compiler converts models directly into executable applications. Those executables can be controlled and debugged at run-time under the control of the toolset. Rational Rose RealTime integrates with source debuggers providing the developer with the choice of debugging at the UML and source code level. A combination of UML editors, a model compiler, and run-time debugging tools address the complete life-cycle of a project from early use case analysis through design, implementation, and test.

This document describes how to add support to Rational Rose RealTime 6.0 and later for target control and observability, and how to integrate Rational Rose RealTime with source code debuggers.

Model Compilation and Target Control

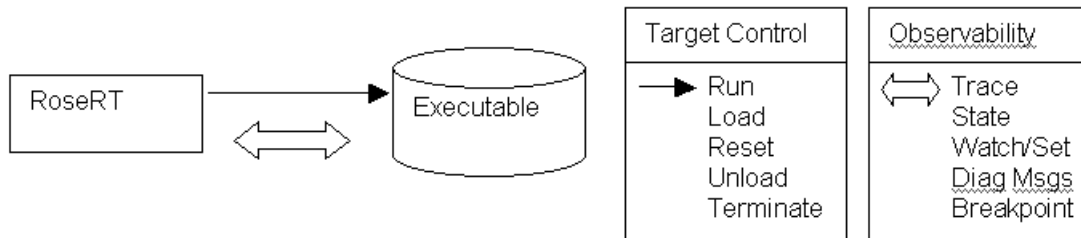
Rational Rose RealTime models are compiled seamlessly into applications ready for execution on the host or target operating systems. Figure 1 provides a high level overview of model compilation.

Figure 1 UML Model Compilation



Rational Rose RealTime also has the ability to control the executing application at run-time (for example, during debugging). Target Observability provides the ability to observe and debug the executing application at the UML level. Figure 2 shows a simplified high-level overview of Target Control and Observability.

Figure 2 Target Control and Observability



Rational Rose RealTime also supports inter-working with traditional source code debuggers. This enables developers to control, observe, and debug the application at the UML level and detailed source code level simultaneously.

Intended Audience

This guide is specifically designed for technical staff responsible for enabling these capabilities for a specific target execution environment. It is assumed that the reader has significant knowledge and experience with the development environment, operating system, and target hardware.

Target Control

Target Control refers to the Rational Rose RealTime toolset's features that load, unload, execute, and terminate a Rational Rose RealTime-generated application, as well as the ability to reset a remote target platform.

Target Control is not the same feature as Target Observability. Target Observability allows the observation of the application executing on a target from the UML level (such as state change, state machine breakpoints, event tracing, and so on) on the host-based toolset. Target Control interacts with the APIs of the target execution environment to load, run, and terminate the application, whereas Target Observability communicates directly with the running application.

Target Control Modes

Rational Rose RealTime supports three different Target Control modes:

- Manual Mode
- Basic Mode
- Debugger Mode

Manual Mode

In **Manual mode**, Rational Rose RealTime does not provide any Target Control functionality. The user is responsible for performing Target Control operations (such as loading, executing). After the target application starts, the user can direct the Rational Rose RealTime toolset to connect to the executing target application for Target Observability.

Basic Mode

In **Basic mode**, Rational Rose RealTime uses the target environment's APIs to control the execution of the target application. Rational Rose RealTime supports automatic target control for a number of host and target platform combinations. Users deploy on a number of other target environments as well.

Rational Rose RealTime uses Perl scripts to perform the Target Control operations. These scripts can call the target APIs directly or can call some intermediary helper application to control the execution on the target.

There are five Target Control scripts:

- reset.pl
- load.pl
- unload.pl
- execute.pl
- terminate.pl

Debugger Mode

Debugger mode provides same the capabilities as Basic mode and, in addition, provides the ability to inter-work with a C or C++ source debugger (for example, Visual C++) to set source code level breakpoints from within the UML model. When these source breakpoints are hit at run-time, control of the executable is passed to the source debugger. When the application is continued, control of the executable is passed back to the Rational Rose RealTime toolset. Debugger mode provides an integrated debug environment that permits a simultaneous use of source code and UML debugging styles.

