

Visual Execution Analyzer in Enterprise Architect

Enterprise Architect is an intuitive, flexible and powerful UML analysis and design tool for building robust and maintainable software.

This booklet explains the Visual Execution Analyzer (Debugger) feature of Enterprise Architect.



Visual Execution Analyzer in Enterprise Architect

© 1998-2010 Sparx Systems Pty Ltd

All rights reserved. No parts of this work may be reproduced in any form or by any means - graphic, electronic, or mechanical, including photocopying, recording, taping, or information storage and retrieval systems - without the written permission of the publisher.

Products that are referred to in this document may be either trademarks and/or registered trademarks of the respective owners. The publisher and the author make no claim to these trademarks.

While every precaution has been taken in the preparation of this document, the publisher and the author assume no responsibility for errors or omissions, or for damages resulting from the use of information contained in this document or from the use of programs and source code that may accompany it. In no event shall the publisher and the author be liable for any loss of profit or any other commercial damage caused or alleged to have been caused directly or indirectly by this document.

Printed: May 2010

Publisher

Sparx Systems

Managing Editor

Geoffrey Sparks

Technical Editors Geoffrey Sparks

Steve Meagher

Special thanks to:

All the people who have contributed suggestions, examples, bug reports and assistance in the development of Enterprise Architect. The task of developing and maintaining this tool has been greatly enhanced by their contribution.

I

Table of Contents

Foreword	1
Visual Execution Analyzer	2
Access and Use the Visual Execution Analyser	3
Structure of the Visual Execution Analyzer	
Model Driven Development Environment	
-	
Getting Started With The MDDE	
Prerequisites.	
Available Tools	
Workspace Layout General Workflow	
Basic Setup	
Managing Scripts	
Defining Script Actions	
Setting the Default Script	
Code Generation and Synchronization - Safeguards	
Code Editing For MDDE	
Build	
Add Commands	. 12
Recursive Builds	
Debugging	15
How it Works	. 15
Setup for Debugging	. 15
Operating System Specific Requirements	16
UAC-Enabled Operating Systems	17
WINE Debugging	18
Microsoft C++ and Native (C, VB)	19
Debug Symbols	
Java	
General Setup for Java	
Advanced Techniques	
Attach to Virtual Machine	
Internet Browser Java Applets	
Working with Java Web Servers JBOSS Server	
Apache Tomcat Server	
Apache Tomcat Server	
.NET	
General Setup for .NET	
Debug Assemblies	
Debug - CLR Versions	
Debug COM Interop	
Debug ASP .NET	
The Debug Window	
Breakpoint and Marker Management	
How Markers are Stored	
Setting Code Breakpoints	38
Setting Data Breakpoints	38
Debugging Actions	. 39
Displaying Windows	39
Start & Stop Debugger	
Debug Another Process	
Step Over Lines of Code	41

II

Step Into Function Calls	41
Step Out of Functions	42
View the Call Stack	42
View the Local Variables	43
View Content Of Long Strings	43
View Variables in Other Scopes	44
Inspect Process Memory	45
Break When a Variable Changes Value	
Show Loaded Modules	
Show Output from Debugger	
Debug Tooltips in Code Editors	
•	
Recording Actions	
Create Sequence Diagram of Call Stack	
Saving the Call Stack	
Searching in Files	
Search in Files	
Testing Command	
Add Testing Command	
Run Command	
Add Run Command	. 55
Deploy Command	56
Add Deploy Command	. 56
Execution Analysis	57
Recording Sequence Diagrams	57
How it Works	
Setup for Recording.	
Pre-Requisites	
Configure Recording Detail	
Enable Filter	
Record Arguments To Function Calls	
Record Calls To External Modules	
Record Calls to Dynamic Modules	
Limit Auto Recording	63
Enable Diagnostic Messages	
Advanced Techniques	64
Recording Activity for a Class	
Recording Activity for a Single Method	65
Place Recording Markers	. 66
Marker Types	66
Setting Recording Markers	70
The Breakpoints and Markers Window	71
Activate and Disable Markers	71
Working with Marker Sets	72
Differences to Breakpoints	
Control the Recording Session	
Auto-Recording	
Manual Recording	
Pause Recording	
Resume Recording	
Stop Capture	
Generating Sequence Diagrams	
The Recording History	
Generate a Diagram	
Diagram Features	
Saving Recording	
Add State Transitions	
Setup for Capturing State Changes	
The State Machine	77

Conte	nts	
Recording and Mapping State Changes		 7
Unit Testing		 8
Set Up Unit Testing		 8
Run Unit Tests		
Record Test Results		
Profiling Native Applications		
System Requirements		 8
Getting Started		
Start & Stop the Profiler		 8
Profiler Operation		
Setting Options		
Save and Load Reports		
Save Report in Team Review		
Object Workbench		 8
How it Works		 8
Workbench Variables		
Create Workbench Variables		
Invoke Methods		90

Index

© 1998-2010 Sparx Systems Pty Ltd

94

Foreword

This user guide provides an introduction to the Visual Execution Analyzer feature of Enterprise Architect.

Visual Execution Analyzer



The *Visual Execution Analyzer* provides facilities to model, develop, debug, profile and manage an application from within the modeling environment.

The Visual Execution Analyzer can generate a number of outputs, including:

- · Sequence Diagrams, recording live execution of an application, or specific call stacks
- State Transition Diagrams, a Sequence diagram with states, illustrating changes in data structures
- Profiler reports, showing application sequences and operation call frequency.

These outputs provide a better understanding of how your system works, enabling you to document system features and providing information on the sequence of events that lead to an erroneous event or an unexpected system behavior.

Note:

The Visual Execution Analyzer is available in the Enterprise Architect Professional, Corporate, Business and Software Engineering, System Engineering, and Ultimate editions.

1 Access and Use the Visual Execution Analyser

With the Visual Execution Analyzer, you can create and store custom scripts that specify how to build, test, run and deploy code associated with a package. You can investigate and manipulate the output of the debug process. The Analyzer also includes an Execution *Profiler*, which enables you to determine how the functions in an application are called and executed.

You access the Visual Execution Analyzer using the **Project | Execution Analyzer** menu option, or the context menu of the required package in the **Project Browser**. These menus provide a number of options to facilitate debugging, such as setting recording options or breakpoints.

The Visual Execution Analyzer can be used to:

3

- · Optimize existing system resources and understand resource allocation
- Ensure that the system is following the rules as designed
- · Produce high quality documentation that more accurately reflects system behavior
- · Understand how and why systems work
- Train new employees in the structure and function of a system
- Provide a comprehensive understanding of how existing code works
- · Identify costly or unnecessary function calls
- Illustrate interactions, data structures and important relationships within a system
- Trace problems to a specific line of code, system interaction or event
- Visualize why a sequence of events is important
- Establish the sequence of events that occur immediately prior to system failure.

4

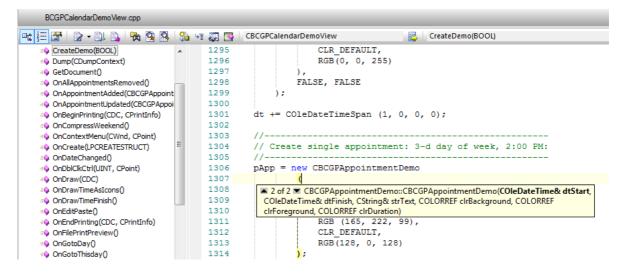
2 Structure of the Visual Execution Analyzer



The Visual Execution Analyzer comprises a Model Driven Development Environment and an Execution Analyzer.

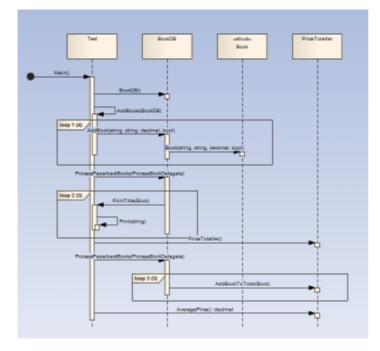
The Model Driven Development Environment 6 (MDDE) provides tools to design, build and debug an application:

- UML technologies and tools to model software (see Extending UML Using Enterprise Architect and UML Modeling With Enterprise Architect - UML Modeling Tool)
- Code generation tools to generate/reverse engineer source code (see Code Engineering Using UML Models)
- Tools to import source code and binaries (see Code Engineering Using UML Models)
- <u>Code editors that support different programming languages</u>
- Intellisense to aid coding (see Using Enterprise Architect UML Modeling Tool)
- Package scripts that enable a user to describe how to build, debug, test and deploy the application 9.



The Execution Analyzer [57] (EA) provides tools to visualize an existing application's behaviour:

- <u>Record sequence diagrams of application activities</u>
- <u>Capture State Transitions for a particular State Machine</u>
- Capture Stacktraces at points in execution 37
- Profiling tool to sample application activity 82
- Object Workbench 87
- Unit Testing 80



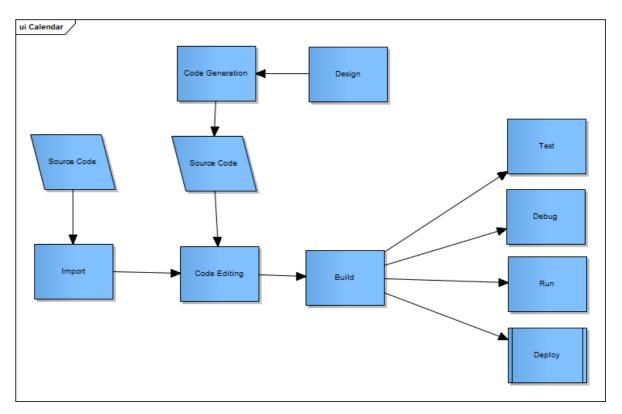
5



3 Model Driven Development Environment

The Model Driven Development Environment (MDDE) provides tools to design, build and debug an application.

The MDDE integrates code and model by providing options to either generate source code from the model or reverse engineer existing source code into a model. Source code and model can be synchronized in either direction.



The MDDE provides development environments for popular languages including:

- C++
- C
- Java
- Microsoft .NET family
- ADA
- Python
- Perl

Toolboxes provide for different modeling technologies.

Note:

Although you can generate and reverse engineer code in a range of languages, Execution Analysis debugging and recording are supported for the following platforms / languages only:

- Microsoft Windows Native C
- Microsoft Windows Native C++
- Microsoft Windows Visual Basic
- Microsoft .NET Family (C#, J#, VB)
- Sun Microsystems Java.

To use the MDDE, work through the following sections:

- Getting Started 7
- Basic Setup 8
- Code Engineering (see Code Engineering Using UML Models)
- Using Code Editors (see Using Enterprise Architect UML Modeling Tool)
- Build Application 12
- Debug 39
- <u>Test</u> 53
- Run 55
- Deploy 56

3.1 Getting Started With The MDDE

To quickly start development in the Model Driven Development Environment, check through the following topics:

- <u>Prerequisites</u> 7
- Available Tools
- Workspace Layouts
 8
- General Workflow

3.1.1 Prerequisites

Before using the Model Driven Development Environment:

- You should be using the correct edition: Enterprise Architect Professional, Corporate or Suite Editions.
- You should be connected to the required model.
- Relevant source code should be imported into the model.
- <u>Basic Setup</u>⁸ should be complete.

3.1.2 Available Tools

This section describes the tools available in the Model Driven Development Environment:

- Workspace Layouts
- Code Engineering (see Code Engineering Using UML Models)
- Using Code Editors (see Using Enterprise Architect UML Modeling Tool)
- Intellisense (see Using Enterprise Architect UML Modeling Tool)
- <u>Application Management</u>
- Debugger Management 15

3.1.3 Workspace Layout

You can choose from many predefined workspace layouts (see *Using Enterprise Architect - UML Modeling Tool*) depending on the tasks you perform. When you are familiar with the environment and controls available to you, you can create your own.

Workspace Toolbar



Predefined Workspace Layouts

3.1.4 General Workflow

In working with the Model Driven Development Environment, you apply the following workflow as a circular process, refining as necessary in each iteration.

- Configure and set up scripts 9
- Model Edit Build Debug Test Profile Deploy Document and Analyze.

3.2 Basic Setup

To use the execution tools of the Model Driven Development Environment - debugging, build and recording - it is necessary to record information about the application. This is achieved in Enterprise Architect through the use of Package Scripts.

A Package Script, when created, is naturally associated with the package that is currently selected.

A Package Script houses all the information the MDDE requires in order to provide support for tasks such as building the application, debugging and performing unit testing. A model can contain many Package Scripts. Each can build a separate application, or perhaps the same application but with different compilation options.

When you select a package or child Class in the Project Explorer, the Debug Management window displays any Package Scripts associated with that package. When you select a package root, the Debug Management window displays the scripts for the first package it finds under the root that has Scripts.

When you selected another package, the scripts displayed in the Debug Management window change also. You can force the scripts for a particular package to remain visible at all times by 'pinning 11' the package in the Debug Management window.

8

External Tools and Environment

If you plan on using any of the debugging features of the MDDE, you must have the appropriate Framework installed on your machine:

- The Java Runtime Environment for Java
- The .NET Framework for managed applications

Any Operating System Environment Variables such as \$PATH required by these kits should also be established.

3.2.1 Managing Scripts

9

In Enterprise Architect, any package within the UML Model can be configured to act as the 'root' of a source code project. By setting compilation scripts, xUnit commands, debuggers and other configuration settings for a package, all contained source code and elements can be built, tested or debugged according to the currently active configuration. Each package can have multiple scripts, but only one is active at any one time. The Package Build Scripts dialog enables you to create and manage those scripts.

To access the Package Build Scripts dialog, either:

- Press [Shift]+[F12]
- On the Debug toolbar, click on the drop-down arrow on the Scripts icon (the first icon on the left) and select the Package Build Scripts option
- · Select the Project | Execution Analyzer | Package Build Scripts menu option, or
- Right-click on a package in the Project Browser, and select the Execution Analyzer | Package Build Scripts context menu option.

Active	Name	Build	Test	Run	Debug	Deploy	Add
🗹 Exar	nple 1.6.0_03				Х		Edit
							Сору
							Import
							Export
							Delete
All Package S	cripts						
otions							
	📃 Use Liv	ve Code Genera	ation				ОК
							Cancel
efault Languag	ge: <pre><model e<="" pre=""></model></pre>	A faults	•				Help

The Package Build Scripts dialog shows which script is active for the current package, and whether or not the script contains Build, Test, Run, Debug and Deploy components. The current package is as selected in the Project Browser; if a different package is selected, different scripts are available and different breakpoints and markers are applied.

Note that you must close the Package Build Scripts dialog to select a different package in the Project Browser. However, if the Debug window is open ([Alt]+[8]) you can see which debugging configuration is available and selected, and which breakpoints and markers are displayed, as you change packages in the Project Browser.

- To create a new script, click on the Add button; the Build Script dialog displays.
- To modify an existing script, highlight the script name in the list and click on the Edit button.
- To copy a script with a new name, highlight the script name to copy and click on the **Copy** button; Enterprise Architect prompts you to enter a name for the new copy. Enter the new name in the dialog and click on the **OK** button. The new copy appears in the list and can be modified as usual.
- To delete a script, highlight the script name to delete, click on the **Delete** button, and click on the **OK** button.
- To export your scripts, click on the Export button to choose the scripts to export for this package.
- To import build scripts, click on the Import button to choose a .xml file of the scripts to import.

The **Default Language** field enables you to set the default language for generating source code for all new elements within this package and its descendents.

Select the **Use Live Code Generation** checkbox to update your source code instantly as you make changes to your model.

Click on the All Package Scripts button to open a new window that displays all scripts in the current project.

Once you have created new scripts or made changes to existing ones, click on the **OK** button to confirm the changes, otherwise click on the **Cancel** button to quit the Package Build Scripts dialog without saving any changes.

3.2.2 Defining Script Actions

Scripts are associated with a Package. When you create a Package Script you can define a number of actions.

If you plan to use any of the features of the Execution Analyzer, you must complete at least the Build and Debug tabs.

- <u>Build</u> 12
- Debug 15
- Test 53
- Run 55
- Deploy 56

Build Script		
Name:	Calendar]
Directory:	C: \Program Files \BCGSoft \BCGControlBarPro \Examples \BCGPCalendarDemo	
	est Run Debug Deploy Sequence Diagram Recording	
	s\devenv.com" /rebuild Debug BCGPCalendarDemo.sln	•
Captur	re Output Output Parser	T: Microsoft .NET ▼
		DK Cancel Help

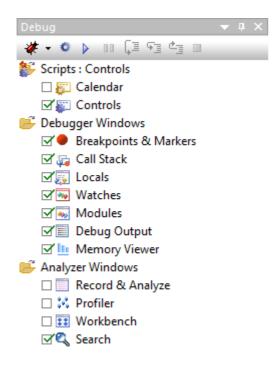
3.2.3 Setting the Default Script

Normally the target for any debugging session changes, tracking the package selected in the Project Explorer.

You can change this behaviour so that the scripts for a package remain selected in the Debug window. Use the context menu on the *Scripts* folder in the Debug window to either Pin or Unpin the currently-selected package.

	Package Build Scripts	Shift+F12
°	Find in Project Browser	
~	Pin Package	
0	Help	

When a package is pinned, the Debug window always displays the scripts defined for that package, and the debugger always uses the selected Package Script.



3.3 Code Generation and Synchronization - Safeguards

It is important that the model and source code are kept synchronized for the Visual Execution Analyzer to produce useful results.

Use the Code Generation tools to synchronize your model after any design changes or code editing (see *Code Engineering Using UML Models*).

Always build the application prior to any Execution Analysis session - debugging, recording or profiling.

3.4 Code Editing For MDDE

See the Code Editors topic in Using Enterprise Architect - UML Modeling Tool.

3.5 Build

The topics in this section describe how you specify the commands to build the project / package:

- Add Comands 12
- <u>Recursive Builds</u>

3.5.1 Add Commands

The Build tab enables you to enter multiple commands for building the current package. These commands are executed when you select the **Project | Execution Analyzer | Build** menu option. The following examples are for Java and .NET respectively.

Name:	JarLoader	
Directory:	C:\Benchmark\Java\JarLoader	
Build Te	est Run Debug Deploy Sequence Diagram Recording	
Enter the	path to the build application for the chosen compiler	
%JAVA %JAVA %JAVA %JAVA %JAVA %JAVA	A%\bin\javac" -cp %dasspath%;:;"-g JarLoader.java A%\bin\javac" -cp %dasspath%;:;"-g Base\TestBase.java A%\bin\jar" cfm Base.jar Base\Testbase.txt Base\TestBase.dass A%\bin\javac" -cp %dasspath%;:;"-g -cp Base.jar Base\Test1*.java A%\bin\jar" cfm Test1.jar Base\Test1\Test.txt Base\Test1\Test.dass A%\bin\javac" -cp %dasspath%;:;"-g -cp Base.jar Base\Test2*.java A%\bin\javac" -cp %dasspath%;:;"-g -cp Base.jar Base\Test2*.java A%\bin\jar" cfm Test2.jar Base\Test2\Test.txt Base\Test2\Test.dass	
📝 Captur	ire Output Output Parser: Ja	va SDK 🔹
	OK	Cancel Help

¹³ Model Driven Development Environment | Build

Name:	Example 1.6.0_03
Directory:	C:\Debugging\Assemblies\MyClassLibrary
Build Te	est Run Debug Deploy Sequence Diagram Recording
Enter the	path to the build application for the chosen compiler
C:\Progr	am Files \Microsoft Visual Studio 8 \Common 7 \IDE \devenv.com /Build Debug MyClassLibrary.sln
	·
🔽 Captur	re Output Output Parser: Microsoft .NET
	OK Cancel Help

Write your script in the large text box using the standard *Windows Command Line* commands. You can specify, for example, compiler and linker options, and the names of output files. The format and content of this section depends on the actual compiler, make system, linker and so on that you use to build your project. You can also wrap up all these commands into a convenient batch file and call that here instead.

If you select the **Capture Output** checkbox, output from the script is logged in Enterprise Architect's **Output** window. This can be activated by selecting the **View | System Output** menu option.

The **Output Parser** field enables you to define a method for automatically parsing the compiler output. If you have selected the **Capture Output** checkbox, Enterprise Architect parses the output of the compiler so that by clicking on an error message in the **Output** window, you directly access the corresponding line of code.

Notes:

- The command listed in this field is executed as if from the command prompt. Therefore, if the executable path or any arguments contain spaces, they must be surrounded by quotes.
- Throughout this dialog, you can use Local Paths in specifying paths to executables; see the Code Engineering Settings section in Code Engineering Using UML Models.

When you run the compile command inside Enterprise Architect, output from the compiler is piped back to the Output window and displayed as in the following illustration:

Jtput
Running build script - JarLoader
C:\Benchmark\Java\JarLoader> "C:\Program Files\Java\jdk1.6.0_07\bin\javac" -cp "C:\Program Files\Java\jdk1.6.0_07;;" -g JarL
Note: JarLoader.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
Note: JarLoader.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
C:\Benchmark\Java\JarLoader>"C:\Program Files\Java\jdk1.6.0_07\bin\javac" -cp "C:\Program Files\Java\jdk1.6.0_07;;"-g Base
:\Benchmark\Java\JarLoader>"C:\Program Files\Java\jdk1.6.0_07\bin\jar" cfm Base.jar Base\TestBase.txt Base\TestBase.class
:\\Benchmark\Java\JarLoader>"C:\Program Files\Java\jdk1.6.0_07\bin\javac" -cp "C:\Program Files\Java\jdk1.6.0_07;;" -g -cp
:\Benchmark\Java\JarLoader>"C:\Program Files\Java\jdk1.6.0_07\bin\jar" cfm Test1.jar Base\Test1\Test.txt Base\Test1\Test.c
C:\Benchmark\Java\JarLoader>"C:\Program Files\Java\jdk1.6.0_07\bin\javac" -cp "C:\Program Files\Java\jdk1.6.0_07;;" -g -cp
:\Benchmark\Java\JarLoader>"C:\Program Files\Java\jdk1.6.0_07\bin\jar" cfm Test2.jar Base\Test2\Test.txt Base\Test2\Test.c
:\Benchmark\Java\JarLoader Build completed with exit code 0
Image: System Script

If you double-click on an error line, Enterprise Architect loads the appropriate source file and positions the cursor on the line where the error has been reported.

3.5.2 Recursive Builds

For any project you can apply the command entered in the build script to all sub folders of the initial directory by specifying the token *%r* immediately preceding the files to be built. The effect of this is Enterprise Architect iteratively replaces the token with any subpath found under the root and executes the command again.

Name:	Java	a - Examp	ole 1				
Directory:	C:\D	ebugging	g\Java\E	ample			
Build		Dura	Dahua	Dealars	Received Discours Description	1	
	Fest	Run	Debug	Deploy	Sequence Diagram Recording		
Enter the	e path t	to the bu	ild applica	ation for t	the chosen compiler		
CulProc	ram Fil	ec laval	idk1 5 0	06\bin\ia	avac.exe -g %r*.java		
Capitog	jrain Fii	espaval	JUK 1. 5.0_	_uo (pin i)a	avac.exe -y % .javaj		

The output from this Java example is shown below:



Note:

The path being built is displayed along with the exit code.

3.6 Debugging

This section describes how you define the debugging actions:

- How it works 15
- <u>Setup for Debugging</u> 15
- Breakpoint and Marker Management 37
- Debugging Actions 39
- <u>Recording Actions</u>
 49

3.6.1 How it Works

The Model Driven Development Environment provides Debuggers for the following frameworks:

- Microsoft Native Code applications
- Microsoft .NET applications
- Java applications

To begin debugging:

- 1. Ensure the model is open.
- 2. Ensure Basic Setup has been completed for the Package or Project.
- 3. Ensure any source code for the areas of interest have been generated, or imported into the Model.
- 4. Ensure the application has been built. You can do this internally using the Build Script or you can build the application externally. The important thing is that the application is built on the latest versions of the source.
- 5. Ensure that the model and source are synchronized (see Code Engineering Using UML Models).
- 6. <u>Set breakpoints</u> 38⁻.
- 7. <u>Start</u> 39 the Debugger.

3.6.2 Setup for Debugging

To begin debugging you must specify

- The Debugger to use
- The Application path
- Runtime options, if applicable

The following example shows a .NET Debug script.

Build Te	st Run	Debug	Deploy	Sequence Diagram Recording	3		
Applica	tion (Enter p	oath)	Attac	h to process			
bin \debug	\example.ne	et.exe "C#	t" example	using ".Net 1.1"			*
							÷
Enter any	run time var	iables belo	w				
							*
							▼
Show Con	sole 🔽				Use Debugger:	Microsoft .NET 1.1	•

3.6.2.1 Operating System Specific Requirements

Important:

Please read the information provided in this topic.

Prerequisites

Creation of a Package Build Script and configuration of the Debug command in that script.

Supported Platforms

Enterprise Architect supports debugging on these platforms:

.Net

- Microsoft[™] .NET Framework 1.1 and later
- Language support: C, C#, C++, J#, Visual Basic

Note:

Debugging under Windows Vista (x64) - If you encounter problems debugging with Enterprise Architect on a 64-bit platform, you should build a Win32 platform configuration in Visual Studio; that is, do not specify **ANY-CPU**, specify **WIN32**.

Java

- Java 2 Platform Standard edition (J2SE) version 5.0
- J2EE JDK 1.4 and above
- Requires previous installation of the Java Runtime Environment and Java Development Kit from Sun Microsystems[™].

Debugging is implemented through the Java Virtual Machine Tools Interface (JVMTI), which is part of the Java Platform Debugger Architecture (JPDA). The JPDA is included in the J2SE SDK 1.3 and later.

Windows for Native Applications

Enterprise Architect supports debugging native code (C, C++ and Visual Basic) compiled with the Microsoft[™] compiler where an associated PDB file is available. Select **Microsoft Native** from the list of debugging platforms in your package script.

You can import native code into your model, and record the execution history for any Classes and methods. You can also generate Sequence diagrams from the resulting execution path.

Note:

Enterprise Architect currently does not support remote debugging.

3.6.2.1.1 UAC-Enabled Operating Systems

The Microsoft operating systems *Windows Vista* and *Windows 7* provide User Account Control (UAC) to manage security for applications.

The Enterprise Architect Visual Execution Analyser is UAC-compliant, and users of UAC-enabled systems can perform operations with the Visual Execution Analyser and related facilities under accounts that are members of only the *Users* group.

However, when attaching to processes running as services on a UAC-enabled operating system, it might be necessary to log in as an Administrator. To do this, follow the step below:

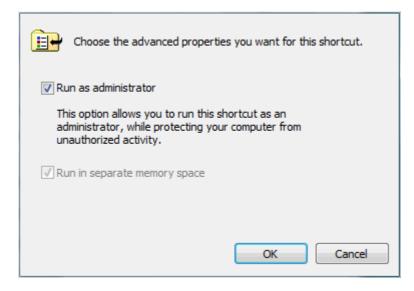
1. Before you run Enterprise Architect, right-click on the Enterprise Architect icon on the desktop and select the **Run as administrator** option.

Alternatively, edit or create a link to Enterprise Architect and configure the link to run as an administrator; follow the steps below:

1. Right-click on the Enterprise Architect icon and select the **Properties** menu option. The Enterprise Architect Properties dialog displays.

Security	Details	Previous Versions						
General	Shortcut	Compatibility						
Enterprise Architect								
Target type:	Application							
Target location:	EA							
Target:	"C:\Program Files\Sp	arx Systems\EA\EA.exe"						
Start in:								
Shortcut key:	None							
Run:	Normal window	_						
Comment:								
Open File Lo	Change le	con Advanced						
	ОК	Cancel Apply						

2. Click on the Advanced button. The Advanced Properties dialog displays.



- 3. Select the **Run as administrator** checkbox.
- 4. Click on the **OK** button, and again on the Enterprise Architect Properties dialog.

3.6.2.1.2 WINE Debugging

At the command line, run \$ winecfg.

Set the library overrides for dbghelp to (native, builtin), and accept the warning about overriding this DLL:

⚠	Changing the load order of this library is not recommended. Are you sure you want to do this?
	Yes No

Note:

If WINE crashes, the back traces may not be correct.

- 1. Set *dbghelp* to **native** by using *winecfg*.
- 2. Copy the application source code plus executable(s) to your bottle. (The path must be the same as the compiled version; that is:

If Windows source = C:\Source\SampleApp, under Crossover it must be C:\Source\SampleApp.)

- 3. Copy any Side-By-Side assemblies that are used by the application.
- 4. Import the source code into Enterprise Architect. (Optional.)
- 5. <u>Create a build script</u> 12 on a package.

Set the path of the application on the Debug tab, and set Use Debugger to Microsoft Native.

- 6. Open the Profiler 82 (View | Execution Analyzer | Profiler).
- 7. Click on the **Launch** button (first button on the **Profiler** window).

If the sample didn't start, click on the **Sampling** button (third button on the Profiler window).

- 8. Once you have finished profiling, shut down the application (not Enterprise Architect).
- 9. View the Sampler report by clicking the View Report button (fifth button on the Profiler window).

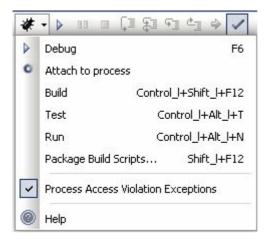
Tips:

- If you are using MFC remember to copy the debug side-by-side assemblies to the C:\window\winsxs directory.
- To add a windows path to WINE, modify the Registry entry:

HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session Manager\Environment

Access Violation Exceptions

Due to the manner in which WINE handles direct drawing and access to DIB data, an additional option is provided on the Debug window toolbar drop-down menu to ignore or process access violation exceptions thrown when your program directly accesses DIB data.



Select this option to catch genuine (unexpected) access violations; deselect it to ignore expected violations. As the debugger cannot distinguish between expected and unexpected violations, you might have to use trial and error to capture and inspect genuine program crashes.

3.6.2.2 Microsoft C++ and Native (C, VB)

The example script below is configured to enable debugging of a C++ project built in Microsoft Visual Studio 2005 or 2008.

You can debug native code only if there is a corresponding PDB file for the executable. You normally create the PDB file as a result of building the application.

The build should include full debug information and there should be no optimizations set.

Name:	Subway	
Directory:	C: \Tutorial \Native \Subway	
Build Te	est Run Debug Deploy Sequence Diagram Recording	
Applic	ation (Enter path)	
debug\si	ibway.exe	*
		·
Enter any	y run time variables below	
		*
		·
Show Cor	nsole 🗌 Use Debugger: Microsoft Native	-
	OK Cancel	Help

The script must specify two things to support debugging:

- The path to the executable
- Microsoft Native as the debugging platform.

3.6.2.2.1 Debug Symbols

For Applications built using *Microsoft Platform SDK* Debug Symbols are written to an Application PDB file when the Application is built.

The *Debugging Tools for Windows*, an API used by the Visual Execution Debugger, uses these symbols to present meaningful information to Execution Analyzer controls.

These symbols can easily get out of date and cause errant behaviour. The debugger might highlight the wrong line of code in the editor whilst at a breakpoint. It is therefore best to ensure the application is built prior to any debugging or recording session.

The debugger must inform the API how to reconcile addresses in the image being debugged. It does this by specifying a number of paths to the API that tell it where to look for PDB files. The API automatically picks up the path to the main image PDB from the image itself.

For system DLLs (kernel32, mfc90ud ...) for which no debug symbols are found, the Call Stack shows some frames with module names and addresses only .

You can supplement the symbols translated by passing additional paths to the API. To do this there must be a Package Script selected and it must have the Native debugger specified.

You pass additional symbol paths in a semi-colon separated list in the **Enter any runtime variables..**. field of the Debug tab, as illustrated below.

21 Model Driven Development Environment | Debugging

Application (Enter path)	Attach to process				
debug\stepping.exe				*	
				Ŧ	
Enter any run time variables belo	W				
c: \window \symbol				*	
				Ŧ	
Show Console	Use	Debugger:	Microsoft Native	•	

3.6.2.3 Java

This section describes how to configure Enterprise Architect for debugging Java applications and Web Servers.

3.6.2.3.1 General Setup for Java

This is the general setup for debugging Java applications:

Name:	Example 1.6.0_03	
Directory:	C:\benchmark\java\example1	
Build Te	est Run Debug Deploy Sequence Diagram Recording	
Applic	ation (Enter path)	
source.e	xample "1" "2" "3"	*
Enter any	y run time variables below	
jre=c:\Pi	rogram Files\Java\dk1.6.0_03,-Djava.class.path=%classpath1603%;c:\benchmark\java\example1	*
Show Cor	nsole 🔲 Use Debugger: Java	•
	OK Cancel	Help

Option	Use to
Application (Enter path)	Identify the fully qualified Class name to debug, followed by any arguments. The Class must have a method declared with the following signature: public static void main(String[]); The debugger calls this method on the Class you name. In the example above, the parameters 1 , 2 and 3 are passed to the method. You can also debug a Java application by <u>attaching to an existing Java</u> <u>process</u> 22 ² .
Enter any run time variables below	Type any required command line options to the Java Virtual Machine. You also must provide a parameter (jre) that is a path to be searched for the jvm.dll. This is the DLL supplied as part of the Java runtime environment or Java JDK from Sun Microsystems TM (see <u>Debugging</u> 57). In the example above, a virtual machine is created with a new Class path
	property that comprises any paths named in the environment variable <i>classpath 1603</i> plus the single path "C:\benchmark\java\example1". If no Class path is specified, the debugger always creates the virtual machine with a Class path property equal to any path contained in the environment variable plus the path entered in the default working directory of this script.
	Note: If source files and .class files are located under different directory trees, the Class path property MUST include both root path(s) to the source and root path(s) to binary class files.
Show Console	Create a console window for Java. If no console window is required, leave blank.
Use Debugger	Select Java.

3.6.2.3.2 Advanced Techniques

In addition to the standard Java debugging techniques, you can also:

- Attach to a Virtual Machine 22
- Debug Internet Browser Java Applets. 23

You can debug a Java application by attaching to an existing Java process.

However, the Java process requires a specific startup option specifying the Sparx Systems Java Agent. The format of the command line option is:

-agentlib:SSJavaProfiler80

or:

-agentpath:"c:\program files\sparx systems\ea\SSJavaProfiler80"

The example below is for attaching to the *Tomcat Webserver*. Select the **Attach to process** radio button, and then the keyword **Attach** is all that you have to enter. This keyword causes the debugger to prompt you for a process at runtime.

Note:

The Show Console checkbox has no effect when attaching to an existing virtual machine.

23

Name:	Java Web Server	
Directory:	C:\benchmark\java\web	
Build Te	est Run Debug Deploy Sequence Diagram Recording	
Applic	cation (Enter path) Attach to process 	
		*
		▼
Enter an	y run time variables below	
		*
		▼
Show Co	insole 🗌 Use Debugger: Java	•
	OK Cancel	Help

No run time variables are necessary when attaching as these are specified as startup parameters to the process.

This topic describes the configuration requirements and procedure for debugging Java Applets running in a browser from Enterprise Architect.

The procedure requires you to attach to the browser process hosting the Java Virtual Machine (JVM) from Enterprise Architect, as summarized below:

- 1. Ensure binaries for the applet code to be debugged have been built with debug information.
- 2. Configure the JVM using the Java Control Panel.

General	Update	Java	Security	Advanced				
🗆 Java	Applet Ru	intime S	ettinas —					_
	Runtime settings are used when an applet is executed in the browser.							
							View	
	Applicatio	n Dunti	me Setting	-				_
			_		ation or an	nlat is lau	inched using the	
			ng Protoco			piecis iau	incrica asing the	
							View	
								_
					ОК	Cance	el Apply	

3. In the Java Applet Runtime Settings panel, click on the View button. The Java Runtime Settings dialog displays.

	🚣 Java Runtim	e Setting	S COLOR	Ration alling an and due a line splitator or split of an field any file	×
ſ	Java Runtime V	ersions			
	Product Name	Version	Location	Java Runtime Parameters	
	JRE	1.7.0	C:\SunJava\jre1.7.0	-agentpath:c:\program files\sparx systems\ea\ssjavaprofiler80 -Djava.class.path="%classpat	h%;c:\myapplets;"
				OK	Cancel
L		_			

25 Model Driven Development Environment | Debugging

- 4. Click on the appropriate entry and click on the OK button to load the Sparx Systems Agent.
- 5. Import source code into the Enterprise Architect model, or synchronize existing code. (See Code Engineering Using UML Models.)
- 6. Create or modify the Package Build Script ³ to specify the option for attaching to the process.
- 5. Set breakpoints 37.
- 6. Launch the browser.
- 7. Attach to the browser process from Enterprise Architect.

Note that the *class.path* property specified for the JVM includes the root path to the applet source files. This is necessary for the Enterprise Architect debugger to match the execution to the imported source in the model.

3.6.2.3.3 Working with Java Web Servers

This topic describes the configuration requirements and procedure for debugging Java web servers such as $\underline{\text{JBOSS}}$ and Apache Tomcat (both $\underline{\text{Server}}$ shows and $\underline{\text{Server}}$ configuration and $\underline{\text{Windows Service}}$ configuration) in Enterprise Architect.

The procedure involves attaching to the process hosting the Java Virtual Machine from Enterprise Architect, as summarized below:

- 1. Ensure binaries for the web server code to be debugged have been built with debug information.
- 2. Launch the server with the Virtual Machine startup option 25 described in this topic.
- 3. Import source code into the Enterprise Architect Model, or synchronize existing code.
- 4. Create or modify the Package Build Script 25 to specify the **Debug** option for attaching to the process.
- 5. Set breakpoints. 37
- 6. Launch the client.
- 7. Attach to the process from Enterprise Architect.

Server Configuration

The configuration necessary for the web servers to interact with Enterprise Architect must address the following two essential points:

- Any VM to be debugged, created or hosted by the server must have the Sparx Systems Agent SSJavaProfiler80 command line option specified in the VM startup option (that is: -agentlib:SSJavaProfiler80)
- The CLASSPATH, however it is passed to the VM, must specify the root path to the package source files.

The Enterprise Architect debugger uses the java.class.path property in the VM being debugged, to locate the source file corresponding to a breakpoint occurring in a Class during execution. For example, a Class to be debugged is called:

a.b.C

This is located in physical directory:

C:\source\a\b

So, for debugging to be successful, the CLASSPATH must contain the root path:

c:\source.

Package Script Configuration

Using the <u>Debug tab</u> 2^{+} of the <u>Build Script</u> 2^{-} dialog, create a script for the code you have imported and specify the following:

- Select the Attach to process radio button, and in the field below type attach.
- In the Use Debugger field, click on the drop-down arrow and select Java.

Name:	Java Web Server	
Directory:	C:\Benchmark\Java\Web	
Build Te	st Run Debug Deploy Sequence Diagram Recording	
	ation (Enter path) Attach to process 	
attach		*
		·
Enter any	run time variables below	
		*
		▼
Show Cor	sole 🔽 Use Debugger: Java	•
	OK Cancel	Help

All other fields are unimportant. The **Directory** field is normally used in the absence of any Class path property.

Debugging

First ensure that the server is running, and that the server process has loaded the Sparx Systems Agent DLL SSJavaProfiler80.DLL (use *Process Explorer* or similar tools to prove this).

Launch the client and ensure the client executes. This must be done before attaching to the server process in Enterprise Architect.

After the client has been executed at least once, return to Enterprise Architect, open the source code you imported and set some breakpoints 37.

Click on the **Run Debugger** 15 button in Enterprise Architect. The Attach To Process dialog displays.

27

PID		Name	Path	1
	344	inetinfo.exe	C:\Windows\System32\inetsrv\inetinfo.exe	
	936	sqlservr.exe	C:\Program Files\Microsoft SQL Server\MSSQL.1\MSSQL\Binn\sqlservr.exe	
	1356	nod32krn.exe	C:\Program Files\Eset\nod32krn.exe	
	1908	svchost.exe	C:\Windows\System32\svchost.exe	
	648	sqlbrowser.exe	C:\Program Files\Microsoft SQL Server\90\Shared\sqlbrowser.exe	
	1500	sqlwriter.exe	C:\Program Files\Microsoft SQL Server\90\Shared\sqlwriter.exe	
	1668	svchost.exe	C:\Windows\System32\svchost.exe	
	2036	vds.exe	C:\Windows\System32\vds.exe	
	2056	vmnat.exe	C:\Windows\System32\vmnat.exe	
=	2084	svchost.exe	C:\Windows\System32\svchost.exe	
	5368	w3wp.exe	C:\Windows\System32\inetsrv\w3wp.exe	1
	2100	svchost.exe	C:\Windows\System32\svchost.exe	
	2136	vmnetdhcp.exe	C:\Windows\System32\vmnetdhcp.exe	
	2220	vmware-authd.exe	C:\Program Files\VMware\VMware Player\vmware-authd.exe	
	3752	WmiApSrv.exe	C:\Windows\System32\wbem\WmiApSrv.exe	1
+	2128	explorer.exe	C:\Windows\Explorer.EXE	
	3872	SearchIndexer.exe	C:\Windows\System32\SearchIndexer.exe	
	2956	SMSvcHost.exe	C:\Windows\Microsoft.NET\Framework\v3.0\Windows Communication Fou.	
	2568	iexplore.exe	C:\Program Files\Internet Explorer\jexplore.exe	
	5668	avant.exe	C:\Program Files\Avant Browser\avant.exe	
•				

Click on the **OK** button. A confirmation message displays in the Debug Output window, stating that the process has been attached.

The breakpoints should remain enabled (bright red). If the breakpoints fail, and contain either an exclamation mark or a question mark, then either the process is not hosting the SSJavaprofiler80 Agent or the binaries being executed by the server are not based on the source code. If so, check your configuration.