Target Control Scripts

When you open the Specification dialog for a **Processor** in the **Deployment View**, notice that **Load Scripts** text box specifies the path to the Target Control scripts (for example, `$TARGET_PATH/win32/`, `$TARGET_PATH/tornado2/`). This directory contains up to five Target Control scripts, each of which has a different function:

- **reset.pl** - resets the target processor, see Reset
- **load.pl** - loads a Component onto a target, see Load
- **unload.pl** - unloads a Component from a target, see Unload
- **execute.pl** - executes a Component, see Execute
- **terminate.pl** - terminates the execution of a Component, see Terminate

The Target Control Scripts determine the Target Control capabilities for the Processor. If a script exists in the Target Control Scripts directory, then the toolset assumes that the corresponding capability exists. Whenever a Component Instance is created on a Processor (that is, a Component in the **Component View** is assigned to a Processor in the **Deployment View**), the toolset checks to see which scripts are available and enables those capabilities in the toolset menus that are accessible by right-clicking on a Component Instance. These menu options are now available to the user.

The presence of the scripts is not their only purpose. Each existing Target Control script must also provide the associated capability. For example, the load script must load the corresponding component onto the target specified by the Processor, and so on. The scripts use information from Processor and Component Instances specifications, but note that the scripts do not need to use all the parameters that are passed to them. Any script just needs to process the arguments that allow it to perform its intended operation.

These scripts are written in Perl, but they may spawn other executables that may be needed to provide the desired capability. Every script also indicates whether it was successful.

Menu Commands

If the path to the Target control scripts contains the following scripts, that corresponding menu command will become active on the **Processor** menu:

- **reset.pl** - resets the target processor and activates the **Reset** menu option
- **load.pl** - loads a Component onto a target and activates the **Load** menu option
- **unload.pl** - unloads a Component from a target and activates the **Unload** menu option
- **execute.pl** - executes a Component and activates the Run menu option (**Execute**)
- **terminate.pl** - terminates the execution of a Component and activates the **Shutdown** menu option (**Terminate**)

Reset

Description

The `reset.pl` script resets a target processor. If this script exists, the **Reset** menu item will be active on the corresponding Processor menu.

Command Line

```
Rtperl reset.pl -ip target -server targetServer -os targetOS -cpu targetCPU
```

Arguments

<code>-ip <i>target</i></code>	Target name or address.
<code>-server <i>targetServer</i></code>	Target server name or address.
<code>-os <i>OS</i></code>	OS executing on target.
<code>-cpu <i>CPU</i></code>	CPU on the target.

Returns

<code>::Ok::</code>	String indicating success.
Error String	Error string to be displayed in error message box in the toolset.

Note: The data for the script arguments are retrieved from the **Processor Specification** dialog.

Load

Description

The load.pl script loads a component onto the corresponding target processor. If this script exists, the **Load** menu item will be active on the corresponding Component Instance menu when the Component Instance is in a "loadable" state.

Command Line

```
Rtperl load.pl -ip target -server targetServer -os targetOS -cpu targetCPU  
-exe componentDir -prio priority -port Toport
```

Arguments

-ip <i>target</i>	Target name or address.
-server <i>targetServer</i>	Target server name or address.
-os <i>OS</i>	OS executing on target.
-cpu <i>CPU</i>	CPU on the target.
-exe <i>executable</i>	6.1 and later: Fully qualified executable name.
-prio <i>priority</i>	Priority to run the component instance
-port <i>Toport</i>	Target Observability port.

Returns

::Ok:: [-warning 'xxx'] [-passback xxx]	6.1 and later: String indicating success. Now two option parameters may follow the ::Ok:: string : -warning and -passback . See General Issues.
Error String	Error string to be displayed in error message box in the toolset.

Note: The data for the options are retrieved from the **Processor** and **Component Instance Specification**.

Unload

Description

The unload.pl script removes a component from the corresponding target processor. If this script exists, the **Unload** menu item will be active on the corresponding Component Instance menu when the Component Instance is in an "unloadable" state.

Command Line

```
Rtperl unload.pl -ip target -server targetServer -os targetOS -cpu targetCPU  
-exe componentDir -prio priority -port TOport ParamsFromLoad
```

Arguments

-ip target	Target name or address.
-server targetServer	Target server name or address.
-os OS	OS executing on target.
-cpu CPU	CPU on the target.
-exe executable	6.1 and later: Fully qualified executable name.
-prio priority	Priority to run the component instance
-port Toport	Target Observability port.
ParamsFromLoad	Any parameters that were returned from a successful Load operation.