Consider the JBoss example below. The source code for a simple servlet is located in the directory location:

Core a coc	al Disk (C:) 🕨 be	nchmark 🕨 Java I	JBOSS 🕨	Inventory + co	om 🕨 inventory	► dto 🕶 🐓
🄄 Organize 👻 📗	/iews 🔻 🚷 Bi	ım	_	_		_
Favorite Links	Name	Date modified	Туре	Size		
Documents	🖹 carDTO.ja	va				

The binaries executed by JBOSS are located in the JAW.EAR file in this location:

n	build > distribution	JBOSS > 03b-dad	isk (C:) 🕨 JBC	🖉 🗢 📕 🗢 🕹	$\bigcirc \bigcirc$	
		Burn	ve 🚽 🙆 Bu	Organize 👻 🖽 Viev		
		C Dum	/s • 🕥 00			
	D I IC I	A	N		Favorite Links	
	Date modified		Name	vorite Links		
	2/07/2000 1-44 DM		inw ear			
	5/07/2009 1.44 PW	w.edi	jaw.cai	Documents		
	Date modified 3/07/2009 1:44 PM	~	vs ▼ 🔮 Bu Name 😭 jaw.ear	Documents	Favorite	

The Enterprise Architect debugger has to be able to locate source files during debugging. To do this it also uses the CLASSPATH, searching in any listed path for a matching JAVA source file, so the CLASSPATH must include a path to the root of the package for Enterprise Architect to find the source during debugging.

The following is an excerpt from the command file that executes the JBOSS server. Since the Class to be

debugged is at com/inventory/dto/carDTO, the root of this path is included in the JBOSS classpath.

```
RUN.BAT

set SOURCE=C:\Benchmark\Java\JBOSS\Inventory

set JAVAC_JAR=%JAVA_HOME%\lib\tools.jar

if "%JBOSS_CLASSPATH%" == ""

( set JBOSS_CLASSPATH=%SOURCE%;%JAVAC_JAR%;%RUNJAR%;

)

else

( set JBOSS_CLASSPATH=%SOURCE%;%JBOSS_CLASSPATH%;%JAVAC_JAR%;%RUNJAR%;

)

set JAVA_OPTS=%JAVA_OPTS% -agentpath:"c:\program files\sparx systems\ssjavaprofiler80"
```

This configuration is for the same application as outlined in the <u>JBOSS server</u> on figuration topic.

There are two things to notice of importance.

- The Java VM option: -agentpath:c:\program files\sparx systems\ea\ssjavaprofiler80
- The addition to the Class path property of the path to the source code: C:\JBOSS\03b-dao\common\src;

General Log On Logging	Java Startup Shutdown						
Use default							
Java Virtual Machine:							
C:\SunJava\jre1.7.0\bin\dient\jvm.dll							
Java Classpath:							
C:\JBOSS\03b-dao\common\src;C:\Java\jdk1.6.0_07\jre;C:\Java\jdk1.6.							
Java Options:							
-Djava.util.logging.config	ger =org.apache.juli.ClassLoaderLogManage g.file =C:\Program Files\Apache Software Fou illes\sparx systems\ea\ssjavaprofiler 80						
Initial memory pool:	MB						
Maximum memory pool:	MB						
Thread stack size:	КВ						
	OK Cancel Apply						

For users running Apache Tomcat as a Windows[™] service, it is important to configure the service to enable interaction with the Desktop. Failure to do so causes debugging to fail within Enterprise Architect.

General Log On Recovery Dependence	ies					
Log on as:						
Ocal System account						
Allow service to interact with desktop						
This account:	Browse					
Password:						
Confirm password:						
Help me configure user account log on opti	ons					
You can enable or disable this service for th						
Hardware Profile	Service					
Undocked Profile	Enabled					
Troubleshooting using hardware profiles.	Enable Disable					

Select the Allow service to interact with desktop checkbox.

3.6.2.4 .NET

This section describes how to configure Enterprise Architect for debugging .NET applications. It covers:

- General Setup 30
- Debug Assemblies
- Debug CLR Versions 31
- Debug COM Interop 32
- Debug ASP .NET 32

3.6.2.4.1 General Setup for .NET

This is the general setup for debugging .NET applications:

Build	Test R	un Deb	Jg Deploy	Sequence D)iagram Recording			
⊚ Ap	oplication (Er	nter path)	O Att	ach to process				
bin\d	lebug\examp	ble.net.exe	"C#" exam;	ole using ".Net	1.1			* •
Enter	any run tim	e variables	pelow					
								•
								···
Show	Console					Use Debugger:	Microsoft .NET 1.1	•

Option	Use to		
Application (Enter path)	Select and enter either the full or the relative path to the application executable, followed by any command line arguments.		
Enter any runtime variables below	Type any required command line options, if debugging a single <u>.NET</u>		
Show Console	Create a console window for the debugger. Not applicable for attaching to a process.		
Use Debugger	Select the debugger to suit the .NET Framework under which your application runs.		

Note:

If you intend to debug managed code using an unmanaged application, please see the <u>Debug - CLR</u> <u>Versions</u> 3^{+} topic.

3.6.2.4.2 Debug Assemblies

Enterprise Architect permits debugging of individual assemblies.

The assembly is loaded and a specified method invoked. If the method takes a number of parameters, these can be passed.

Constraints

Debugging of assemblies is only supported for .NET version 2.

The following image is of a Build Script configured for debugging a .NET assembly.

Name:	MyClassLibrary	
Directory:	C:\benchmark\dotnet\csharp\Assemblies\MyClassLibrary	
Build Te	st Run Debug Deploy Sequence Diagram Recording	
Applic	ation (Enter path)	
bin\Debu	gWyClassLibrary.dll	*
		·
Enter any	run time variables below	
MyClass	.ibrary.CJohn,SetAge,23	*
		-
Show Cor	Isole 🔽 Use Debugger: Microsoft .NET 2.0, 3.0, 3.	5 🔻
	OK Cancel	Help

Notice the **Enter any run time variables below** field. This field is a comma-delimited list of values that must present in the following order:

type_name, method_name, { method_argument_1, method_argument2,....}

where:

- type_name is the qualified type to instantiate
- method_name is the unqualified name of the method belonging to the type that is invoked
- the argument list is optional depending on the method invoked.

The information in this field is passed to the debugger.

3.6.2.4.3 Debug - CLR Versions

Please note that if you are debugging managed code using an unmanaged application, the debugger might fail to detect the correct version of the Common Language Runtime (CLR) to load. You should specify a config file if you don't already have one for the debug application specified in the *Debug* command of your script. The config file should reside in the same directory as your application, and take the format:

name.exe.config

where name is the name of your application.

The version of the CLR you should specify should match the version loaded by the managed code invoked by the debuggee.

Sample config file:

<configuration> <startup> <requiredRuntime version="version "/> </startup> </configuration>

where version is the version of the CLR targeted by your plugin or COM code.

For further information, see http://www.msdn2.microsoft.com/en-us/library/9w519wzk.aspx.

3.6.2.4.4 Debug COM Interop

Enterprise Architect enables you to debug .NET managed code executed using COM in either a Local or an In-Process server.

This feature is useful for debugging Plugins and ActiveX components.

- 1. Create a package in Enterprise Architect and import the code to debug. See *Code Engineering Using UML Models*.
- 2. Ensure the COM component is built with debug information.
- 3. Create a Script for the Package.
- 4. In the Debug tab, you can elect to either attach to an unmanaged process or specify the path to an unmanaged application to call your managed code.

Build Test Run Debug Deploy Sequence Diagram Recording	
Application (Enter path) O Attach to process	
bin\debug\consoleapplication1.exe	•
Enter any run time variables below	
Show Console 🔽 Use Debugger: Microsoft .NET 1.1	·

5. Add breakpoints in the source code to debug.

Attach to an Unmanaged Process

- If an In-Process COM server, attach to the client process or
- If a Local COM Server, attach to the server process.

Click on the Debug window Run button (or press [F6]) to display a list of processes from which you can choose.

Important:

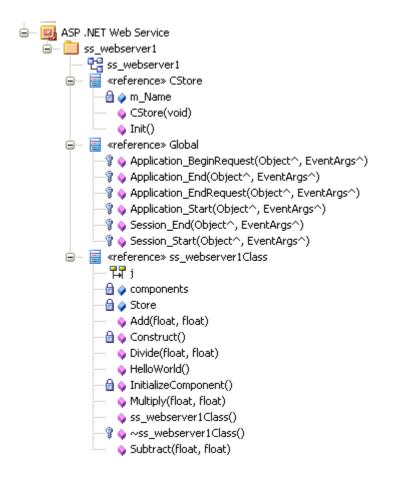
Detaching from a COM interop process you have been debugging terminates the process. This is a known issue for Microsoft .NET Framework, and information on it can be found on many of the MSDN .NET blogs.

3.6.2.4.5 Debug ASP .NET

Debugging for web services such as ASP requires that the Enterprise Architect debugger is able to attach to a running service.

Begin by ensuring that the directory containing the ASP .NET service project has been imported into Enterprise Architect and, if required, the web folder containing the client web pages. If your web project directory resides under the website hosting directory, then you can import from the root and include both ASP code and web pages at the same time.

The following image shows the project tree of a web service imported into Enterprise Architect.



It is necessary to launch the client first, as the ASP .NET service process might not already be running. Load the client by using your browser. This ensures that the web server is running. The only difference to a debug script for ASP is that you specify the **attach** keyword in your script, as follows:

Name:	Webservice	
Directory:	C:\Inetpub\wwwroot\ss_webserver1	
Build Te	est Run Debug Deploy Sequence Diagram Recording	
C Applic	ation (Enter path) Attach to process 	
attach		*
		·
Enter an	y run time variables below	
		A
		▼
Show Cor	nsole 🔲 Use Debugger: Microsoft .NET 2.0, 3.0, 3.5	5 🕶
	OK Cancel	Help

Build	Test	Run	Debug	Deploy	Sequence Diagram Recording	
Catao I			مناحده امان		ha ahaana aawataa	
Entert	ne patn	to the bu	ilia applica	ation for t	he chosen compiler	
dever	nv.com /	build deb	ug ss_we	bserver 1	.sln	
						
Build	Test	Run	Debug	Deploy	Sequence Diagram Recording	
Enter t	he nath	to the co	mpiled ap	plication		

http://localhosts/ss_webserver1/

When you start the debugger (click on the <u>Debug window</u> **Run** button) the Attach To Process dialog displays.

© 1998-2010 Sparx Systems Pty Ltd

PID		Name	Path	•
	344	inetinfo.exe	C:\Windows\System32\inetsrv\inetinfo.exe	
	936	sqlservr.exe	C: \Program Files \Microsoft SQL Server \MSSQL. 1 \MSSQL \Binn \sqlservr.exe	
	1356	nod32krn.exe	C:\Program Files\Eset\nod32krn.exe	
	1908	svchost.exe	C:\Windows\System32\svchost.exe	
	648	sqlbrowser.exe	C:\Program Files\Microsoft SQL Server\90\Shared\sqlbrowser.exe	
	1500	sqlwriter.exe	C:\Program Files\Microsoft SQL Server\90\Shared\sqlwriter.exe	
	1668	svchost.exe	C:\Windows\System32\svchost.exe	
	2036	vds.exe	C:\Windows\System32\vds.exe	
	2056	vmnat.exe	C:\Windows\System32\vmnat.exe	
-	2084	svchost.exe	C:\Windows\System32\svchost.exe	
	5368	w3wp.exe	C:\Windows\System32\inetsrv\w3wp.exe	
	2100	svchost.exe	C:\Windows\System32\svchost.exe	1
	2136	vmnetdhcp.exe	C:\Windows\System32\vmnetdhcp.exe	
	2220	vmware-authd.exe	C:\Program Files\VMware\VMware Player\vmware-authd.exe	1
	3752	WmiApSrv.exe	C:\Windows\System32\wbem\WmiApSrv.exe	
+	2128	explorer.exe	C:\Windows\Explorer.EXE	
	3872	SearchIndexer.exe	C:\Windows\System32\SearchIndexer.exe	
	2956	SMSvcHost.exe	C:\Windows\Microsoft.NET\Framework\v3.0\Windows Communication Fou.	
	2568	iexplore.exe	C:\Program Files\Internet Explorer\jexplore.exe	
	5668	avant.exe	C:\Program Files\Avant Browser\avant.exe	l
۱			III •	

Note that the name of the process varies across Microsoft operating systems.; check the *ASP*.*NET SDK* for more information. The image above shows the IIS process *w3wp.exe*, which is the name of the process that runs under Windows Vista.

On Windows XP, the name of the process is something like *aspnet_wp.exe*, although the name could reflect the version of the .NET framework that it is supporting. There can be multiple ASP.NET processes running under XP; you must ensure that you attach to the correct version, which would be the one hosting the .NET framework version that your application runs on. Check the *web.config* file for your web service to verify the version of .NET framework it is tied to.

The Debug window **Stop** button should be enabled and any <u>breakpoints</u> should be red, indicating they have been bound.

Note:

35

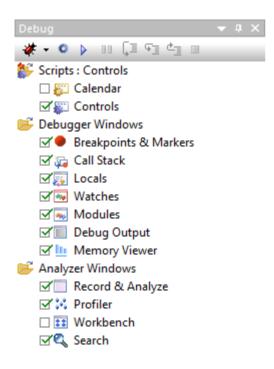
Some breakpoints might not have bound successfully, but if none at all are bound (indicated by being dark red with question marks) something has gone out of sync. Try rebuilding and re-importing source code.

You can set breakpoints at any time in the web server code. You can also set breakpoints in the ASP web page(s) if you imported them.

3.6.3 The Debug Window

The Debug window gives access to the scripts and windows of the debug facility.

To access the Debug window, select the View | Execution Analyzer | Debugger menu option.



The Debug Window has three top-level folders:

Scripts - The Scripts <Package Name> folder lists the scripts available for the currently-selected package, the first in the list being, by default, the active script that is executed when you start debugging, as indicated by the selected checkbox. If you are using recording markers, this is also the script that determines what recording options are applied set. If you want to execute a different script, select the appropriate checkbox. The context menu for each script provides further scripting options, such as Debug, Build, Test and Edit.

You can pin the package scripts so that they remain listed in the Debug window even if you select a different package. To do this, right-click on the folder title and select the **Pin Package** context menu option; the Scripts folder icon changes. To unpin the scripts, right-click on the folder title and deselect the **Pin Package** option.

- The *Debugger Windows* folder lists the debug windows, which you can display or hide by selecting or deselecting the checkbox against each one. If the window is docked, you can bring it to the front by clicking on the window name:
 - <u>Breakpoints & Markers</u> lists any breakpoints placed in the package source code, along with their status (enabled/disabled), line number, and the physical source file in which they are located
 - <u>Call Stack</u> 42 shows the position of the debugger in the code; clicking on the > button advances the stack through the code until the next breakpoint is reached
 - Locals 43 shows the local variables defined in the current code segment, their type and value
 - <u>Watches</u> 44 shows the values of static and globally scoped expressions you have entered
 - Modules 47 displays all the modules loaded during a debug session
 - <u>Debug Output</u> 47 displays output from the debugger including any messages output by the debugged process, such as writes to standard output.
- The Analyzer Windows folder lists the advanced control windows of the Execution Analyzer, which you can display or hide by selecting or deselecting the checkbox against each one:
 - <u>Record & Analyze</u> 74 records any activity that takes place during a debug session; once the activity has been logged, Enterprise Architect can use it to create a new Sequence diagram
 - <u>Profiler</u> 2 opens the Profiler window to sample an application
 - Workbench 87 enables you to create instances of .NET and Java Classes
 - Search 51 enables you to search for text in files.

You can dock and combine the windows to suit your working requirements; see the Arrange Windows and Menus section in Using Enterprise Architect - UML Modeling Tool.

3.6.4 Breakpoint and Marker Management

Breakpoints work in Enterprise Architect much like in any other debugger. Adding a breakpoint notifies the debugger to trap code execution at the point you have specified. When a breakpoint is encountered by a thread of the application being debugged, the source code is displayed in an editor window, and the line of code where the breakpoint occurred is highlighted.

Selecting a different package in the project affects which breakpoints are displayed.

Note:

37

The debugger does not stop automatically. It runs to completion unless it encounters a breakpoint.

An Enterprise Architect model maintains breakpoints for every package having a *Build Script - Debug* command. Breakpoints themselves are listed in their own Breakpoints & Markers window (**View | Execution** Analyzer | Breakpoints & Markers).

Breakpoints & Mar	kers		×
In the second se	‡ No	•ne 🔹 🖹 🔹 🔞	
Enabled	Line	Source	Details
• 🗹	15	Y:\Dev\Roy\ClassLib.java::15	
	36	Y:\Dev\Roy\ClassLib.java::36	
	48	Y:\Dev\Roy\ClassLib.java::48	

Breakpoint States

	DEBUGGER STATE	
	Running	Not running
•	Bound	Enabled
	Disabled	Disabled
0	Not bound - this usually means that the DLL is not yet loaded or was not built with debug information	N/a
0	Failed - this usually means a break could not be set at this time, and can occur when the source file is newer or older than that used to build the application.	N/a

Delete, Disable and Enable Breakpoints

To delete a specific breakpoint, either:

- If the breakpoint is enabled, click on the red breakpoint circle in the left margin of the Source Code Editor
- Right-click on the breakpoint marker in the editor and select the appropriate context menu option, or
- · Select the breakpoint in the Breakpoints & Markers tab and press [Delete].

Whether you are viewing the *Breakpoints* folder or the <u>Breakpoints</u> & <u>Markers</u> window, you can right-click on an existing breakpoint and select a context menu option either to delete it or to convert it to a <u>start recording</u> <u>marker or end recording marker</u> 6

You can also delete all breakpoints by clicking on the Delete all breakpoints button on the Breakpoints &

Markers window toolbar (3).

To disable a breakpoint, deselect its checkbox on the Breakpoints & Markers window or, to disable all

breakpoints, click on the **Disable all breakpoints** button in the toolbar (¹). The breakpoint is then shown as an empty grey circle. Select the checkbox or use the **Enable all breakpoints** button to enable it again (¹).

3.6.4.1 How Markers are Stored

Breakpoints created that are not part of any set are maintained in an external file for the current model.

The file format is as follows:

path\guid.brkpt

where:

• path = The O/S application data directory for each user

• guid = model Guid.

Marker Sets are stored in the model and are available to all users of the Model.

3.6.4.2 Setting Code Breakpoints

To set breakpoints for a code segment:

- 1. Open the model code to debug.
- 2. Find the appropriate code line and click in the left margin column. A solid red circle in the margin indicates that a breakpoint has been set at that position.

If the code is currently halted at a breakpoint, that point is indicated by a blue arrow next to the marker.

```
6 int _tmain(int argc, _TCHAR* argv[])
7 □ {
8 CTest Test(_T("Model"), CTest::Regression);
9 return Test.Run();
10 }
```

3.6.4.3 Setting Data Breakpoints

Data breakpoints can currently only be set by right-clicking on the variable in the Locals 43^{-1} window and selecting the **Set Data Breakpoint** context menu option. This means that in order to establish a data breakpoint you must first set a normal breakpoint 37^{-1} at a point in the code that presents the required scope of local variables to choose from.

Locals				🔷 🔶 Ф. (
able	Value	Туре	Address	
… ∳ m_bDefaultImage	1	int	0x0012ce1c	
🖽 🧄 m_sizeImage		CSize	0x0012ce20	
🖃 🧄 m_strLabel		ATL.CStringT <char< td=""><td>0x0012ce28</td><td>[</td></char<>	0x0012ce28	[
ATL::CSimpleStringT			0x0012ce28	
🖃 🧄 m_pszData	0x672b4e50	char*	0x0012ce28	
			0x672b4e50	
m_strDefF Break when	Variable is Modified	ATL.CStringT <char< td=""><td>0x0012ce2c</td><td></td></char<>	0x0012ce2c	
💷 🧅 m_strFileF View Memor	ry at Address	Break When Variable is N	odified ce30	
		Break when item modifie	d ce34	
Help				•
The sector is a sector of the	o woulds			

3.6.5 Debugging Actions

This section describes the actions you perform in running a debug session. It covers:

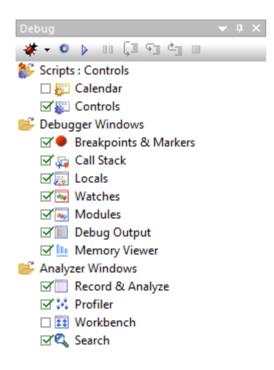
- Displaying Windows 39
- Starting and Stopping the Debugger 40
- Debugging a Subsequent Process 40
- <u>Stepping Over Lines of Code</u> 41
- Stepping Into Function Calls 41
- <u>Stepping Out of Functions</u> 42
- Viewing the Call Stack 42
- Viewing the Local Variables 43
- <u>Viewing the Content of Long Strings</u> 43
- Viewing Variables in Other Scopes 44
- Inspecting Process Memory 45
- Setting Breaks for When a Variable Changes Value 46
- Showing Loaded Modules 47
- <u>Showing Output from the Debugger</u> 47
- <u>Debugging Tooltips in Code Editors</u>. 48

3.6.5.1 Displaying Windows

Debugger Actions - Displaying Windows

The <u>Debugger windows</u> are available from the **View | Execution Analyzer** menu options.

These windows can also be displayed and hidden from the debug management control checkboxes shown below:



3.6.5.2 Start & Stop Debugger

Debugging Actions - Start & Stop

If <u>Basic Setup</u> ⁽⁸⁾ has been completed, pressing **[F6]** starts the application using the configured Debugger.

If not, debugging is still possible by using the Attach button on either one of the Debugger toolbars.

To stop debugging, click on the Stop button line the Debug window toolbar, or press [Ctrl]+[Alt]+[F6].

Notes:

In most situations, the debugger ends:

- · when it encounters breakpoints (which should be set beforehand)
- · when the debug process terminates or
- when the Java Class thread exits.

However, due to the nature of the Java Virtual Machine, it is necessary at times for Java developers to stop the debugger manually with the **Stop** button.

3.6.5.3 Debug Another Process

Debugging Actions - Debug Another Process

When debugging another process, the Attach To Process dialog is displayed.

You can limit the processes displayed using the radio buttons at the top of the dialog. To find a service such as Apache Tomcat or ASP.NET, select the **System** radio button.

You must choose the debugger when you select a process. However, if the selected Package has already been configured for debugging then the Debugger listed is the one specified in the Script.

Deb	ugger: Mi	icrosoft Native Show processes: User	🔘 System 🛛 🔘 All		
PID		Image	Session	User	Descriptio
	3564	C:\Windows\System32\taskeng.exe	1	SPARXSYSTEMS\smeagher	Task Sch
	3672	C:\Windows\System32\dwm.exe	1	SPARXSYSTEMS\smeagher	Desktop
-	3780	C:\Windows\explorer.exe	1	SPARXSYSTEMS\smeagher	Windows
	4000	C:\Program Files\Eset\nod32kui.exe	1	SPARXSYSTEMS\smeagher	NOD32 (
	4008	C: \Program Files \Windows Sidebar \sidebar.exe	1	SPARXSYSTEMS\smeagher	Windows
	4016	C: \Program Files \MWSnap \MWSnap.exe	1	SPARXSYSTEMS\smeagher	
	3184	C:\Program Files\HelpandManual4\HelpMan.exe	1	SPARXSYSTEMS\smeagher	Help Ma
	3724	C:\Program Files\HelpandManual4\HelpMan.exe	1	SPARXSYSTEMS\smeagher	Help Ma
	3788	C:\Program Files\OpenOffice.org 3\program\swriter.exe	1	SPARXSYSTEMS\smeagher	
	3236	C:\Program Files\Sparx Systems\EA\EA.exe	1	SPARXSYSTEMS\smeagher	Enterpris
	3352	C:\Program Files\WinMerge\WinMergeU.exe	1	SPARXSYSTEMS\smeagher	WinMerg
	3912	C:\Windows\System32\conime.exe	1	SPARXSYSTEMS\smeagher	Console
	3796	C:\Windows\System32\wuauclt.exe	1	SPARXSYSTEMS\smeagher	Windows
•	980	C:\Program Files\OpenOffice.org 3\program\soffice.exe	1	SPARXSYSTEMS\smeagher	OpenOf
	2544	C:\Program Files\OpenOffice.org 3\program\soffice.bin	1	SPARXSYSTEMS\smeagher	OpenOf
	2516	C:\Program Files\Sparx Systems\EA\SScripter.exe	1	SPARXSYSTEMS\smeagher	

Once Enterprise Architect is attached to the process, any breakpoints encountered are detected by the debugger and the information is available in the Debugger windows.

To detach from a process, click on the **Debug Stop** button.

3.6.5.4 Step Over Lines of Code

Debugging Actions - Step Over

You can only step over the lines of a function using the Debug toolbar buttons.

When you step to the end of the function, you step back to the caller.

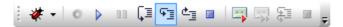


Alternatively, press [Alt]+[F6] or select the Project | Execution Analyzer | Step Over context menu option.

3.6.5.5 Step Into Function Calls

Debugging Actions - Step In

The Step In function is executed by clicking on the Step In button.



Alternatively, press [Shift]+[F6] or select the Project | Execution Analyzer | Step In context menu option.

If no source is available for the function then the debugger continues stepping till it either enters a new

function or reaches the next line of the current one.

3.6.5.6 Step Out of Functions

Debugging Actions - Step Out

The Step Out function is executed by clicking on the Step Out button.



Alternatively, press [Ctrl]+[F6] or select the Project | Execution Analyzer | Step Out context menu option.

If no source is available for the function then the debugger will continue stepping till it either enters a new function or reaches the

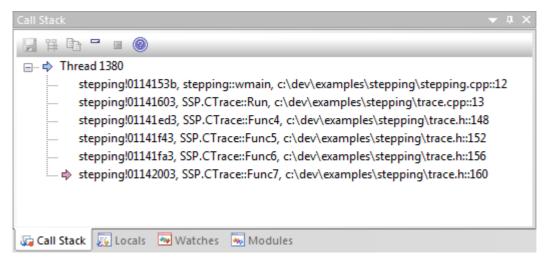
next line of the current one.

3.6.5.7 View the Call Stack

Debugging Actions - View the Call Stack

The Call Stack window shows all currently running threads. A Stack trace is displayed whenever a thread is suspended, through one of the step actions or through encountering a <u>breakpoint</u> 37.

- · A green or yellow arrow highlights the current stack frame
- A blue arrow indicates a thread that is running
- A red arrow indicates a thread for which a stack trace history is being recorded
- Double-clicking a frame takes you to that line of code in the Source Code Editor; local variables are also refreshed for the selected frame.



Toolbar

Save Stack to file
 Generate Sequence diagram from Stack
 Copy Stack to recording history
 Toggle Stack View

- Stop recording

43

3.6.5.8 View the Local Variables

Debugging Actions - Viewing Local Variables

Whenever a thread encounters a <u>breakpoint</u> $\overline{37}$, the Locals window displays all the local variables for the thread at its current <u>stack</u> $\overline{427}$ frame.

The value and the type of any in-scope variables are displayed in a tree, as illustrated below:

Variable	Value	Туре	Address	
🖃 🖕 Train	0x31e198	CTrain*	0x0384ff90	
🛓 🦕 CTrain			0x0031e198	
🖕 🧄 TObject			0x0031e198	
😟 🧄 Events	0x31e278	void**	0x0031e19c	
	8	int	0x0031e1a0	
🗊 🧄 Position		CPoint	0x0031e1a4	
🦾 🧄 Туре	TypeIsTrain	TObjectType	0x0031e1ac	
🕀 🧄 h_thread	0x16c	void*	0x0031e1b0	
	2608	unsigned long	0x0031e1b4	
🗈 🧅 Network	0x12f784	CNetwork*	0x0031e1b8	
🖶 🧄 Arriving	0x31bfc0	CStation*	0x0031e1bc	
🛶 Distance	0	float	0x0031e1c0	
🛶 Capacity	500	int	0x0031e1c8	
	92	int	0x0031e1cc	
🛶 Number	2	unsigned long	0x0031e1d0	
<u> </u>	<u> </u>		0.0004.4.14	Þ

Local variables are displayed with colored box icons with the following meanings:

- Blue Object with members
- Green Arrays
- Pink Elemental types
- Yellow Parameters
- Red Workbench Instance

3.6.5.9 View Content Of Long Strings

Debugging Actions - View Entire Content Of Long Strings

For efficiency, the Locals window only shows partial strings. The size of any variable value displayed in the window can be up to 256 characters.

To view the entire value of a variable, right-click on it and select the **View in Editor** context menu option. The String Viewer dialog displays.

Variable	Value	Туре	Address
🛁 🔶 argc	1	int	0x0020f8b8
🛓 🧄 argv	0x953118	wchar**	0x0020f8bc
•• x	0	int	0x0020f89c
	0x955d10	SSP::CTrace*	0x0020f8a8
GSP::CTrace			0x00955d10
i⊒ o m_rect		CRect	0x00955d14
🕀 🧄 members	0x958e30	int*	0x00955d24
i m_ptrs	0x95ff18	SSP::CPtr**	0x00955d28
🖨 💊 m_pBigString	0x9531b8, "ABCDEF	wchar*	0x00955d2c
L.	"ABCDEFGHIJKL		0.000521.68
🕀 💊 Formats		Break When Variabl	e is Modified
🗊 🖕 Strings		View Memory at Ad	dress
m_Color	Green	View in Editor	
		Сору	4
😼 Locals 🛛 💀 Watches 🛛 💀 Modules	📃 Debug 📃 Re 🍙	Help	ut



3.6.5.10 View Variables in Other Scopes

Debugging Actions - Viewing the Variables in Other Scopes

The Watches window is most useful for native code (C, C++, VB) where it can be used to evaluate data items that are not available as Local Variables 43 - data items with module or file scope and static Class member items.

You can also use the window to evaluate static Class member items in Java and .NET.

his	1 🙀 💿	
ariable	Value	Туре
🛯 🧅 lpszDefault	0x442a64, "Default"	char*
lpszCustom	0x442a6c, "Custom"	char*
⊪ 🖕 lpszNo	0x442a60, "No"	char*
indicators		unsigned int[4]

To use the Watches window, type the name of the variable to examine in the field in the window toolbar, and either press **[Enter]** or click on the **Add new watched item** icon.

To examine a static Class member variable in C++, Java or Microsoft .NET enter its fully qualified name. For example:

CMyClass::MyStaticVar

To examine a C++ data symbol with module or file scope, just enter its name. Note, items are evaluated only if the package in whose scope the item resides is currently loaded by the process being debugged. If the debugger is not running, no items are listed.

The names of the items to evaluate persist for the package and user ID, so the next time *you* debug the same project, the items evaluate automatically whenever a breakpoint occurs. They do not appear if another user debugs the same code.

If necessary, you can delete items using the **Delete all watched items** icon in the toolbar, or the right-click context menu options inside the window.

3.6.5.11 Inspect Process Memory

Debugging Actions - Inspecting Process Memory Debugging

You can display the raw values at a memory address or for a variable in a window using the Memory Viewer - select the View Memory at Address context menu option.

Variable	Value	Туре		Address	
🖨 🔶 [0]		wchar*[80)]	0x0087711c	
🖨 🔷 [0]	0x877e48, "String[0]	wchar*		0x0087711c	
· · · · •	"String[0][0]"			0v00877=48	
🕀 💊 [1]	0x878088, "String[0]	wchar*		Break When Variable is Mo	dified
🕀 🖕 [2]	0x8782c8, "String[0]	wchar*		View Memory at Address	
🛓 🔷 [3]	0x878508, "String[0]	wchar*		Сору	
🚊 💊 [4]	0x878748, "String[0]	wchar*			
🛓 🧄 🚺	0x878988, "String[0]	wchar*	0	Help	
🛓 🧄 🌖	0x878bc8, "String[0	wchar*		0x00877134	
⊞ 🍦 [7]	0x878e08, "String[0]	wchar*		0x00877138	

The Memory Viewer displays the raw values at a memory address

Memory Viewe	r	
0x00877E48	۹. 🕲	
0x00877E48	53 00 74 00 72 00 69 00 6E 00 67 00 5B 00 30 00 5D 00 5B 00 30	S.t.r.i.n.g.[
0x00877E5D	00 5D 00 00 00 CD	.]
0x00877E72	CD C	
0x00877E87	CD C	
0x00877E9C	CD C	
0x00877EB1	CD C	
0x00877EC6	CD C	
0x00877EDB	CD C	
0x00877EF0	CD C	
0x00877F05	CD C	
0x00877F1A	CD C	
0x00877F2F	CD C	
0x00877F44	CD C	
0x00877F59	CD C	
0x00877F6E	CD C	
0x00877F83	CD C	
0x00877F98	CD C	
Output	🗐 Output 🛛 🗐 Recording History 🔎 Breakpoints & Markers 🛛 🕰 Se	arch 🛄 Memory View

The Memory Viewer is available for debugging Microsoft Native Code Applications (C,C++,VB) running on Windows or within WINE on Linux.

3.6.5.12 Break When a Variable Changes Value

Debugging Actions - Break when a Variable Changes Value

An invalid or uninitialised variable can cause the program behaviour to differ from expected. This tool enables you to halt execution whenever a certain variable has its value changed.

Note:

This feature is not presently supported by the Microsoft .NET platform.

The example below creates a notification on a variable from the Watches window. The item being watched is an integer in the *SSP* namespace scope.

Watches			▼ ₽ ×
	1 🔜 😨		
Variable	Value	Туре	Address
SSP::divx	0	int	0x0114a1c0
	Break When Y View Memory Copy Delete Watch Delete All Wa	1	
🔹 🔤 Watches 🛛 🙀 Ca	I 🛞 Help		4

3.6.5.13 Show Loaded Modules

Debugging Actions - Show loaded modules

The debugger Modules window lists the modules loaded by the process being debugged.

Modules					
Path	Modified Date	Debug Sym	Symbol File Match	Symbol Path	Modified Date
ntdll.dll		Export, False	True		
C:\Windows\system32\kernel32.dll	21/01/2008 2:24	Export, False	True		
C:\Benchmark\Native\TwoDLLs\Console.exe	19/01/2009 5:49	PDB, True, Lines	True	c:\Benchmark	19/01/2009 5:49
C:\Benchmark\Native\TwoDLLs\Files.dll	19/01/2009 5:49	PDB, True, Lines	True	c:\Benchmark	19/01/2009 5:49

The columns on this window are described below:

Column	Use To
Path	Determine the file path of the loaded module.
Modified Date	Determine the local file date and time the module was modified.
Debug Symbols	Establish the debug symbols type, whether debug information is present in the module and whether line information is present for the module (required for debugging).
Symbol File Match	Check the validity of the symbol file; if the value is false, the symbol file is out of date.
Symbol Path	Determine the file path of the symbol file, which must be present for debugging to work.
Modified Date	Determine the local file date and time the symbol file was created.

3.6.5.14 Show Output from Debugger

Debugging Actions - Show Output From Debugger

During a debug session the Debugger emits messages detailing both startup and termination of session, to its Output tab. Details of exceptions and any errors are also output to this tab. Any trace messages such as those output using *Java System.out* or *.NET System.Diagnostics.Debug* are also captured and displayed here.

Output
🖵 💿
05/08/2009 09::49 Warning: no filters defined for package script; debugger will record every call
05/08/2009 09::49 Default Directory is Y:\Dev\Builds\Test Model Data\Debug & Profile\Microsoft .NET\
05/08/2009 09::49 Y:\Dev\Builds\Test Model Data\Debug & Profile\Microsoft .NET\Delegates\Delegate
05/08/2009 09::49 Debugger CLR runtime version is v2.0.50727
05/08/2009 09::49 Process created \Device\Mup\SPARXSYS02\SparxShare\Dev\Builds\Test Model Data
05/08/2009 09::49 Debug process in domain DefaultDomain
05/08/2009 09::49 Debug process in domain Delegates2.exe
05/08/2009 09::49 Hello, A!
05/08/2009 09::49 Invoking delegate b:
05/08/2009 09::49 Goodbye, B!

3.6.5.15 Debug Tooltips in Code Editors

Debugging Actions - Viewing Variable Values in Code Editors

During debugging, whenever a thread is suspended at a line of execution, you can inspect member variables in the Editor window.

To evaluate a member variable, use the mouse to move the cursor over the variable in the Editor window, as shown in the following examples.

```
public void Print()
{
    int n = 0;
    while(names[n].Length > 0)
    {
        names = {[4] names[0]=book, names[0]=book, names[1]=novel, names[2]=film}, ...}
        Document d = new Document(names[n++]);
        d.Print();
    }
}
```

```
public void Print()
{
    int n = 0;
    while32-bit signed integer n=0 0)
    {
        Document d = new Document(names[n++]);
        d.Print();
    }
}
```

3.6.6 Recording Actions

49

This section describes how to perform the following debug recording actions:

- Step through function calls 49
- Create a Sequence diagram of the Call Stack 49
- Save the Call Stack. 50

3.6.6.1 Step Through Function Calls

Debugging Actions - Step Through

The Step Through function can be executed by clicking on the **Step Through** button on the **Record & Analyze** window toolbar.



Alternatively, press [Shift]+[F6] or select the Project | Execution Analyzer | Step Into context menu option.

The *Step Through* command causes a *Step Into* command to be executed. If any function is detected, then that function call is recorded in the History window. The debugger then steps out, and the process can be repeated.

This button enables you to record a call without having to actually step into a function. The button is only enabled when at a breakpoint and in manual recording mode.

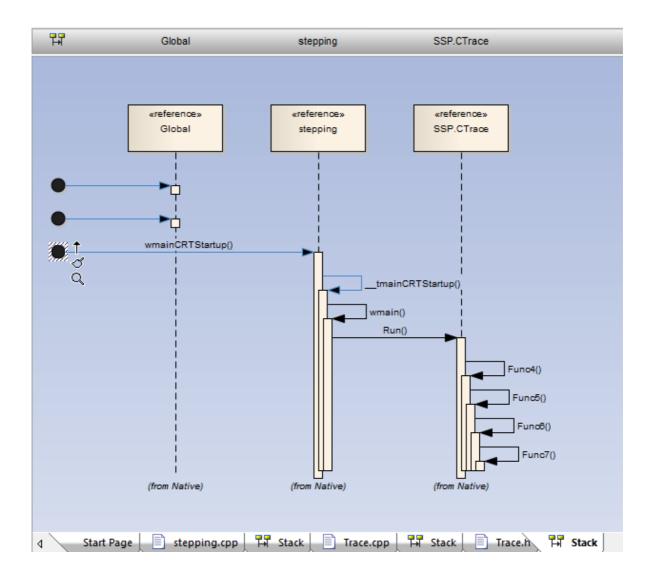
3.6.6.2 Create Sequence Diagram of Call Stack

Debugging Actions - Create Sequence Diagram from Current Call Stack

To generate a Sequence diagram from the current Stack, click on the **Generate Sequence Diagram of Stack** button on the Call Stack window toolbar.

Call Stack	
🔒 🋱 🗄	
□ • • •	Generate Sequence Diagram of Stack Generate Sequence Diagram of Stack
	stepping!00a92b4f, stepping::wmainCRTStartup stepping!00a92d08, stepping::tmainCRTStartup
	stepping:00a92008, stepping::tmainCk1startup stepping!00a9153b, stepping::wmain, c:\dev\examples\stepping\stepping.cpp::12 stepping!00a91603, SSP.CTrace::Run, c:\dev\examples\stepping\trace.cpp::13
	stepping!00a91ed3, SSP.CTrace::Func4, c:\dev\examples\stepping\trace.h::148 stepping!00a91f43, SSP.CTrace::Func5, c:\dev\examples\stepping\trace.h::152
	stepping!00a91fa3, SSP.CTrace::Func6, c:\dev\examples\stepping\trace.h::156 stepping!00a92008, SSP.CTrace::Func7, c:\dev\examples\stepping\trace.h::161

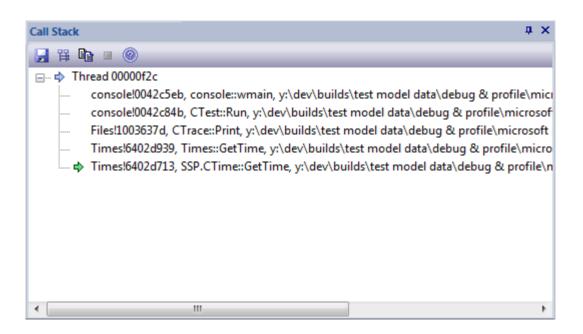
This immediately generates a Sequence diagram in the Diagram View.



3.6.6.3 Saving the Call Stack

Debugging Actions - Saving the Call Stack

On the Call Stack window, you can save the current Stack to file or copy the Stack to the recording history.



Toolbar

- Save Stack to file

- Copy Stack to recording history

3.7 Searching in Files

This topic describes how to use the File Search facility.

3.7.1 Search in Files

This topic describes the File Search control.

File Text Searches are provided by the Search Window and from within the Code Editors.

The Search window enables you to search for text in code files and scripts. You can select to display the results of the search in one of two formats:

• List View - each result line consists of the file path and line number, followed by the line text; multiple lines from one file are listed as separate entries

Search	# ×
C:\Benchmark 😅 🔹 Window 🔹 .cpp,.h,.c 🔹 🔍 Aa ax 🖁 🗮 💺 🖉 🔞	
; C:\Benchmark\DotNet\cpp\example_net_1\Form1.h::11, using namespace System::Windows::Forms;	
	E
C:\Benchmark\DotNet\cpp\example_net_1\Form1.h::20, private: System::Windows::Forms::CheckedListBox * ThreadList;	
C:\Benchmark\DotNet\cpp\example_net_1\Form1.h::167, private: System::Windows::Forms::Button * button1;	
C:\Benchmark\DotNet\cpp\example_net_1\Form1.h::170, private: System::Windows::Forms::Button * button2;	
C:\Benchmark\DotNet\cpp\example_net_1\Form1.h::171, private: System::Windows::Forms::TextBox * ThreadName;	
C:\Benchmark\DotNet\cpp\example_net_1\Form1.h::185, this->button1 = new System::Windows::Forms::Button();	
);
C:\Benchmark\DotNet\cpp\example_net_1\Form1.h::209, this-> button2-> DialogResult = System::Windows::Forms::DialogRe	sult::OK:
	indows headers
C:\Benchmark\DotNet\csharp\Loader\Loader\Form1.h::9, using namespace System::Windows::Forms;	
C:\Benchmark\DotNet\csharp\Loader\Loader\Form1.h::48, private: System::Windows::Forms::Label^ label1;	
C:\Benchmark\DotNet\csharp\Loader\Loader\Form1.h::49, private: System::Windows::Forms::Panel^ panel2;	
	Þ.