Returns

::Ok:: [-warning 'xxx']	6.1 and later: String indicating success. Now, one option parameter may follow ::Ok:: string: -warning . See General Issues.
Error String	Error string to be displayed in error message box in the toolset.

Note: The data for the options are retrieved from the **Processor** and **Component Instance Specification**.

Execute

Description

The `execute.pl` script starts execution of a component instance on the corresponding target processor. If this script exists, the **Run** menu item is available on the Component Instance menu when the Component Instance is in a "runable" state.

Command Line

```
Rtperl execute.pl -ip target -server targetServer -os targetOS -cpu targetCPU  
-exe componentDir -prio priority -port Toport  
-args commandLineArgs
```

Arguments

-ip <i>target</i>	Target name or address.
-server <i>targetServer</i>	Target server name or address.
-os <i>OS</i>	OS executing on target.
-cpu <i>CPU</i>	CPU on the target.
-exe <i>componentDir</i>	6.0.x: Path to Component directory. It is used to locate the component.
-exe <i>executable</i>	6.1 and later: Fully qualified executable name.
-prio <i>priority</i>	Priority to run the component instance
-port <i>Toport</i>	Target Observability port.
-args <i>commandLineArgs</i>	Command Line arguments that are to be used when starting the target application. Parameters that follow the -args tag are all passed to the target application.

Returns

::Ok:: paramsFromExecute	String indicating success. Any strings passed back after the ::Ok:: will be based to the terminate.pl script when the user invokes the <i>Shutdown</i> command.
::Ok:: [-warning 'xxx'] [-passback xxx]	6.1 and later: String that represents the operation was successful. Now two option parameters may follow the ::Ok:: string: -warning and -passback . See General Issues.
Error String	Error string to be displayed in error message box in the toolset.

Note: The data for the options are retrieved from the Processor and Component Instance Specification.

An example of **paramsFromExecute** is a handle that identifies the process that was created. For example, on Windows we return **-pid nnnnnn**. This allows us to pass back the PID (Process ID) to the Terminate script.

Terminate

Description

The terminate.pl script is used to kill a component instance on the corresponding target processor. If this script exists, the **Shutdown** menu item will be active on the corresponding Component Instance menu when the Component Instance is in a "killable" state.

Command Line

```
Rtperl execute.pl -ip target -server targetServer -os targetOS -cpu targetCPU  
-exe componentDir -prio priority -port TOport  
paramsFromExecute
```

Arguments

-ip target	Target name or address.
-server targetServer	Target server name or address.
-os OS	OS executing on target.
-cpu CPU	CPU on the target.

-exe executable	6.1 and later: Fully qualified executable name.
-prio priority	Priority to run the component instance.
-port Toport	Target Observability port.
ParamsFromExecute	Any parameters that were returned from a successful Run operation.

Returns

::Ok:: [-warning 'xxx']	6.1 and later: String indicating success. Now optional parameter may follow the ::Ok:: string: -warning . See General Issues.
Error String	Error string to be displayed in error message box in the toolset.

Note: The data for the options are retrieved from the **Processor** and **Component Instance Specification**.

General Issues

- The **-exe** option is followed by the Component Directory in releases **6.0.x**. The **Load** and **Execute** scripts call a Perl script (**findexe.pl**) to find the corresponding executable.
- The **-exe** option is followed by the fully qualified executable name in releases **6.1** and later.
- **6.1** formalized what comes after the **::Ok::** string. The **Load**, **Unload**, **Execute**, and **Terminate** can succeed (in other words, return **::Ok::**) but may return a warning. The warning is identified by the parameter **-warning** followed by a string enclosed in single quotes ('). The toolset will display a dialog box specifying that a warning occurred. The string returned in quotes is appended to the toolset logs. Anything appearing after the **-passback** parameter will be returned to the originating call.

Third-Party Source Code Debugger Integration

The format for the Debugger Mode is **Debugger-X** where **X** is the name of the debugger DLL. This DLL must exist in the **\$ROSERT_HOME/bin/\$ROSERT_HOST** directory and is called **libX.dll**.

Registering Threads on Unix

When building a debugger integration DLL without MainWin and using **callback** functions, additional steps are required to ensure that Rational Rose RealTime knows about the **callback** thread. The following steps are necessary for a thread-safe interface:

- Call **tcThreadInit()** from the **callback** thread before doing any callbacks.
- The **callback** thread must call **tcThreadCleanup()** before terminating.

There is a header file for this service in `$ROSSERT_HOME/bin/tc/tcsetup.h` and a supporting dynamic library (for Solaris) in `$ROSSERT_HOME/bin/tc/sun5/libtcsetup.so`.

You may call **tcThreadInit** (`init`) and **tcThreadCleanup** (`cleanup`) as many times as you like, as long as the **tcThreadInit** is always followed by a **tcThreadCleanup** before the next **init** occurs. This is useful if you wanted to do something similar to the following: **tcThreadInit**, **callback**, **tcThreadCleanup**, for each **callback** instead of **tcThreadInit** at thread startup, and **tcThreadCleanup** at thread termination. However, we recommend that the **tcThreadInit** and **tcThreadCleanup** functions be called only once (**tcThreadInit** at startup and **cleanup** at termination) since this approach is less error prone.

Calling Sequence

Source code debuggers come with a variety of capabilities. For the toolset to use the debugger DLL in the best possible way, the DLL must provide a list of its capabilities. The following are capabilities of the debugger DLL that are available to Rational Rose RealTime:

Capability	Description
Function Breakpoints	The DLL uses the function name to set a breakpoint.
Line Breakpoints	The DLL uses a file name and line number to set a breakpoint.
Detects Breakpoint Hits	The DLL calls the callback function when a breakpoint is hit.
User Termination Detected	The DLL calls the callback function when it detects that the user terminated the debugger manually.
Debugger Loads Target	The DLL must be called to load the target. If not, the toolset uses the Basic mode mechanism, if one exists.
Debugger Unloads Target	The DLL must be called to unload the target. If not, the toolset uses the Basic mode mechanism, if one exists.
Debugger Executes Component	The DLL must be called to start the Component Instance. If not, the toolset uses the Basic mode mechanism, if one exists.

Capability	Description
Debugger Terminates Component Instance	The DLL must be called to terminate a component instance. If not, the toolset will use the Basic mode mechanism if one exists.
Supports Search Paths	The DLL can use a given search path to search for source code.
Reload Before Restarting	The target must be reloaded before it is restarted.

The values of these flags determine how and which debugger DLL functions are called. The following are the rules of operation.

- 1 The debugger DLL is loaded once the user applies the change to the Operation Mode in the Component Instance specification for the Component Instance. The debugger DLL is loaded only once per toolset session.
- 2 If the DLL is loaded successfully, the toolset obtains the debugger DLLs capabilities and saves them.
- 3 Next, the toolset calls the **tcCreateDebugSession** function to create a new session.
Note: A new session is created for each Component Instance that uses the debugger DLL.
- 4 The Target Control capabilities (**Load, Unload, Run, Shutdown**) are determined using the debugger DLL capabilities as well as the Target Control scripts. The debugger DLL capabilities take precedence over the Target Control scripts.
- 5 If a target must be loaded, it can be loaded in one of two ways: using the debugger or the Basic mode Target Control script. If the "Debugger Loads Target" flag is set, the debugger DLL is expected to load the target in the **tcInitializeDebugger** function. Otherwise, the Target Control load script is used to load the target, and then the **tcInitializeDebugger** function is called.
- 6 If the target is not loadable, then the **tcInitializeDebugger** function is called when the user invokes the Run command.
- 7 If the "Debugger Executes Component" flag is set, then the **tcStartDebugger** function is called. If not set, then the Target Control execute script is called and then followed by a call to the **tcStartDebugger** function. **Note:** The breakpoint functions may be called before the **tcStartDebugger** function if breakpoints were set in the previous debug session.

- 8 When the user invokes the Shutdown command, all breakpoints are removed, and the **tcStopDebugger** function is called. If the "Debugger Terminates Component Instance" is set, the **tcStopDebugger** must terminate the Component Instance. If not set, then the Target Control terminate script is called. If the target does not need to be unloaded, then the **tcCleanupDebugger** function is called as well.
- 9 When the user invokes the Unload command and the "Debugger Unloads Component" flag is set, then the **tcCleanupDebugger** function is called. This function must unload the component from the target. If this capability is not set, then the Target Control unload script is called.
- 10 When the Debugger DLL is unloaded from the toolset (when the Component Instance Operation mode is changed or when the toolset is shut down) **tcDestroySession** is called. This function is responsible for releasing any resources associated with this debugger DLL session.