• *Tree View* - each result line consists of the file path that matches the search criteria, and the number of lines matching the search text within that file; you can expand the entry to show the line number and text of each line.

Search				ą ×
C:\Benchmark	💕 🔹 Window	 .cpp,.h,.c 	• 🔍 Aa a* 😫 🖺 🍹 🔌 🤇	0
	k\DotNet\cpp\example_net_1	\Form1.h (11)		*
🚊 C:\Benchmark	k\DotNet\cpp\example_net_1	\stdafx.h (1)		
7, #defir	ne WIN32_LEAN_AND_MEAN	// Exclude rarely-used stuff fro	m Windows headers	
🖶 C:\Benchmark	k\DotNet\csharp\Loader\Loa	der\Form1.h (19)		
🛓 C:\Benchmark	k\DotNet\csharp\Loader\Loa	der\Loader.cpp (1)		
🖶 C:\Benchmark	k\DotNet\csharp\Threading\	Threading\Form1.h (66)		
🕀 C:\Benchmark	k\DotNet\csharp\Threading\	Threading\Threading.cpp (1)		
E C:\Benchmark	k\Interop\Native\stdafx.h (8)			
🗄 C:\Benchmark	k\Interop\Native_Native_p.c	(1)		E
E C:\Benchmark	k\Native\Benchmark\Bench.c	:pp (1)		
E C:\Benchmark	k\Native\Benchmark\BenchD	lg.cpp (32)		
E C:\Benchmark	k\Native\Benchmark\ProfileD)lg.cpp (20)		
E C:\Benchmark	k\Native\Benchmark\stdafx.h	(12)		
E C:\Benchmark	k\Native\Benchmark\ToLowe	erDlg.cpp (10)		
E C:\Benchmark	k\Native\C Console\C-Code\	stdafx.h (2)		
E C:\Benchmark	k\Native\Example\Example.cp	op (1)		
E C:\Benchmark	k\Native\Example\stdafx.h (12	2)		
🛓 C:\Benchmark	k\Native\NativeClient\Proxy.ł	h (1)		
C:\Benchmark	k\Native\Scripter\Scripter\Scr	ripter.cpp (11)		
🛓 C:\Benchmark	k\Native\Scripter\Scripter\std	lafx.h (8)		*

Search Toolbar

You can use the toolbar options in the Search window to control the search operation. The state of all buttons persists over time to always reflect your previous search criteria.

C:\Benchmark	🗃 🔹 Window	 .cpp,.h,.c 	- 🔍 Aa	a* 🗀 📕 🍢 🔌 🎯 👘
--------------	------------	--------------------------------	--------	----------------

The options, from left to right, are as follows:

Option	Use to
Search Path list box	Specify the folder to search.

Option	Use to
	You can type the path to search directly into the text box, or click on the folder icon to browse for the path. Any paths you enter are automatically saved in the drop-down list, up to a maximum of ten; paths added after that overwrite the oldest path in the list.
	A fixed option in the drop-down list is Search in Scripts , which sets the search to operate on all local and user-defined scripts in the Scripts tab of the Scripter window. This option disables the Search File Types list box.
Search Text list box	Specify the text to look for.
	You can type the text directly into the text box or click on the drop-down arrow to select from a previous entry in the list. The search text you enter is automatically saved in the list when you click on the Search button.
	The list box saves up to ten search queries. Search queries added after that overwrite the oldest query in the list.
Search File Types list box	Limit the search to specific types of files. You can select multiple file types in a string, separated by either a comma or a semi-colon as shown in the image above.
Search button	Begin the search.
	During the course of the search all other buttons in the toolbar are disabled. You can cancel the search at any time by clicking on the Search button again.
	If you switch any of the toggle buttons below, you must run the search again to change the output.
Case Sensitivity button	Toggle the case sensitivity of the search. The tooltip message identifies the current status of the button.
Word Match button	Toggle between searching for any match and searching for only those matches that form an entire word. The tooltip message identifies the current status of the button.
SubFolders button	Toggle between limiting the search to a single path and including all subfolders under that path. The tooltip message identifies the current status of the button.
Result View button	Select the presentation format of the search results - List View or Tree View format.
Clear Results button	Clear the results.
Clear Search Criteria button	Remove all the entries in the Search Path , Search Text and Search File Types list boxes, if required.
Help button	Display this Help topic.

3.8 Testing Command

53

This section describes how to create a <u>command for performing unit testing</u> 53 on your code.

3.8.1 Add Testing Command

This topic explains how you enter a command for performing unit testing on your code.

The command is entered in the text box using the standard *Windows Command Line* commands. A sample script would contain a line to execute the testing tool of your choice, with the filename of the executable produced by the **Build** command as the option. To execute this test select the **Project | Execution Analyzer | Test** menu option.

Testing could be integrated with any test tool using the command line provided, but in these examples you can see how to integrate *NUnit* and *JUnit* testing with your source code. Enterprise Architect provides an inbuilt

MDA Transform from source to Test Case, plus the ability to capture *xUnit* output and use it to go directly to a test failure. xUnit integration with your model is now a powerful means of delivering solid and well-tested code as part of the complete model-build-test-execute-deploy life-cycle.

Note:

NUnit and JUnit must be downloaded and installed prior to their use. Enterprise Architect does not include these products in the base installer.

The **Capture Output** checkbox enables Enterprise Architect to show the output of the program in the **Output** window, while the **Output Parser** field specifies what format output is expected. When parsing is enabled, double-clicking on a result in the **Output** window opens the corresponding code segment in Enterprise Architect's code window.

Selecting the Build before Test checkbox ensures that the package is recompiled each time you run the test.

Two example test scripts are included below. The first is an NUnit example that shows the **Build before Test** checkbox selected. As a result, every time the test command is given it runs the build script first.

Enter your test script below and select the appropriate output parser for the type of testing required						
	Enter your test script below and select the appropriate output parser for the type of testing required					
"C: \Program Files \Nunit\bin \nunit-console.exe" bin \debug \customer.exe						
·						
Capture Output 📝 Build before Test Output Parser: NUnit						

Note:

The command listed in this field is executed as if from the command prompt. As a result, if the executable path or any arguments contain spaces, they must be surrounded in quotes.

The second example is for JUnit. It doesn't have the **Build before Test** checkbox selected, so the build script won't be executed before every test, but as a result it could test out of date code. This also shows the use of % *N*, which is replaced by the fully namespace-qualified name of the currently selected Class when the script is executed.

Build	Test	Run	Debug	Deploy	Sequence Diagram Recording								
Enter	Enter your test script below and select the appropriate output parser for the type of testing required												
java	java junit. texttui. Testrunner %N												
					-								
📃 Ca	pture Out	put 📄	Build bef	ore Test	Output Parser: JUnit 🔻]							

3.9 Run Command

This section describes how to create a <u>command for running</u> 55th your executable code.

3.9.1 Add Run Command

This topic explains how you enter a command for running your executable.

This is the command that is executed when you select the **Project | Execution Analyzer | Run** menu option. At its simplest, the script would contain the location and name of the file to be run.

Note:

Enterprise Architect provides the ability to start your application normally OR with debugging from the same script. The **Execution Analyzer** menu has separate options for starting a normal run and a debug run.

The following two examples show scripts configured to run a .Net and a Java application in Enterprise Architect.

Build	Test	Run	Debug	Deploy	Sequence Diagram Recording								
Enter t	Enter the path to the compiled application												
C:\be	enchmark	:\cpp\exa	mple_net	_1\releas	e\example.exe								
	-					1							

build Test	1. Call	Debug Deploy Sequence Dia	gram Recording
Enter the path	n to the c	compiled application	
customer			

Note:

The command listed in this field is executed as if from the command prompt. As a result, if the executable path or any arguments contain spaces, they must be surrounded in quotes.

3.10 Deploy Command

This section describes how to create a <u>command for deploying</u> 55 the current package.

3.10.1 Add Deploy Command

This topic enables you to create a command for deploying the current package.

These are the commands that are executed when you select the **Project | Execution Analyzer | Deploy** menu option.

Write your script in the large text box using the standard Windows Command Line commands.

Recording											
Enter your script below for deploying the current package											
deploy.bat											

4 Execution Analysis



This section describes the Visual Analysis of executing applications by recording application execution and generating:

- Sequence Diagrams
- Sequence/State Diagrams
- Profile (execution) Reports

Execution analysis is configured by creating a <u>debug script</u> for the packages to be tested. One of the primary objectives of this feature is to enable you to perform a debug walk-through executing code, and capture your stack trace for direct conversion into a Sequence diagram. This is a great way to document and understand what your program is doing during its execution phase.

Execution Analysis debugging and recording are supported for the following platforms / languages:

- Microsoft Windows Native C
- Microsoft Windows Native C++
- Microsoft Windows Visual Basic
- Microsoft .NET Family (C#, J#, VB)
- Sun Microsystems Java.

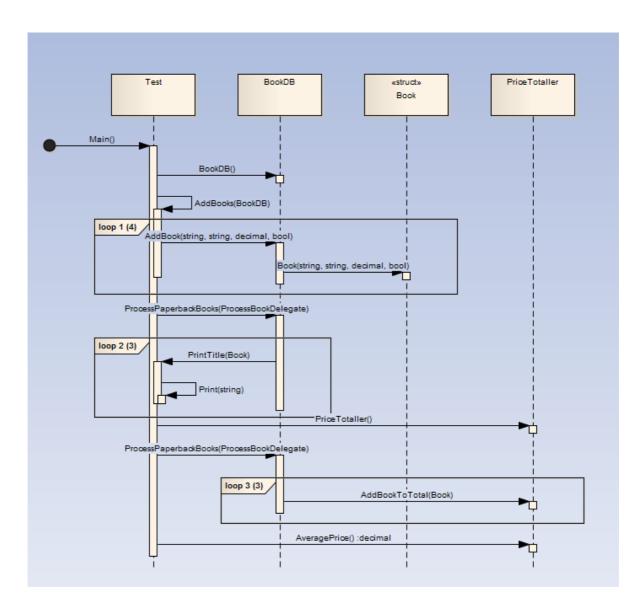
4.1 Recording Sequence Diagrams

This section explains how to use the Visual Execution Analyzer to record execution data in the form of a Sequence Diagram. It covers:

- An overview of how the process works 57
- <u>Setup for recording</u> 59
- Placing recording markers 66
- Controlling the recording session 72
- <u>Generating Sequence diagrams</u> 74
- Adding State Transitions. 75

4.1.1 How it Works

The Visual Execution Analyzer enables you to generate a Sequence Diagram. The diagram below illustrates the output of a Sequence Diagram for a program that calculates the price of books. The diagram creates a visual representation of the execution of an application, outlining what functions are being called, types of messages being sent, key data structure used and the relationships between different classes. The diagram makes it much simpler to Understand how information is moved throughout the system and what values are being passed by various functions. The first loop structure is executed four times and is being used to add four books to the book database. The arrows indicate information flow and demonstrate the change of states over time.



A Sequence diagram provides easy to understand visual information including:

- An understanding of how information is passed throughout a system.
- The sequence of various functions and their corresponding parameters.
- A clear understanding of how different classes interact to create behavior.
- A visual overview of how data structures are used to produce results.

If an application crashes, data corruption such as a stack overflow can prevent you from diagnosing and rectifying the problem. However the Visual Execution Analyzer allows you to record a given execution sequence and provide a reliable source of information that may further explain why a crash occurred. Enterprise Architect can record arguments to functions, record calls to external modules or capture state transitions based on any given constraint. This information can be integrated with existing system knowledge and test data to optimize code execution, reduce errors and understand why application failure and system crashes occur.

A Sequence Diagram extends traditional analysis to help identify errors in logic, explain unexpected system behavior and identify data flow inconsistencies. The Visual Execution Analyzer extends analysis through the use of a comprehensive array of reports that detail everything from state transitions through to the contents of the stack at a given time. A Sequence Diagram can convey more detail and provide greater understanding than reading unfamiliar code that has potentially been written by someone else. It also makes it easier to document existing code when a Sequence Diagram illustrates functions are being called and the specific sequence of events that occur to produce a particular type of system behavior.

4.1.2 Setup for Recording

59

This section explains how you prepare to record execution of the application. It covers:

- Prerequisites 59
- <u>Configuring Recording Detail</u>
- Advanced Techniques 64

4.1.2.1 Pre-Requisites

Recording is available to users of Enterprise Architect Professional and above.

Basic setup 8 must be completed.

You should first be able to successfully debug 15th the application.

4.1.2.2 Configure Recording Detail

The Sequence Diagram Recording tab enables you to set various options for generating Sequence diagrams from the debugger.

Build	Test	Run	Debug	Deploy	Sequence Diagram R	ecording	
Optio	ns					Filters	ž 🗙
🔲 En	able Filte	r					
🔽 Re	cord arg	uments to	o functior	n calls			
Re	cord calls	to exter	mal modu	iles			
🗸 Re	cord calls	to dyna	mic modu	les			
	pture sta any>	ite transi	tions usin	ng constra	r I		
		ecording		frame thre	eshold		

These options are not all available for each platform, as indicated in the following table:

Option	.NET	Java	Native
Enable Filter	х	х	Х
Record arguments to function calls	х	х	Х
Record calls to external modules	Х	х	Х
Record calls to dynamic modules 62	х	-	-
Capture state transitions using constraint 75	х	х	х
Limit auto recording to stack frame threshold 63	х	х	Х
Enable diagnostic messages	х	х	х

4.1.2.2.1 Enable Filter

If the **Enable Filter** option is selected on the Sequence Diagram Recording tab, the debugger excludes calls to matching methods from the generated sequence history and diagram. The comparison is case-sensitive.

To add a value, click on the **New** (Insert) icon in the right corner of the **Filters** box, and type in the comparison string. Each filter string takes the form:

class_name_token::method_name_token

The *class_name_token* excludes calls to all methods of a Class or Classes having a name that matches the token. The string can contain the wildcard character * (asterisk). The token is optional.

The *method_name_token* excludes calls to methods having a name that matches token. The string can contain the wildcard character *. The token is optional.

Where no Class token is present, the filter is applied only to global or public functions; that is, methods not belonging to any Class.

To Filter	Use Filter Entry
All public functions having a name beginning with Get from the recording session (<i>GetClientRect</i> for example in Windows API).	::Get*
All methods beginning with Get for every Class member method.	*::Get*
All methods beginning with Get from the Class <i>CClass</i> .	CClass::Get*
All methods for Class <i>CClass</i> .	CClass::*
All methods for Classes belonging to Standard Template and Active Template Libraries.	• ATL* • std*
The specific method GetName for Class CClass.	CClass::GetName

In the Java example in the screen below, the debugger would exclude:

- Calls to OnDraw method for Class Example.common.draw.DrawPane
- Calls to any method of any Class having a name beginning with Example.source.Collection
- Calls to any constructor for any Class (ie: *<clint>* and *<init>*).

В	uild	Test	Run	Debug	Deploy	Sequence Diagram	Recor	rding						
ſ	Optio	ns						Filters					쐰	$\boldsymbol{\times}$
	V Ena	able Filte	r					Examp	le.common	n.draw.D	rawPane	::OnDraw		
								Examp	le.source.	Collection	י*ו			
	Rec	cord arg	uments	to functio	n calls			*::init	*					
	_			ernal mod u amic modu										

In the Native code example below, the debugger would exclude:

- Calls made to Standard Template Library namespace
- Calls to any Class beginning with TOb
- Calls to any method of Class *CLock*

Execution Analysis | Recording Sequence Diagrams

- Calls to any Global or Public Function with a name beginning with Get
- Calls to the method GetLocation for Class Ctrain.

61

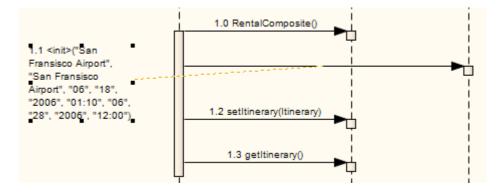
Build Test Run Debug Deploy Sequence Diagram Re	cording					
Options	Filters 🖄 🗙					
C Enable Filter	std* TOb*					
Record arguments to function calls	CLock					
Record calls to external modules	CTrain::GetLocation ::Get*					
Record calls to dynamic modules						
Capture state transitions using constraint						

4.1.2.2.2 Record Arguments To Function Calls

When recording the sequence history, Enterprise Architect can record the arguments passed to method calls.

Build	Test	Run	Debug	Deploy	Sequence Diagram	Reco	ording
Optio	ns						Filters
🔳 Ena	able Filte	r					
🔽 Re	cord argu	uments ti	o functior	n calls			

When the **Record Arguments to function calls** option is selected on the **Build Script** dialog **Sequence Diagram Recording** tab, the resulting Sequence diagram shows the values of elemental and string types passed to the method. See the following Java example.



Where the argument is not an elemental type, the type name is recorded instead.

4.1.2.2.3 Record Calls To External Modules

On the Sequence Diagram Recording tab, the **Record calls to external modules** option causes function calls to external modules outside the model to be included in the sequence history and generated diagram.

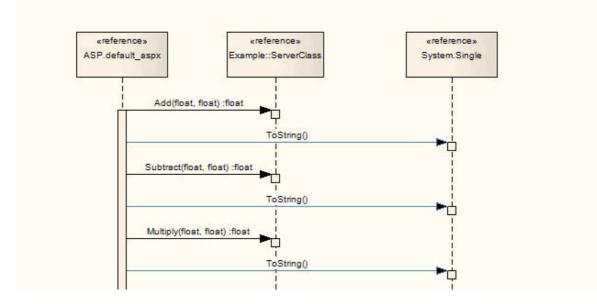
For applications built in a Microsoft Native code (C, C++) you can record calls to the WIN32 API if required, using the **Record calls to external modules** option. This option can also be used to record calls to functions in modules that have a PDB file but for which there is no source.

Build	Test	Run	Debug	Deploy	Sequence Diagram	n Recording
Optio	Ins					Filters
🔳 En	able Filte	r				
Re	cord arg	uments	to functior	n calls		
📝 Re	cord calls	to exte	ernal modu	ules		
Re	cord calls	to dyn	amic modu	ıles		

Only calls originating within the model to functions external to the model are recorded.

Note:

External calls are displayed with a blue connector, as shown below.



This example shows three external calls (*ToString()*) to the Microsoft .NET framework assembly function *System*.*Single*.

4.1.2.2.4 Record Calls to Dynamic Modules

(Available only for .NET platforms.)

On the Sequence Diagram Recording tab, the **Record calls to dynamic modules** option causes the debugger to record execution of dynamic or 'In Memory' function calls, in transitions between normal assemblies and those emitted dynamically.

Build	Test	Run	Debug	Deploy	Sequence Diagram	Recording						
Optio	Filters											
🔲 En	Enable Filter											
🔽 Re	cord argu	uments t	o functior	n calls								
Re	Record calls to external modules											
🔽 Re	cord calls	s to dyna	mic modu	les								

4.1.2.2.5 Limit Auto Recording

Where the Stack window shows recording to be involved in function calls that are not particularly useful, and that are not being excluded in a filter, you can achieve a quicker and more general picture of a sequence by limiting the stack depth being recorded. You can do this on the Sequence Diagram Recording tab, by selecting the Limit auto recording to stack frame threshold: option.

If you use this option, be aware that the threshold value you set is a relative frame count; that is, the count is relative to the frame at which recording begins. For example:

A breakpoint has occurred, and the Stack window shows five frames. If the stack frame threshold is set to 3 and you begin auto-recording at this breakpoint, the debugger records all function calls between the current frame 5 and a maximum stack frame depth of 8 inclusive.

For situations during auto-recording where the stack is very large, it is recommended that you first use a low stack frame threshold of 2 or 3, gradually increasing it if necessary to expand the picture. You can also use the threshold to work out which filters you could add to the script in order to further clarify the Sequence diagram that is ultimately produced.

Build Script							
Name: Directory:	Calendar C:\Program Files\BCGSoft\BCGControlBarPro\Examples\BCGPCalendarDem				BCGPCalendarDemo		
Build Te	est Ru	un	Debug	Deploy	Sequence Diagram	Reco	ording
Options						Filters	
Enable Filter							
Record arguments to function calls							
Record calls to external modules							
Record calls to dynamic modules							
Capture state transitions using constraint							
<any></any>							
 Limit auto recording to stack frame threshold 4 Enable diagnostic messages 							

4.1.2.2.6 Enable Diagnostic Messages

The **Enable diagnostic messages** checkbox triggers the debugger to output more self-reporting, diagnostic messages as it executes. For example, the debugger might output messages about method calls that are being excluded from the recording history due to a filter also having been set in the Sequence Diagram Recording tab of the Build Script dialog.

Build	Test	Run	Debug	Deploy	Sequence Diagram Rec	ording
Optio	Ins					Filters
Enable Filter						
Record arguments to function calls						
Record calls to external modules						
Record calls to dynamic modules						
Capture state transitions using constraint						
<any></any>						
Limit auto recording to stack frame threshold						
Enable diagnostic messages						

4.1.2.3 Advanced Techniques

This section describes the advanced techniques for configuring recording detail:

- <u>Recording Activity for a Class</u>
 ⁶⁴
- Recording Activity for a single method 65

4.1.2.3.1 Recording Activity for a Class

In addition to setting breakpoints and markers in the code editor, you record all the operations of a class, or a subset by using the *Class Markup* Feature.