Debugger DLL API

This section describes the API that must be implemented by a debugger DLL. The file, `tcDllinterface.h` contains all the required type declarations and function prototypes. The functions are:

- Get DLL Capabilities
- Create Debug Session
- Destroy Debug Session
- Initialize Debugger
- Cleanup Debugger
- Start Debugger
- Stop Debugger
- Set Callback
- Event Callback Function
- Set Source Search Path
- Set Breakpoint in File
- Set Breakpoint At Function
- Clear Breakpoint
- Set DllTrace

Note: Several functions have parameters of type **TC_TCHAR**. This type corresponds to **TCHAR** type familiar to Windows developers. It is either a regular character (**char**) or a wide character (**wchar_t**). By default, **TC_TCHAR** is type defined to **char** in the file `tcDllinterface.h`.

Get DLL Capabilities

TCRET

```

tcGetDllCapabilities(
    TCDLLCAPS * pCaps/* Pointer to struct to get the
    capabilities */
) ;

```

Description

This function populates in the given capability structure with the capabilities of the corresponding DLL. This is the first function that is called in the debugger DLL.

Arguments

TCDLLCAPS * pCaps	Structure to receive the DLL capabilities.
--------------------------	--

Returns

TC_OK	Operation was successful.
TC_FAILED	Operation failed. Missing capability structure.

Create Debug Session

```

TCHANDLE
tcCreateDebugSession(
    const TC_TCHAR * szServerName, /* Name of Target
Server */
    const TC_TCHAR * szTargetName, /* Name of Target*/
    const TC_TCHAR * szArchitecture, /* Processor
Architecture */
    const TC_TCHAR * szOS,          /* Operating System
*/
    TCDEBUGFLAG eFlag              /*
Enables/disables Tracing*/
) ;

```

Description

This function is called to create a debug session. It is called after the debugger DLL is loaded. It returns a DLL specific handle that represents the newly created session. This handle is passed back to all other calls except the **tcGetDllCapabilities**. Typically, the handle is a pointer to a DLL-specific structure that maintains session-specific information.

Arguments

const TC_TCHAR * szServerName	Name or address of a Target Server.
const TC_TCHAR * szTargetName	Name or address of the target.
const TC_TCHAR * szArchitecture	Type of CPU on the target.
const TC_TCHAR * szOS	OS running on the target
TCDEBUGFLAG eFlag	Enables/Disables Debug output from the DLL. See Note below.

Returns

TCHANDLE	DLL specific handle identifying the newly created session.
(TCHANDLE)0	Unable to create a session.

Note: Note: Currently, the toolset does not provide any means to set or clear the debug flag.

Destroy Debug Session

TCRET

```
tcDestroyDebugSession(  
    TCHANDLE    hSession /* Session to terminate */  
);
```

Description

This function is called before the Debugger DLL is unloaded. It must release all session-specific resources that may have been allocated during the session.

Arguments

TCHANDLE hSession	A handle identifying a particular debug session.
--------------------------	--

Returns

TC_OK	Operation was successful.
TC_FAILED	Operation failed.

Initialize Debugger

TCRET

```
tcInitializeDebugger(  
    TCHANDLE          hSession, /* Debugger Session */  
    const TC_TCHAR *  szComponent /* Location/name of the  
component */  
);
```

Description

This function is called to identify the component that the debugger is to work with. In some environments this function will load the component onto the target.

Arguments

TCHANDLE hSession	A handle identifying a particular debug session.
const TC_TCHAR * szComponent	The fully qualified name of the component.

Returns

TC_OK	Operation was successful.
TC_FAILED	Operation failed.

Cleanup Debugger

TCRET

```
tcCleanupDebugger(  
    TCHANDLE          hSession /* Debugger Session */  
);
```

Description

This function is called to undo the activities of the `tcInitializeDebugger` function. In some environments this function will unload the component from the target.