This feature is available from the Project Browser context menu while on a Class. Select the operations to record, choose the marker type and enter a name for the set. When you click on the **OK** button the markers are stored as a marker set using the name you specify.

This set can then be loaded either before or during a session.

The marker type specifies the action to take when the process encounters that marker.

- Record function
- Record Stack Trace
- Break execution

You can also specify a recording depth. This limits the recording, which if uncontrolled can ultimately produce Sequence Diagrams that are too complicated to read. When you specify a depth, the Debugger does not record beyond this depth.

The depth is relative to the stack depth where the Debugger first encountered the recording marker. So, if the stack depth is 7 when recording begins, and the Limit Depth is set to 3, the Debugger does not record beyond a Stack depth of 10.

Class Markup Selection					
 Existing marker set: Name: Calendar Tool§ar ✓ CCalendarBar ✓					
Include disabled operations					
Marker Type:		Func 🔻			
Limit recording fr	ame depth:	3			
				ОК	Cancel

4.1.2.3.2 Recording Activity for a Single Method

A <u>Method Auto Record</u> marker enables you to record activity for a particular function during a debug session. The debugger records any function calls executed after the marker point, and always stops recording when this function exits. The function marker combines a Start Recording marker and an End Recording marker in one.

```
185
186
   // CRecurrenceDlg message handlers
187
188 BOOL CRecurrenceDlg::OnInitDialog()
189 🖂 {
    CBCGPDialog::OnInitDialog();
190
191
192
       UINT nMask =
          CBCGPDateTimeCtrl::DTM_SPIN
193
           CBCGPDateTimeCtrl::DTM DATE
194
                                          I
195
           CBCGPDateTimeCtrl::DTM TIME
196
           CBCGPDateTimeCtrl::DTM CHECKBOX
                                          1
197
           CBCGPDateTimeCtrl::DTM DROPCALENDAR |
           CBCGPDateTimeCtrl::DTM CHECKED;
198
199
200
       UINT nFlags = CBCGPDateTimeCtrl::DTM CHECKED | CBCGPDateTimeCtrl::DT
201
       //-----
202
       // Setup date fields:
```

4.1.3 Place Recording Markers

This section explains how to deploy recording markers:

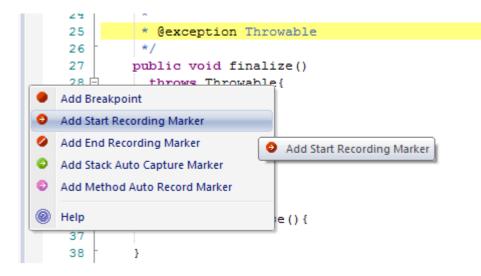
- Marker types 66
- Setting Recording Markers 70
- The Breakpoint and Markers window 71
- Activate and Disable Markers 71
- Working with Marker Sets 72
- Differences between breakpoints and markers. 72

4.1.3.1 Marker Types

Trace marking is a feature that enables you to silently record code executed between two points, and incorporate it in a Sequence diagram. The feature also enables you to capture the execution of multiple threads. It can be particularly useful in capturing event driven sequences (such as mouse and timer events) without any user intervention.

The recording markers are breakpoints; however, instead of stopping, the debugger behaves according to the type of marker. If the marker is denoted as a recording *start point*, the debugger immediately begins to trace all executed calls from that point for the breaking thread. Recording is stopped again when either the thread that is being captured terminates or the thread encounters a *recording end point*.

Recording markers are set in the source code editor. If you right-click on the breakpoint margin at the point to begin recording, a context menu displays:



Select the **Add Start Recording Marker** option, then right-click on the breakpoint margin at the point to stop recording and select the **Add End Recording Marker** context menu option. The markers are shown below:

0	17 18	<pre>private int m_delivery;</pre>
	19 🛱	<pre>public ClassLib() {</pre>
	20	
	21	}
	22	
	23 白	/**
	24	*
	25	* @exception Throwable
	26	*/
	27	<pre>public void finalize()</pre>

When the debugger is run it continues to run the thread, recording a stack history, until either the **End Recording** marker is encountered or the thread terminates, unlike normal breakpoints where the debugger halts and displays the line of code.

It is useful to <u>limit the stack depth</u> as when recording particularly high-level points in an application, as the stack frame count can result in too much information being collected. You can limit stack depth using the <u>Sequence Diagram Recording</u> tab on the <u>Build Script</u> dialog.

Build Script	State of the second	
Name:	Calendar	
Directory:	C:\Program Files\BCGSoft\BCGControlBarPro\Examples	\BCGPCalendarDemo
Build Te	st Run Debug Deploy Sequence Diagram Re	cording
Options		Filters
Enable	Filter	
Record	d arguments to function calls	
Record	d calls to external modules	
Record	d calls to dynamic modules	
	re state transitions using constraint	
<any< td=""><td>></td><td></td></any<>	>	
	uto recording to stack frame threshold	
4	diagnostic messages	

Running this Calendar example with the one function record marker in *CRecurrenceDlg::OnInitDialog()* produced the following output in the Recording History window:

Sequence	Insta	Method	Direction	Method
O000001 O			Call	CRecurrenceDlg.OnInitDialog
☑ 0000002		CRecurrenceDlg.OnInitDialog	Call	CBCGPDialog.On Init Dialog
✓ 0000003		CBCGPDialog.On Init Dialog	Call	CBCGPDialog.lsVisualManagerStyle
☑ 0000004			Return	CBCGPDialog.On Init Dialog
☑ 0000005		CBCGPDialog.On Init Dialog	Call	CBCGPDialog.lsVisualManagerNCArea
0000006			Return	CBCGPDialog.On Init Dialog
☑ 0000007		CBCGPDialog.On Init Dialog	Call	CBCGPDIgImpl.EnableVisualManagerStyle
☑ 0000008		CBCGPDlgImpl.EnableVisualManagerStyle	Call	CBCGPButton.GetThisClass
✓ 00000009			Return	CBCGPDIgImpl.EnableVisualManagerStyle
☑ 00000010		CBCGPDlgImpl.EnableVisualManagerStyle	Call	ATL.operator==
00000011			Return	CBCGPDIgImpl.EnableVisualManagerStyle
00000012		CBCGPDlgImpl.EnableVisualManagerStyle	Call	ATL.operator==
00000013			Return	CBCGPDlgImpl.EnableVisualManagerStyle
00000014		CBCGPDlgImpl.EnableVisualManagerStyle	Call	CBCGPGroup.CBCGPGroup
0000015			Potum	CBCGBDialmal EnableMaualManagerShile

Stack Auto-Capture Marker

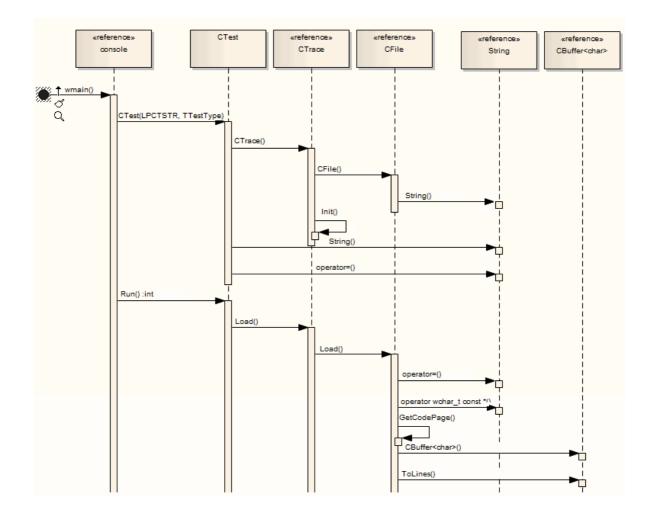
76	/* End - EA generated code for Parts and Ports */
77	/* Begin - EA generated code for Activities and II
78	<pre>public void ClassLib_ActivityGraphWithActionPin()</pre>
79 ¢	3

(Native Code only.) Stack markers enable you to capture any unique stack traces that occur at a point in an application. To insert a marker at the required point in code, right-click on the line and select the Add Stack Auto Capture Marker context menu option.

Each time the debugger encounters the marker it performs a stack trace. If the stack trace is not in the recording history, it is copied. The application then continues running. Stack markers provide a quick and

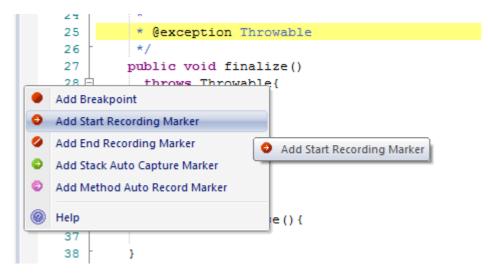
Record & Analyze					→ ↓ >
🕨 🔍 🛱 💱 🛙	• 🗸 🖻 I	异 璀 ⑧			
Sequence	Instance	Method	Direction	Method	In 🔺
⊡ 🗹 0000001		steppingtmainCRTStartup	Call	stepping.wmain	
0000002		stepping.wmain	Call	SSP.CTrace.Run	
✓ 00000003		SSP.CTrace.Run	Call	SSP.CTrace.Get	
✓ 00000004		SSP.CTrace.Get	Call	SSP.CTrace.SetRect	=
✓ 00000005			Return	SSP.CTrace.Get	
0000006			Return	SSP.CTrace.Run	
0000007		SSP.CTrace.Run	Call	SSP.CTrace.Func3	
☑ 0000008		SSP.CTrace.Func3	Call	SSP.CTrace.SetRect	
✓ 00000009			Return	SSP.CTrace.Func3	
00000010			Return	SSP.CTrace.Run	
☑ 00000011		SSP.CTrace.Run	Call	SSP.CTrace.Func4	
00000012		SSP.CTrace.Func4	Call	SSP.CTrace.Func5	
✓ 00000013		SSP.CTrace.Func5	Call	SSP.CTrace.Func6	
✓ 00000014		SSP.CTrace.Func6	Call	SSP.CTrace.Func7	
00000015			Return	SSP.CTrace.Func6	-
•	III				•
Breakpoints &	Memory \	/iewer 📄 Debug Output 🕴	🔍 Search 🛛	Output 📃 Record & An	alyze

useful picture of where a point in an application is being called from.



4.1.3.2 Setting Recording Markers

Recording markers are set in the source code editor. If you right-click on the breakpoint margin at the point to begin recording, a context menu displays:



Select the Add Start Recording Marker option, then right-click on the breakpoint margin at the point to stop recording and select the Add End Recording Marker context menu option. The markers are shown below:

```
17
         private int m delivery;
18
19 白
         public ClassLib() {
20
21
         }
22
23 白
24
            @exception Throwable
25
26
          */
27
         public void finalize()
```

When the debugger is run it continues to run the thread, recording a stack history, until either the **End Recording** marker is encountered or the thread terminates, unlike normal breakpoints where the debugger halts and displays the line of code.

4.1.3.3 The Breakpoints and Markers Window

The Breakpoints and Markers window allows you to manage control of the process. Here you can enable, disable, delete markers and also manage them as sets. You can organize how they are displayed, either in list view or grouped by file or class.

 ▼ = + @
Details
loy\ClassLib.java::15 loy\ClassLib.java::36
loy\ClassLib.java::48

4.1.3.4 Activate and Disable Markers

To delete a specific breakpoint, either:

- If the breakpoint is enabled, click on the red breakpoint circle in the left margin of the Source Code Editor
- Right-click on the breakpoint marker in the editor and select the appropriate context menu option, or
- Select the breakpoint in the Breakpoints & Markers tab and press [Delete].

Whether you are viewing the *Breakpoints* folder or the <u>Breakpoints</u> & <u>Markers</u> window, you can right-click on an existing breakpoint and select a context menu option either to delete it or to convert it to a <u>start recording</u> <u>marker or end recording marker</u> 66.

You can also delete all breakpoints by clicking on the Delete all breakpoints button on the Breakpoints &

Markers window toolbar (🕮).

To disable a breakpoint, deselect its checkbox on the Breakpoints & Markers window or, to disable all

breakpoints, click on the **Disable all breakpoints** button in the toolbar (¹). The breakpoint is then shown as an empty grey circle. Select the checkbox or use the **Enable all breakpoints** button to enable it again (

4.1.3.5 Working with Marker Sets

Marker sets enable you to group markers into collections.

A set can be used to record a specific Use Case, which might involve the operations of various Classes. Once a set is created it is saved with the Model. Any other user using the Model has access to that set.

Sets are normally loaded prior to the point at which an action is to be captured. For example, to record a sequence involving a particular dialog, you might set markers for the areas to record, saving the markers as a set. When you begin debugging, prior to invoking the dialog you would then load the set. Once you bring up the dialog in the application, the operations you have marked are recorded. Review the recording history and create a Sequence diagram.

4.1.3.6 Differences to Breakpoints

Breakpoints differ from Markers in that they always break execution whereas Markers operate silently without intervention.

4.1.4 Control the Recording Session

This section describes how you control the recording session:

- Auto Recording 72
- Manual Recording 73
- Pause Recording 73
- Resume Recording 73
- Stop Capture. 73

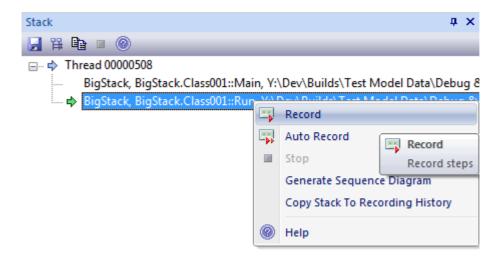
4.1.4.1 Auto-Recording

Auto-Recording is available when the process being debugged is at a breakpoint.

You can use the record button on the Record & Analyze window toolbar.

🕨 🔍 📑	-		<	É.	2	爭	0
-------	----------	--	---	----	---	---	---

Alternatively, select the thread in the stack window:



4.1.4.2 Manual Recording

Manual Recording is available when the process being debugged is at a breakpoint. Display the Stack window and use the context menu to switch to record mode.

Stack			ąχ
🛃 🋱 🖹 🗉 🎯			
🖃 🔿 Thread 00000508			
BigStack, BigStack.Class001::Mai	n, Y:\	Dev\Builds\Test M	1odel Data\Debug 8
📥 🖶 BigStack, BigStack.Class001::Run		NEAD, STELLAND AND AND AND AND AND AND AND AND AND	- J-I D-1-) D-L 9.
		Record	
		Auto Record	Record
		Stop	-
			Record steps
		Generate Sequence	e Diagram
		Copy Stack To Rec	ording History
	0	Help	

Thereafter you must issue debug commands {StepIn, StepOver, StepOut, Stop} manually.

Each time you issue a step command and the thread stack changes, the sequence of execution is logged.

When you have finished tracing, click on the Stop button (\blacksquare).

4.1.4.3 Pause Recording

You can pause recording by using the **Pause/Resume Execution** button on the **Debug** window toolbar or in the **Debug Management** window ([Alt]+[8]).

4.1.4.4 Resume Recording

You can resume recording using the **Pause/Resume Execution** button on the **Debug** window toolbar or in the **Debug Management** window (**[Alt]+[8]**).

4.1.4.5 Stop Capture

To stop recording at any time click on the Stop Recording button on the Record & Analyze window toolbar.

Record & Analyze					₩ Ф
🕨 🔹 🖾 🔁 🔽 🗒 🛱 🛞					
Sequence	Stop Recording		Direction	Method	Instance
□ 🗹 00000001		id	Call	CAppointmentDlg.OnOK	
 ✓ 00000002 ✓ 00000003 	End recording of the current program's execution		Call Call	CBCGPAppointment.SetAll CBCGPAppointment.SetAll	

4.1.5 Generating Sequence Diagrams

Once you have captured activity and are about to generate the diagram, firstly select a package in the Project Browser where you intend the Sequence diagram to be stored. Then use the toolbar on the Record & Analyze window to generate the diagram.

4.1.5.1 The Recording History

All information recorded as a result of the application encountering recording markers set by the user is held in the Record & Analyze window.

Record & Analyze 🔹 🔻 X					
🕨 🔍 🚍 🕄 💱 🛛	• 🗸 🖻 I	异 璀 ⑧			
Sequence	Instance	Method	Direction	Method	In 📤
□ □ 00000001		steppingtmainCRTStartup	Call	stepping.wmain	
 ✓ 00000002 ✓ 00000003 		stepping.wmain SSP.CTrace.Run	Call Call	SSP.CTrace.Run SSP.CTrace.Get	
00000004		SSP.CTrace.Get	Call	SSP.CTrace.SetRect	=
0000005			Return	SSP.CTrace.Get	
0000006		CCD CT D	Return	SSP.CTrace.Run	
0000007		SSP.CTrace.Run SSP.CTrace.Func3	Call Call	SSP.CTrace.Func3 SSP.CTrace.SetRect	
0000009			Return	SSP.CTrace.Func3	
☑ 00000010			Return	SSP.CTrace.Run	
☑ 00000011 ☑ 00000012		SSP.CTrace.Run SSP.CTrace.Func4	Call Call	SSP.CTrace.Func4 SSP.CTrace.Func5	
00000012		SSP.CTrace.Func5	Call	SSP.CTrace.Func6	
00000014		SSP.CTrace.Func6	Call	SSP.CTrace.Func7	
☑ 00000015			Return	SSP.CTrace.Func6	Ψ.
•					•
🔎 Breakpoints & 📋	Memory \	/iewer 🛛 📃 Debug Output 🖓	🔍 Search 🛛	Output 📃 Record & Ana	alyze

The columns in this window are as follows:

Sequence - The unique sequence number

Note:

The checkbox against each number is used to control whether or not this call should be used to create a Sequence diagram from this history. In addition to enabling or disabling the call using the checkbox, you can use context menu options to enable or disable an entire call, all calls to a given method, or all calls to a given Class.

- Threads The operating system thread ID
- Delta The elapsed thread CPU time since the start of the sequence
- **Method** There are two **Method** columns: the first shows the caller for a call or for a current frame if a return; the second shows the function called or function returning
- **Direction** Stack Frame Movement, can be *Call, Return, State, Breakpoint* or *Escape* (*Escape* is used internally when producing a Sequence diagram, to mark the end of an iteration)
- Depth The stack depth at the time of a call; used in the generation of Sequence diagrams
- State The state between sequences
- **Source** There are two **Source** columns: the first shows the source filename and line number of the caller for a call, or for a current frame if a return; the second shows the source filename and line number of the function called or function returning.
- **Instance** There are two **Instance** columns; these columns only have values when the Sequence diagram produced contains State transitions. The values consist of two items separated by a comma the first item is a unique number for the instance of the Class that was captured, and the second is the actual instance of the Class.

For example: supposing a Class *CName* has an internal value of 4567 and the program created two instances of that Class; the values might be:

- 4567,1
- 4567,2

The first entry shows the first instance of the Class and the second entry shows the second instance.

4.1.5.2 Generate a Diagram

To generate a Sequence diagram for all history click on the toolbar Create Sequence Diagram icon (11)

To generate a Sequence diagram for a single sequence, select it and then click the toolbar **Create Sequence Diagram** icon (**H**).

4.1.5.3 Diagram Features

The Sequence diagram produced includes the following:

References

When the VEA cannot match a function call to an operation within the model, it still creates the sequence, but it creates a reference for any Class that it cannot locate. It does this for all languages.

Fragments

Fragments displayed in the Sequence diagram represent loops or iterations of a section(s) of code. The VEA does its best to match function scope with method calls to as accurately as possible represent the execution visually.

States

If a State Machine has been used during the recording process, any transitions in State are presented after the method call that caused the transition to occur. States are calculated on the return of every method to its caller.

4.1.5.4 Saving Recording

To save a sequence to an XML file, click on the sequence and on the toolbar Save button (

To access an existing sequence file, either:

- Click on the toolbar Open icon (), or
- Right-click on a blank area of the screen and click on the **Load Sequence From File** context menu option.

The Windows Open dialog displays, from which you select the file to open.

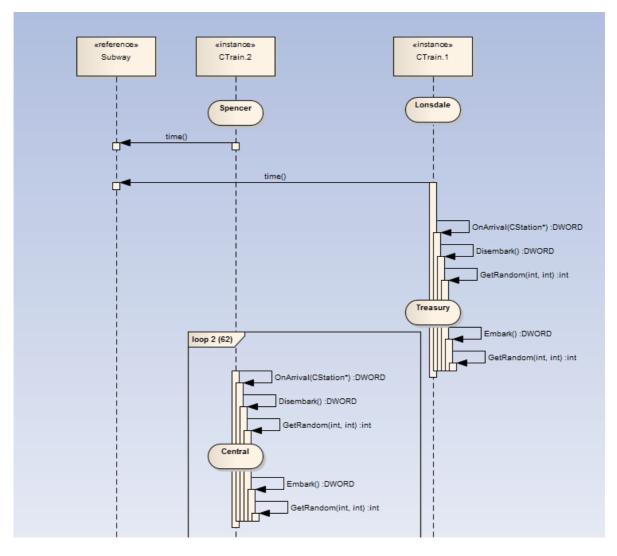
4.1.6 Add State Transitions

This topic describes how to add State Transitions. It covers:

- Setup for Capturing State Changes 76
- The State Machine 77
- Recording and Mapping State Changes.

4.1.6.1 Setup for Capturing State Changes

You can generate Sequence diagrams that show transitions in state as a program executes. The illustration below shows a project that has, in its State Machine, a number of States that correspond to stations in the Melbourne underground railway system.

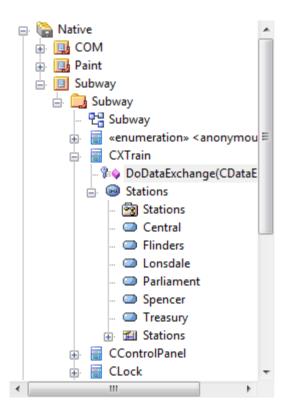


Showing State transitions on your debug-generated Sequence diagrams is optional; you set an option in the package script associated with the Class for which you intend to record States.

Note:

If you do not have a package script for the Class or package you must create one. Sequence diagrams can only be generated for a package that has been configured for debug.

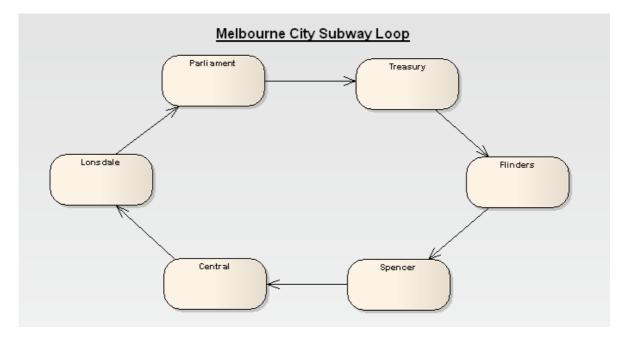
Next, you create a *State Machine* under the Class. On the State Machine you create the *State* elements that correspond to any states to be captured for your Class. The debugger evaluates your States by checking *constraints* on the States you create. The States on this diagram are then used by the debugger and State transitions are incorporated into the diagram.



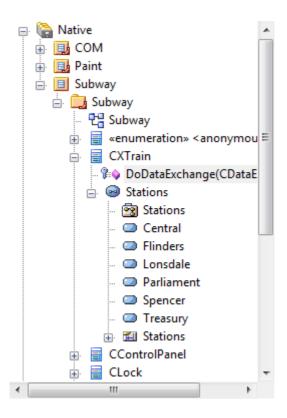
This figure shows the Class *CXTrain* with a State Machine called *Stations*. It has a child diagram also called *Stations*, on which the States {*Central*,*Flinders*,*Lonsdale...*} are placed.

4.1.6.2 The State Machine

A State Transition diagram can be used to illustrate how States change during the execution of an application. The Visual Execution Analyzer can build a State Machine to model all the valid system states and explicitly describe the transitions between each state. The diagram below is a State Machine that shows the different States within the Melbourne Underground Loop subway system. A train traveling on the subway network can be stopped at any of the stations represented on the State Machine below.



This State Machine diagram is a child of the CXTrain Class.



4.1.6.3 Recording and Mapping State Changes

The State Properties dialog below is for the State *Parliament*. The Constraints tab is open to show how the State is linked to the Class *CXTrain*. A State can be defined by a single constraint or by many; in the example below the State *Parliament* has two constraints.

State : Parliament
General Requirements Constraints Links Scenarios Files Tagged Values
Constraint:
▲ Type: Invariant ▼
Status: Approved
$\mathbf{B} I \underline{\mathbf{U}} \mathbf{A} \mathbf{i} \equiv \frac{1}{2} \equiv \left \begin{array}{c} \mathbf{x}^2 & \mathbf{x}_2 \end{array} \right _{\mathbf{A}}$
Defined Constraints
Constraint Type Status
Location=0 Invariant Approved Departing.Name=Parliament Invariant Approved

The CXTrain Class has a member called *Location* of type *int,* and a member called *Departing.Name* of type *CString.*

The values of constraints can only be compared for *elemental, enum* and *string* types. What this constraint means is:

- when an instance of the CXTrain Class exists and
- its member variable Location has the value 0 and
- the member variable Departing.Name has the value Parliament then
- this State is evaluated to **true**.

Operators in Constraints

There are two types of operators you can use on constraints to define a State:

- · Logical operators AND and OR can be used to combine constraints
- Equivalence operators {= and !=} can be used to define the conditions of a constraint.

All the constraints for a State are subject to an AND operation unless otherwise specified. You can use the OR operation on them instead, so you could rewrite the constraints in the above example as:

Location=0 OR

Location=1 AND

Departing.Name!=Central

Below are some examples of using the equivalence operators:

Departing.Name!=Central AND

Location!=1

Note:

Quotes around strings are optional. The comparison for strings is always case-sensitive in determining the truth of a constraint.

4.2 Unit Testing



Enterprise Architect supports integration with unit testing tools in order to make it easier to develop good quality software.

Firstly, Enterprise Architect helps you to create test Classes with the JUnit and NUnit transformations (see the MDA Transformations User Guide). Then you can set up and a test script 53 against any package and run it. Finally, all tests results are automatically recorded and inside Enterprise Architect.

4.2.1 Set Up Unit Testing

{

}

In order to use unit testing in Enterprise Architect, you must first set it up. This happens in two parts.

Firstly the appropriate tests must be defined 53. Enterprise Architect is able to help with this. By using the JUnit or NUnit transformations and code generation (see Code Engineering Using UML Models) you can create test method stubs for all of the public methods in each of your Classes.

The following is an NUnit example in C# that is followed through the rest of this topic, although it could also be any other .Net language or Java and JUnit.

```
[TestFixture]
public class CalculatorTest
    [Test]
    public void testAdd(){
               Assert.AreEqual(1+1,2);
    }
    [Test]
    public void testDivide(){
               Assert.AreEqual(2/2,1);
    }
    [Test]
    public void testMultiply(){
               Assert.AreEqual(1*1,1);
    }
    [Test]
    public void testSubtract(){
               Assert.AreEqual(1-1,1);
    }
```

This code can be reverse engineered into Enterprise Architect so that Enterprise Architect can record all test results against this Class.

Once the unit tests are set up, you can then set up the Build and Test scripts to run the tests. These scripts must be set up against a package.

The sample above can be called by setting up the Package Build Scripts 12 dialog as follows.

Execution Analysis | Unit Testing

Name:	C# Calculator	
Name:	C# Calculator	
Directory:	C:\Calculator	
Build Te	est Run Debug Deploy Sequence Diagram Recording	
Enter you	ir test script below and select the appropriate output parser for the type of testing required	
"C:\prog	ram files/Wunit/bin/nunit-console.exe"pin/debug/Calculator.exe	
		·
Captur	re Output 📝 Build before Test Output Parser: NUnit	t 🔻
- ·		
	OK	Cancel Help

If Enterprise Architect is to handle unit testing, it is important that you select the **Capture Output** checkbox and select the appropriate **Output Parser** for the testing. Without doing this you won't see the program output and therefore you cannot open the source at the appropriate location.

4.2.2 Run Unit Tests

You can run the test script you set up previously, by selecting the **Project | Execution Analyzer | Test** menu option.

The following output is generated.

Output	×
NUnit version 2.2.2 Copyright (C) 2002-2003 James W. Newkirk, Michael C. Two, Alexei A. Vorontsov, Charlie Poole. Copyright (C) 2000-2003 Philip Craig. All Rights Reserved.	
OS Version: Microsoft Windows NT 5.1.2600.0 .NET Version: 1.1.4322.573 F	
Tests run: 4, Failures: 1, Not run: 0, Time: 0.031 seconds Failures:	
1) CalculatorTest.testSubtract : expected:<0>	
but was:<1> at NUnit.Framework.EqualAsserter.FailNotEqual() at NUnit.Framework.EqualAsserter.Assert()	
at CalculatorTest.testSubtract() in C:\SimpleCalculatorCS\CalculatorTest.cs:line 23 Test completed with exit code 1	
I I System Script Build	

Notice how NUnit reports that four tests have run, including one failure. It also reports what method failed and the file and line number the failure occurred at. If you double-click on that error, Enterprise Architect opens the editor to that line of code.

81

Source Code
20
21 [Test]
22 public void testSubtract() {
23 Assert.AreEqual (1-1,1);
24 }
25 }

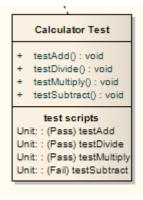
This enables you to quickly find and fix the error.

Enterprise Architect also records the run status of each test as described in Record Test Results 822.

4.2.3 Record Test Results

Enterprise Architect is able to automatically record all results from tests by a <u>testing script</u> and in Enterprise Architect. In order to use this feature, you just reverse engineer the test Class (see *Code Engineering Using UML Models*) into the package containing your test script.

Once your model contains your test Class, on the next <u>run of the test script</u> **Enterprise** Architect adds test cases to the Class for each test method found. On this and all subsequent test runs all test cases are updated with the current run time and if they passed or failed as shown in the following illustration.



The error description for each failed test is added to any existing results for that test case, along with the current date and time. Over time this provides a log of all test runs where each test case has failed. This can then be included in generated documentation and could resemble the following.

```
Failed at 05-Jul-2006 1:02:08 PM
expected: <0>
but was: <1>
```

Failed at 28-Jun-2006 8:45:36 AM expected: <0> but was: <2>

4.3 Profiling Native Applications

The Visual Execution Profiler enables you to quickly report on:

- The most frequently called functions in a running application
- Tasks in an application that are taking more time than expected
- Which functions are taking the most time in an application.

The Profiler, or sampler, is available in the Enterprise Architect Professional, Corporate, Business and Software Engineering, System Engineering and Ultimate editions.

Note:

The Profiler only works with MS Native Windows applications, but can be used under WINE (Linux and Mac) to debug standard Windows applications deployed in a WINE environment.

Profiler		_	άx
🔸 🗢 🛄 🛛 👘 2 📮 🍟	• 🖾 💿 🚽		_
Item			
🗐 Summary		-	
Target	MFC.exe		
PID	1300		
- Session	1		
- Functions	226		
- Modules	32		
— Current sampling time	0.0191		
Max sampling time			
Mean idle time	1.7289		
🗄 - Threads	Sampled	Processed	Time
···· Thread: 740	260	260	
Thread: 3784			
Thread: 1192			
Thread: 2888			
Thread: 284			

The Profiler can generate a report that shows how these functions are called in relation to the application, as illustrated below:

tack			Inclusive Hits	Hits	Inclusive Hi	Hits?
Thread: 2848			276		100%	
🚊 wWinMainCRTStartu			273		99%	
🛓tmainCRTStartu			273		99%	
i⊒∘ wWinMain			273		99%	
CMFCApp	hitInstance		273		99%	
GBCGF	DIFrameWnd::LoadFrame		264		96%	
<u>⊨</u> . CN	nFrame::OnCreate		255		92%	
	MainFrame::OnAppLook		212		77%	
	CBCGPVisualManager::SetDefaultMan	nager	212		77%	
	GBCGPTabbedControlBar::ResetT	abs	205		74%	
	CBCGPVisualManager::GetInst	lance	205		74%	
	GBCGPVisualManager2010	::OnUpdateSystemColors	204		74%	
	GBCGPVisualManager2	2010::SetStyle	203		74%	
	GBCGPVisualMana	ger2007::SetResourceHandle	200		72%	
	☐ CBCGPVisualMi	anager2010::OnUpdateSystemColors	200		72%	
	GBCGPTagN	Manager::ExcludeTag	85		31%	
		1	84		30%	
	BCGCBP	RO1100ud90	1	1	0%	
	CBCGPCont	trolRenderer::Create	32		12%	
	GBCGPTagN	Manager::ReadControlRenderer	62		22%	
		agManager::ParseControlRenderer	56		20%	
	EBC	GPControlRenderer::Create	49		18%	
		BCGPToolBarImages::LoadStr	48		17%	
		CBCGPPngImage::Load	46		17%	
		CBCGPPngImage::LoadFromBuffer	43		16%	
		ATL::CImage::Load	35		13%	
		ATL::CImage::CreateFromGdiplusBitmap	17		6%	
		Gdiplus::Bitmap::LockBits	15		5%	
		Image::GetPitch Image::GetPitch	2	1	1%	

Visual Execution Analyzer in Enterprise Architect

See Also

- Profiler System Requirements 84
- Profiler Operation 85

4.3.1 System Requirements

Prerequisites

The <u>Profiler window</u> becomes available when a model is opened. <u>Options</u> on the Profiler window toolbar enable you to attach to an existing process or launch a new application if a Package Script been specified.

Supported Platforms

Enterprise Architect supports profiling on native Windows applications (C, C++ and Visual Basic) compiled with the Microsoft[™] native compiler where an associated PDB file is available. Select **Microsoft Native** from the list of debugging platforms in your package script.

The Profiler can sample both Debug and Release configurations of an application, providing the PDB for each executable exists and is up to date.

4.3.2 Getting Started

The Profiler window can be accessed by selecting the View | Execution Analyzer | Profiler menu option, or by selecting it from the *Analysis Windows* folder on the Debugger window ([Alt]+[8]). The toolbar options are explained in the table below.

The Profiler operates by taking samples of a process at intervals of up to 250 milliseconds. At these intervals the Profiler interrupts the process and collects stack information for all threads running at that time. This information is sent back to Enterprise Architect where it is collected sorted and and stored.

You can Pause and Resume profiling at any time during the session. You can also clear any sample data collected and begin again.

If you stop the Profiler and the process is still running, you can quickly attach to it again.

lcon	Use to
+	(When an application is <u>configured for the package) [82]</u> create the Profiler process, which launches the configured application.
0	Profile an application that is already running.
\$ 11	When the application is running, pause and resume sample capture. Pausing sampling enables the Report and Erase buttons.
0	Stop the Profiler process. If any samples have been collected, the Report button is enabled.
1	Generate a report set on the current number of samples collected.
5 🛟	Set the interval, in milliseconds, at which samples are taken of the target process. The range of possible values is 1 - 250 .
1	 Set Profiler options, using a drop-down menu. The options are: Load Report from Disk - load and display a previously-generated report from an xml disk file Package Build Scripts ([Shift]+[F12]) - display the Build Script 12 dialog to enable creation or

lcon	Use to
	editing of package scripts and debug configuration
	• Start Sampling Immediately - begin sample collection immediately upon either process start (main thread entry point executed) or attachment of process by Profiler
	Capture Debug output - capture any appropriate debug output and redirect it to the Enterprise Architect Output window
	• Stop Process on Exit - select to terminate the target process when the Profiler is stopped.
D	Erase the collected data.
	Display the Help topic for this window.

4.3.3 Start & Stop the Profiler

For most debugging operations it is necessary to have first configured a Package Script that typically defines the application to build, test and debug as well as sequence recording options.

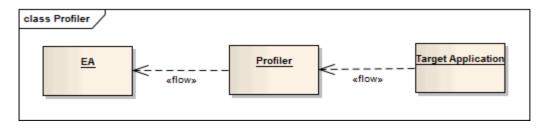
It is possible to use the Profiler without doing any of this by using the Attach to Process button.

If the Application to Profile is the one defined in the current Package, use the **Launch** button.

+ 0	₃ ∞ 👘 2 🗘 🎁 ▾ 💟 🛞
+	(When an application is <u>configured for the package)</u> SE create the Profiler process, which launches the configured application.
0	Profile an application that is already running.
۲	Stop the Profiler process.

4.3.4 Profiler Operation

Enterprise Architect creates a Profiler process whenever you click on the **Launch** or **Attach to Process** button on the **Profiler** window toolbar. This process operates by collecting samples from the stacks of every thread in the target process.



The sampler process exits if you click on the **Stop** button, if the target application terminates, or if you close the current model.

You can turn sample collection on and off at any time during a session. When sampling is turned on or resumed, the Profiler process becomes active and samples are collected from the target. Resuming sampling collects completely new samples.

The Profiler process idles if sampling is turned off or paused during a session. The **Report** and **Erase** buttons then become enabled.

Click on the **Report** button to produce a call graph summary similar to that in the <u>Visual Execution Profiler</u> (82) topic. This report can be saved to file.

Click on the Erase button to discard any samples currently collected for the target.

4.3.5 Setting Options

Interval

5 🛟	Set the interval, in milliseconds, at which samples are taken of the target process. The range of possible values is 1 - 250 .
*	 Set Profiler options, using a drop-down menu. The options are: Start Sampling Immediately - begin sample collection immediately upon either process start (main thread entry point executed) or attachment of process by Profiler Capture Debug output - capture any appropriate debug output and redirect it to the Enterprise Architect Output window Stop Process on Exit - select to terminate the target process when the Profiler is stopped.

4.3.6 Save and Load Reports

The Profiler Reports can be *saved* in either binary format or xml format. Save the report using the toolbar above the report (Stack) view.

ack		Inclusive Hits	Hits	Inclusive Hi	Hits?
Thread: 2848		276		100%	
wWinMainCRTStartup		273		99%	
tmainCRTStartup		273		99%	
🛓 wWinMain		273		99%	
CMFCApp::InitInstance		273		99%	
GBCGPMDIFrameWn	d::LoadFrame	264		96%	
🚊 CMainFrame::On	Create	255		92%	
🖨 CMainFrame:	OnAppLook	212		77%	
	ualManager::SetDefaultManager	212		77%	
GBCGF	PTabbedControlBar::ResetTabs	205		74%	
⊟- CB	CGPVisualManager::GetInstance	205		74%	
	CBCGPVisualManager2010::OnUpdateSystemColors	204		74%	
	CBCGPVisualManager2010::SetStyle	203		74%	
	CBCGPVisualManager2007::SetResourceHandle	200		72%	
	CBCGPVisualManager2010::OnUpdateSystemColors	200		72%	
	CBCGPTagManager::ExcludeTag	85		31%	
	ia⊷ mfc90ud	84		30%	
	BCGCBPRO1100ud90	1	1	0%	
	CBCGPControlRenderer::Create	32		12%	
	CBCGPTagManager::ReadControlRenderer	62		22%	
	CBCGPTagManager::ParseControlRenderer	56		20%	
	CBCGPControlRenderer::Create	49		18%	
	CBCGPToolBarImages::LoadStr	48		17%	
	GBCGPPngImage::Load	46		17%	
	CBCGPPngImage::LoadFromBuffer	43		16%	
	ATL::CImage::Load	35		13%	
	ATL::CImage::CreateFromGdiplusBitmap	17		6%	
	⊕ Gdiplus::Bitmap::LockBits	15		5%	
		2	1	1%	

To *load* a report use the Profiler Toolbar **Options** button and select the Load Report From Disk option.

Profiler		_	άx
🔸 💿 🛄 💿 👘 2 🌲 🍟	• 🖾 💿 🚽		_
Item			
Target	MFC.exe		
···· PID	1300		
- Session	1		
- Functions	226		
Modules	32		
— Current sampling time	0.0191		
Max sampling time			
Mean idle time	1.7289		
- Threads	Sampled	Processed	Time
···· Thread: 740	260	260	
Thread: 3784			
Thread: 1192			
Thread: 2888			
Thread: 284			

4.3.7 Save Report in Team Review

You can save any current report as a resource for a Category, Topic or Post in the Team Review. The report can then be shared and reviewed at any time as it is saved with the model.

No current diagram				•)	×	feam Review		
 		_						t Model> 💌 🔎 🎯 🖥 CDe
Call Stack	Inclusive Hits	Hits	Inclusive Hi.	Hits%		····· 🍋 Profi	es	New Topic
	14		100%					New Topic from Template
_DIIMainCRTStartup	6		43%					Rename (F2)
DIIMainCRTStartup	6		43%					
General CRT_INIT	4		29%					Copy Path To Clipboard
initterm	4		29%					Show Contents
`dynamic initializer for 'afx	3		21%					Share Resource
AFX_GLOBAL_DATA::A	3		21%	ickage from C	urren	it Model		Share Resource
□ AfxCtxLoadLibraryA	2		14% Im	age of Active	Diag	ram		Refresh Category 'Profiles'
🖃 kernel32	2		14% Ad	dd Active Profi	iler R	eport		Reload Current Connection
ntdll	2	1	14% Im	age from Clip	boar	d		
AFX_GLOBAL_DAT	1		7%				d Act	ive Profiler Report
`dynamic initializer for 'g_b	1		7%					
	2		14%				N.	Connections
in AfxLoadLangResourceDLL	2		14%					Options
	1		7%					
kernel32	1		7%					Delete Category 'Profiles'
CActivationContext::Create	1		7%				0	Help
kornol22	1		79/		Ŧ		9	

4.4 Object Workbench

This section describes the Object Workbench:

- How it works 88
- Workbench variables 88
- <u>Create Workbench Variables</u>
- Invoke Methods. 90

4.4.1 How it Works

The Workbench is a tool in Enterprise Architect Debugging, enabling you to <u>create your own variables</u> and <u>invoke methods</u> on them. Stack trace can be recorded and Sequence diagrams produced from the invocation of such methods. It provides a quick and simple way to debug your code.

Platforms Supported

The Workbench supports the following workbench platforms:

- Microsoft .NET (version 2.0 or later)
- Java (JDK 1.4 or later)

Note:

The Workbench does not currently support the creation of Class instances written in native C++, C or VB.