Arguments

TCHANDLE hSession	A handle identifying a particular debug session.
---------------------------------	--

Returns

TC_OK	Operation was successful.
TC_FAILED	Operation failed.

Start Debugger

TCRET

```
tcStartDebugger (  
    TCHANDLE                    hSession, /* Debugger Session */  
    const TC_TCHAR * pszArgs, /* Command line arguments for  
comp */  
    int                        nPriority   /* start up priority */  
);
```

Description

This function is called to start the Component Instance. If the debugger does not start the Component instance, this is the point where the debugger should attach to it.

Arguments

TCHANDLE hSession	A handle identifying a particular debug session.
const TC_TCHAR * pszArgs,	Command line arguments for the Component Instance.
int nPriority	Priority to run the application.

Returns

TC_OK	Operation was successful.
TC_FAILED	Operation failed.

Stop Debugger

TCRET

```
tcStopDebugger(  
    TCHANDLE          hSession /* Loader.Debugger Session */  
) ;
```

Description

This function is called to terminate the Component Instance. If the debugger does not terminate the Component instance, this is the point where the debugger should detach from it.

Arguments

TCHANDLE hSession	A handle identifying a particular debug session.
---------------------------------	--

Returns

TC_OK	Operation was successful.
TC_FAILED	Operation failed.

Set Callback

TCRET

```
tcSetCallback(  
    TCHANDLE          hSession,          /* Debugger Session */  
    CALLBACKFNC      pfncCallback, /* function to call on event */  
    USERDEFINED      lUserDefined1, /* toolset defined data */  
    USERDEFINED      lUserDefined2 /* toolset defined data */  
) ;
```


Description

This function is called during the Target Observability session if the debugger DLL can detect breakpoint hits or user termination. It is used to set or clear a Toolset defined function.

Arguments

TCHANDLE hSession	A handle identifying a particular debug session.
CALLBACKFNC pfncCallback,	Pointer to function the debugger DLL is to call when a breakpoint hit or user termination is detected.
USERDEFINED IUserDefined1	Toolset information that must be passed back in the callback function.
USERDEFINED IUserDefined2	Toolset information that must be passed back in the callback function.

Returns

TC_OK	Operation was successful.
TC_FAILED	Operation failed.

Event Callback Function

```
void  
fncCallback(  
    TCDLLEVENT* pEvent, /* identifies what event  
occurred */  
    USERDEFINED data1, /* data from SetCallback */  
    USERDEFINED data2 /* data from SetCallback */  
);
```

Description

This is the prototype of the callback function that is to be called by the debugger DLL when a breakpoint hit or user termination is detected.

Arguments

TCDLLEVENT * pEvent	Identifies the type of event the Debugger DLL is notifying the toolset of.
USERDEFINED IUserDefined1	Toolset information from the last tcSetCallback.
USERDEFINED IUserDefined2	Toolset information from the last tcSetCallback.

Returns

void	Nothing
-------------	---------

Set Source Search Path

TCRET

```
tcSetSearchPath(  
    TCHANDLE          hSession, /* Debugger Session */  
    int                nEntries, /* number of paths */  
    const TC_TCHAR ** ppszSearchPaths /* list of search paths */  
);
```

Description

This function is called by the toolset to specify the directories that contain the generated source code.

Arguments

TCHANDLE hSession	A handle identifying a particular debug session.
int nEntries	The number of paths specified in the next parameter.
const TC_TCHAR ** ppszSearchPaths	A list of search paths.

Returns

TC_OK	Operation was successful.
--------------	---------------------------

TC_FAILED	Operation failed.
------------------	-------------------

Set Breakpoint in File

```

unsigned long
tcSetBreakpointInFile(
    TCHANDLE          hSession, /* Debugger Session */
    const TC_TCHAR *  szFileName, /* File to set breakpoint in
*/
    int               nLineNo /* line number in file */
) ;

```

Description

This function is called when a breakpoint is required and the Debugger DLL supports breakpoints using file name and line number. This function may be called before the tcStartDebugger.

Arguments

TCHANDLE hSession	A handle identifying a particular debug session.
const TC_TCHAR * szFileName	Name of file where we want to set the breakpoint.
int nLine	No The line number in the file where the breakpoint is to be set.

Returns

unsigned long	A number uniquely identifying the corresponding breakpoint.
0	Unable to set breakpoint.

Set Breakpoint At Function

```

unsigned long
tcSetBreakpointAtFnc(
    TCHANDLE          hSession, /* Debugger Session */
    const TC_TCHAR *  * szFunctionName /* fully qualified name
*/

```

```
) ;
```

Description

This function is called when a breakpoint is required and the Debugger DLL supports breakpoints using function names. This function may be called before the `tcStartDebugger`.

Arguments

TCHANDLE hSession	A handle identifying a particular debug session.
const TC_TCHAR * szFunctionName	The fully qualified name of the function.

Returns

unsigned long	A number uniquely identifying the corresponding breakpoint.
0	Unable to set breakpoint.

Clear Breakpoint

```
TCRET
```

```
tcClearBreakpoint (  
    TCHANDLE            hSession,            /* Debugger Session */  
    unsigned long      nBreakpointId      /* breakpoint to remove  
*/  
);
```

Description

This function is removes the specified breakpoint for the given session.

Arguments

TCHANDLE hSession	A handle identifying a particular debug session.
unsigned long nBreakpointId	Identifier of breakpoint to be removed. Returned by a set breakpoint function.

Returns

TC_OK	Operation was successful.
TC_FAILED	Unable to remove breakpoint.

Set DllTrace

void

```
tcSetDllTrace(  
    TCHANDLE          hSession, /* Debugger Session */  
    TCDEBUGFLAG       eFlag    /* enables/disables trace output  
*/  
);
```

Description

This function is enables or disables Debugger DLL output for the given session.

Arguments

TCHANDLE hSession	A handle identifying a particular debug session.
TCDEBUGFLAG eFlag	Specifies whether to enable or disable output.

Returns

TC_OK	Operation was successful.
TC_FAILED	Operation failed.

Note: This function is not currently used by the toolset, but it must exist. If this function is omitted from the debugger DLL, the toolset will not load the DLL successfully.

Index

A

audience 8

B

basic mode 9

building

- debugger integration DLL without
MainWin 17

C

callback functions 17

callback thread 17

calling sequence 17

Cleanup Debugger 22

Clear Breakpoint 28

commands

- Execute 14

- Load 12

- Reset 11

- Terminate 15

- Unload 13

Create Debug Session 20

D

debugger DLL API 19

- Cleanup Debugger 22

- Clear Breakpoint 28

- Create Debug Session 20

- Destroy Debug Session 21

- Event Callback Function 25

- Get DLL Capabilities 19

- Initialize Debugger 22

- Set Breakpoint At Function 27

- Set Breakpoint in File 27

- Set Callback 24

- Set DllTrace 29

- Set Source Search Path 26

- Start Debugger 23

- Stop Debugger 24

- debugger integration DLL 17

- debugger mode 10

- Destroy Debug Session 21

- DLL functions 18

E

Event Callback Function 25

Execute command 14

execute.pl 11

G

get DLL Capabilities 19

I

Initialize Debugger 22

L

Load command 12

load.pl 11

M

manual mode 9

menu commands 11

model compilation 7

modes

- basic 9

- debugger 10

- manual 9

- target control 9

O

observability
 adding support for 7

R

Registering Threads on Unix 17
Reset command 11
reset.pl 11
rules of operation 18

S

scripts 10
 target control 9
Set Breakpoint At Function 27
Set Breakpoint in File 27
Set Callback 24
Set DllTrace 29
Set Source Search Path 26
source code debugger integration 16
Start Debugger 23
Stop Debugger 24

T

target control
 adding support for 7
 basic mode 9
 calling sequence 17
 debugger mode 10
 defined 8
 Execute command 14
 general issues 16
 Load command 12
 manual mode 9
 menu commands 11
 model compilation 7
 modes 9
 overview 8
 Reset command 11
 rules of operation 18
 scripts 10

 Terminate command 15
 third-party source code debugger
 integration 16
 Unload command 13
target control scripts 9, 10
 defined 10
target observability
 defined 8
 overview 8
tcThreadCleanup 17
tcThreadInit 17
Terminate command 15
terminate.pl 11
threads
 registering on UNIX 17
thread-safe interface 17

U

UNIX
 Registering threads 17
Unload command 13
unload.pl 11