Mode

The Workbench operates in two modes:

Idle mode

When the Workbench is in idle mode, instances can be created and viewed and their members inspected.

Active mode

When methods are invoked on an instance, the Workbench enters *Active* mode, and the variables displayed change if the debugger encounters any breakpoints. If no breakpoints are set, then the variables do not change. The Workbench immediately returns to *Idle* mode.

Logging

The results of creating variables and the results of calls on their methods are displayed in the Debug Output window.

4.4.2 Workbench Variables

You can create (and delete) workbench variables from any Class in your model. When you do so, you are asked to name the variable. It then displays in the Workbench window. It shows the variable in a hierarchy, displaying its type and value and those of any members.

Workbench		×
Variable	Value	Туре
🖃 🖕 Rob		MyClassLibrary.CRobert
🗆 🧅 MyClassLibrary.CPerson		
AverageAge	0	float
FriendCount	1	int
💊 Age	2	int
🖃 🧅 Friends		MyClassLibrary.CPerson[]
E 🧄 [0]		MyClassLibrary.CPerson
🖕 Town	"Daylesford"	String
Name	"Robert"	String
occupation	"Programmer"	String
🖃 🖕 Fred		MyClassLibrary.CFred
🖃 🧅 MyClassLibrary.CPerson		
💊 AverageAge	0	float
FriendCount	0	int
💊 Age	2	int
🛨 🧅 Friends		0
🖕 Town	"hepbum"	String
🖕 Name	"Fred"	String
 occupation 	"Programmer"	String

Workbench Requirements

- NET framework version 2 is required to workbench any .NET model.
- The package from which the variable is created must have a debugger configured (see the <u>Debug Tab</u> 15) topic).

Constraints (.NET)

- Members defined as struct in managed code are not supported.
- Classes defined as internal are not supported.

Delete Workbench Variables

You can delete variables using the **Delete** shortcut menu on any instance on the Workbench. If all instances are deleted the debugger is shut down, and the Workbench window is closed.

4.4.3 Create Workbench Variables

Right-click on the required Class node in the Project Browser and select the **Create Workbench Instance** context menu option, or press **[Ctrl]+[Shift]+[J]**. The menu option is also available from within a Class diagram.

Naming the Workbench

When you elect to create an instance of a type Enterprise Architect prompts you with the Workbench dialog to name the variable. Each instance name must be unique for the workbench.

William		
Value		
MyClassLibra	y.CPerson	

Choosing a Constructor

Having given the variable a name, you must now choose which constructor to use.

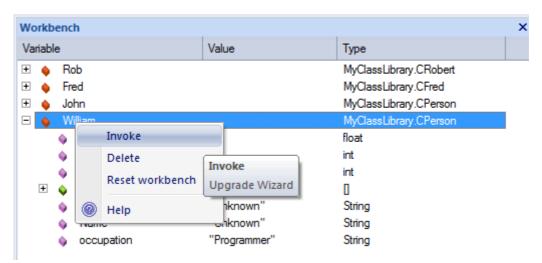
If you do not define a constructor, or define a single constructor taking no arguments, the default constructor or the defined constructor is automatically invoked.

Otherwise the following dialog displays. Select the constructor from the drop-down list and fill in any parameters required.

William (MyClassLibrary:: CPerson)		
CPerson()		-
CPerson()		
CPerson(MyClassLibrary.CPerson) CPerson(string,string,int)		
	Invoke Help	Cancel

4.4.4 Invoke Methods

On the Workbench window, right-click on the instance on which to execute a method, and select the **Invoke** context menu option.



Choose Method

A list of methods for the type are presented in a dialog. Select a method from the list and click on the **Invoke** button. Note that all methods listed are public; private methods are not available.

Work	Rob(MyClassLibrary::CRobert)	×
Varia + + + + -	Int AddFriend(MyClassLibrary.CPerson) bool Find(string) Int AddFriend(MyClassLibrary.CPerson) Int AddFriends(MyClassLibrary.CPerson) Int CreateFriends(mt) bool Find(string]) float GetAverage() bool IsFriend(MyClassLibrary.CPerson) void main(string]) root GetAverage() bool IsFriend(MyClassLibrary.CPerson) void main(string])	
Ŧ	string Occupation() void Run(string]) void Set(string,int) int SetAge(int) void SetAverage(float) void SetName(string) String SetTown(string) void Test(string)	
	Invoke Help Cancel	

Supply Arguments

In this example, you have created an instance or variable named *Rob* of type *MyClassLibrary.CRobert*, and have invoked a method named *AddFriends* that takes an array of *CPerson* objects as its only argument. What you now supply to it are the three other Workbench instances *Fred, John* and *William*.

Workb	ench				×
Variabl	le	Valu	e	Туре	
+ • + • + •	Rob Fred John Willian	1 erageAge 0		MyClassLibrary.CRobert MyClassLibrary.CFred MyClassLibrary.CPerson MyClassLibrary.CPerson	
	 Fr 	Rob(MyClassLibrary::CRo	bert)	-100	X
+	♦ A♦ Fr	int AddFriend(MyClassLibr	ary.CPerson)		•
T	 Fr Tr 	MyClassLibrary.CPerson	F	red, John, William	
	• 00				
			I	nvoke Help	Cancel

Arguments

In the dialog above, type any parameters required by the constructor.

- Literals as arguments
 - Text: abc or "abc" or "a b c"
 - Numbers: 1 or 1.5

• Objects as arguments

If an argument is not a literal then you can supply it in the list only if you have already created an instance of that type in the workbench. You do this by typing the name of the instance as the argument. The debugger checks any name entered in an argument against its list of workbench instances, and substitutes that instance in the actual call to the method.

• Strings as arguments

Surrounding strings with quotes is unnecessary as anything you type for a string argument becomes the value of the string; for example, the only time you should surround strings with quotes is in supplying elements of a string array, or where the string is equal to the name of an existing workbench instance.

"A b c" "a b \$ % 6 4" A b c d As 5 7) 2 === 4

Arrays as arguments

Enter the elements that compose the array, separated by commas.

Туре	Arguments
String[]	one,two,three,"a book","a bigger book"
CPerson[]	Tom,Dick,Harry

Note:

If you enter text that matches the name of an existing instance, surround it in quotes to avoid the debugger passing the instance rather than a string.

void SetName(string)	•
string	"Bill"
_	
	Invoke Help Cancel

Invoke

Having chosen the constructor and supplied any arguments, click on the **Invoke** button to create the variable. Output confirming this action is displayed in the <u>Output tab</u> 47.

Index

- . -

.NET ASP, Debug 32 Debug 29 **Debug Another Process** 40 Debug Assembly 30 31 Debug CLR Versions Debug With COM Interop Process 32 Debug, System Requirements 16 Set Up Debug Session 30

- A -

Activate Recording Markers 71 Analyzer Windows From the Debug Window 35 Apache Tomcat Server Configuration 28 Server, Debugging 25 Service Configuration 29 Applets Java, In Internet Browsers, Debug 23 ASP .NET Debug 32 Assembly Debug 30 Attach To Process Dialog 40 Automatic Recording Execution Analysis, Recording Sequence Diagrams 72

- B -

Breakpoint Delete 37 **Difference From Recording Marker** 72 Disable 37 Enable 37 Management 37 Set For Modifiable Data 38 Set In Code 38 States 37 Storage 38 Breakpoints And Markers Window 71 **Build Script** Create 12

Deploy Script, Create 56 Enable Diagnostic Messages, Sequence **Diagram Recording Tab** 64 Enable Filter, Sequence Diagram Recording Tab 60 Filters 60 Limit Auto Recording, Sequence Diagram Recording Tab 63 Options, Sequence Diagram Recording Tab 59 Record Arguments To Function Calls, Sequence Diagram Recording Tab 61 Record Calls To Dynamic Modules, Sequence **Diagram Recording Tab** 62 Record Calls To External Modules, Sequence Diagram Recording Tab 61 Recursive 14 Run Script, Create 55 Test Script, Create 53 Wildcard in Filter 60 **Build Scripts** Introduction 12

- C -

C++**Debug Symbols** 20 Set Up Debug Session 19 Call Stack Copy To Recording History 50 **Create Sequence Diagram** 49 Save 50 View 42 Window 42 **Capture State Changes** Setup To, Visual Execution Analyzer 76 **CLR Versions** Debug .NET 31 Code Breakpoint, Set 38 Debug, Step Into Function Calls 41 Debug, Step Out Of Functions 42 Debug, Step Over Lines 41 Debug, Step Through Function Calls 49 Code Breakpoint Set 38 Code Editor, Common Debug Tooltips 48 Tooltips, Debug 48 COM Interop Debug .NET 32 Command Deploy, Create 56

Command Deploy, Introduction 56 Run, Create 55 Run, Introduction 55 Unit Test, Create 53 Unit Test, Introduction 53 Configuration Apache Tomcat Server 28 JBOSS Server 27 **Tomcat Server** 28 29 **Tomcat Service** Control Recording **Execution Analysis, Recording Sequence** Diagrams 72 Create Build Script 12

- D -

Data Breakpoint, Set 38 Data Breakpoint Set 38 Debug .NET 16, 29 .NET Assembly 30 .NET CLR Versions 31 .NET With COM Interop Process 32 Another .NET Process 40 ASP .NET 32 Break On Variable Changing Value 46 Create Sequence Diagram, Call Stack 49 Deploy Script, Create New 56 Deploy Script, Introduction 56 File Search, Introduction 51 File Search, Use 51 Inspect Process Memory 45 Java 16, 21 Java Applets In Internet Browsers 23 Java Web Servers 25, 29 Java, Advanced Techniques 22 Java, General Setup 21 On Windows 7 And Windows Vista 17 Platforms 16 **Recording Actions** 49 Run Script, Create New 55 Run Script, Introduction 55 Save Call Stack 50 Script Search 51 Search Window 51 Show Loaded Modules 47 Show Output 47

41 Step Into Function Calls Step Out Of Functions 42 Step Over Lines Of Code 41 Step Through Function Calls 49 **Tooltips In Code Editor** 48 Under Windows Vista 16 Unit Test Script, Create 53 Unit Test Script, Introduction 53 View Call Stack 42 **View Local Variables** 43 View Local Variables, Long Values 43 View Variables In Other Scopes 44 WINE Applications 18 **Debug Session** Debug C++ 19 Java, Attach To VM 22 Microsoft Native Setup 19 Set Up 15 Set Up For .NET 30 Set Up For Microsoft Native 19 **Debug Symbols** Debug C++ 20 Microsoft Native 20 Debugger Actions 15 **Debug Another Process** 40 **Detatch From Process** 40 **Execution Analysis** 57 Frameworks 15 How It Works 15 Introduction 15 On Windows 7 And Windows Vista 17 Overview 15 Process 15 Start 40 Stop 40 System Requirements 16 **Debugger Windows** From the Debug Window 35 **Debugging Actions** 39 **Deploy Command** Create 56 Introduction 56 **Deploy Script** Create 56 Introduction 56 Dialog Attach To Process 40 **DIB Data Access Violation** 18 **Disable Recording Markers** 71

- E -

ΕA

Execution Analyzer, Introduction 4 Enable Diagnostic Messages Build Script, Sequence Diagram Recording Tab 64 **Enable Filter** Build Script, Sequence Diagram Recording Tab 60 **Execution Analysis** Add State Transitions 75 Automatic Recording, Record Sequence Diagrams 72 Breakpoints And Markers Window, Record Sequence Diagrams 71 Control Recording, Record Sequence Diagrams 72 **Diagram Features, Generate Sequence** Diagrams 75 Difference Between Recording Marker And Breakpoint 72 Generate Sequence Diagram 74,75 Introduction 57 Manual Recording, Record Sequence Diagrams 73 Marker Types, Record Sequence Diagrams 66 89 **Object Workbench, Create Variables Object Workbench**, Introduction 87 Object Workbench, Invoke Methods 90 Object Workbench, Overview 88 **Object Workbench**, Variables 88 Pause Recording, Record Sequence Diagrams 73 Place Markers, Recording Sequence Diagrams 66 Platforms 57 Profiler Operation 85 Profiler Report, Load 86 Profiler Report, Save 86 Profiler Report, Save As Resource In Team Review 87 Profiler Toolbar 84 Profiler, Attach To Process 85 Profiler, Getting Started 84 Profiler. Launch 85 Profiler, Overview 82 Profiler, Prerequisites 84 Profiler, Set Options 86 Profiler, Set Sample Intervals 86 Profiler. Start 85 Profiler, Stop 85

84 Profiler, Supported Platforms Profiler, System Requirements 84 **Record Activity For Class** 64 Record Activity For Method 65 Record Sequence Diagrams, Advanced Techniques 64 Record Sequence Diagrams, Enable Filter 60 Record Sequence Diagrams, Introduction 57 Record Sequence Diagrams, Overview 57 Record Sequence Diagrams, Prerequisites 59 Record Sequence Diagrams, Recording Options 59 Record Sequence Diagrams, Set Up 59 Record Unit Test Results 82 Recording History 74 Recording Markers, Activate, Record Sequence Diagrams 71 Recording Markers, Disable, Record Sequence Diagrams 71 Resume Recording, Record Sequence Diagrams 73 Run Unit Test 81 Save Recording History 75 Sequence Diagrams, Enable Diagnostic Messages 64 Sequence Diagrams, Limit Auto Recording 63 Sequence Diagrams, Record Arguments To Function Calls 61 Sequence Diagrams, Record Calls To Dynamic Modules 62 Sequence Diagrams, Record Calls To External Modules 61 Set Recording Markers, Record Sequence Diagrams 70 Set Up To Capture State Changes 76 State Machine Diagram 77 State Transition Diagram 77 Stop Recording, Record Sequence Diagrams 73 Team Review, Save Profiler Report As Resource 87 Unit Test Script, Create 80 Unit Test, Record Results 82 Unit Testing, Introduction 80 With Enterprise Architect 57 Work With Marker Sets, Record Sequence Diagrams 72 **Execution Analyzer** Introduction **Execution Profiler** Attach To Process 85 Getting Started 84 Launch 85 Operation 85

Execution Profiler Overview 82 Prerequisites 84 Report, Example 82 86 Report, Load Report, Save 86 Report, Save As Resource In Team Review 87 Set Options 86 Set Sample Intervals 86 Start 85 Stop 85 Supported Platforms 84 System Requirements 84 Team Review, Save Report As Resource 87 Toolbar 84

- F -

File Search Introduction 51 List View 51 Search Window, Debugging 51 Toolbar 51 Tree View 51 Use 51 Function Step Out Of 42 **Function Call** Step Into 41 Step Through 49

- G -

Generate Sequence Diagram Execution Analysis 74, 75 Generate Sequence Diagrams Diagram Features 75

- | -

Internet Browser Applets Java, Debug 23 Invoke Method, Object Workbench 90

- J -

Java Advanced Debug Techniques 22 Applets In Internet Browsers, Debug 23

Visual Execution Analyzer in Enterprise Architect

Debug 21 Debug Session, Attach To VM 22 Debug, System Requirements 16 General Debug Setup 21 Web Servers, Debugging 25 JBOSS Server Configuration 27 Server, Debugging 25

- L -

Limit Auto Recording to Stack Frame Threshold Build Script, Sequence Diagram Recording Tab 63 Loaded Modules Show In Debugger 47 Local Variables View 43 View Long Values 43 Locals Window 43 View Long Values 43

- M -

Manual Recording Execution Analysis, Recording Sequence Diagrams 73 Map State Changes Visual Execution Analyzer 78 Marker Storage 38 Marker Management Debugger 37 Marker Sets Work With 72 MDDE Advanced Debug Techniques, Java 22 Available Tools 7 Basic Setup 8 **Breakpoint Management** 37 Build Script, Create 12 Build Script, Introduction 12 Code Editors 11 Debug .NET 29 Debug .NET Assembly 30 Debug .NET CLR Versions 31 Debug .NET With COM Interop Process 32 Debug Apache Tomcat Server Configuration 28 **Debug Apache Tomcat Windows Service** 29 Debug ASP .NET 32 Debug Java 21

MDDE

Debug Java Applets In Internet Browsers 23 Debug Java Web Servers 25 **Debug JBOSS Server Configuration** 27 Debug Symbols, C++ And Native Applications 20 **Debugger Frameworks** 15 **Debugger System Requirements** 16 Debugger, Overview 15 Default Script, Set 11 **External Tools** 8 For C++ Applications 19 For Microsoft Native Applications 19 For WINE Applications 18 General Debug Setup, Java 21 General Workflow 8 Generate Code 11 Getting Started 7 Java Debug Session, Attach To VM 22 Limitations 6 Marker Management 37 Model Driven Development Environment 4 Overview 6 Package Build Scripts, Manage 9 Pin/Unpin A Package 11 Prerequisites 7 **Recursive Builds** 14 Script Actions, Define 10 Set Up Debug Session 15 Set Up Debug Session For .NET 30 Supported Environments 6 Synchronize Code 11 **UAC-Enabled Operating Systems** 17 Workspace Layout 8 Memory Viewer Window 45 Method Invoke, Object Workbench 90 **Microsoft Native** Debug Symbols 20 Set Up Debug Sessions 19 Model Driven Development Environment Advanced Debug Techniques, Java 22 Available Tools 7 **Basic Setup** 8 **Breakpoint Management** 37 Build Script, Create 12 Build Script, Introduction 12 Code Editors 11 Debug .NET 29 Debug .NET Assembly 30 Debug .NET CLR Versions 31

Debug .NET With COM Interop Process 32 Debug Apache Tomcat Server Configuration 28 Debug Apache Tomcat Windows Service 29 Debug ASP .NET 32 Debug Java 21 23 Debug Java Applets In Internet Browsers Debug Java Web Servers 25 **Debug JBOSS Server Configuration** 27 Debug Symbols, C++ And Native Applications 20 **Debugger Frameworks** 15 **Debugger System Requirements** 16 Debugger, Overview 15 Default Script. Set 11 External Tools 8 For C++ Applications 19 For Microsoft Native Applications 19 For WINE Applications 18 General Debug Setup, Java 21 General Workflow 8 Generate Code 11 Getting Started 7 Introduction 4 Java Debug Session, Attach To VM 22 Limitations 6 Marker Management 37 Overview 6 Package Build Scripts, Manage 9 Pin/Unpin A Package 11 Prerequisites 7 **Recursive Builds** 14 Script Actions, Define 10 Set Up Debug Session 15 Set Up Debug Session For .NET 30 Supported Environments 6 Synchronize Code 11 UAC-Enabled Operating Systems 17 Workspace Layout 8 Modules Window 47

- 0 -

Object Workbench Constraints 88 Introduction, Visual Execution Analyzer 87 Invoke Method 90 Modes 88 Overview, Visual Execution Analyzer 88 Platforms Upported 88 Requirements 88 Object Workbench Workbench Variables, Constructors 89 Workbench Variables, Create 89 Workbench Variables, Delete 88 Output Debugger, View 47 Debugger, Window 47 Overview Visual Execution Analyzer 2

- P -

Package Pin/Unpin, Visual Execution Analyzer 11 Package Build Scripts Manage, Visual Execution Analyzer 9 Pause Recording Execution Analysis, Recording Sequence Diagrams 73 Place Recording Markers **Execution Analysis, Recording Sequence** Diagrams 66 Process Memory Inspect 45 Profiler Attach To Process 85 **Getting Started** 84 Launch 85 Operation 85 Overview 82 Prerequisites 84 Report, Example 82 Report, Load 86 Report, Save 86 Report, Save As Resource In Team Review 87 Set Options 86 Set Sample Intervals 86 Start 85 Stop 85 Supported Platforms 84 System Requirements 84 Team Review, Save Report As Resource 87 Toolbar 84

- R -

Record & Analyze Window 74 Record Activity For Class Execution Analysis, Record Sequence Diagram 64 Record Activity For Method

Execution Analysis, Record Sequence Diagram 65 **Record Arguments To Function Calls** Build Script, Sequence Diagram Recording Tab 61 **Record Calls To Dynamic Modules** Build Script, Sequence Diagram Recording Tab 62 **Record Calls To External Modules** Build Script, Sequence Diagram Recording Tab 61 **Record Sequence Diagrams** Automatic Recording 72 72 Control Recording Execution Analysis, Advanced Techniques 64 manual Recording 73 Pause Recording 73 Resume Recording 73 Stop Recording 73 **Record State Changes** Visual Execution Analyzer 78 **Recording Actions** Create Sequence Diagram, Call Stack 49 Debugger, Overview 49 Debugger, Step Through Function Calls 49 Save Call Stack 50 Recording History Save, Execution Analysis 75 **Recording Markers** Activate, Execution Analysis 71 Breakpoints And Markers Window, Execution Analysis 71 **Difference From Breakpoint** 72 **Disable**, Execution Analysis 71 Marker Types, Execution Analysis 66 Place, Execution Analysis 66 Set, Execution Analysis 70 Work With Marker Sets, Execution Analysis 72 **Recursive Builds** Visual Execution Analyzer 14 Resume Recording Execution Analysis, Recording Sequence Diagrams 73 Run Command Create 55 Introduction 55 Run Script Create 55 Introduction 55

- S -

Script Default, Set In Visual Execution Analyzer 11 Deploy, Create 56 Deploy, Introduction 56 Run. Introduction 55 Search 51 Unit Test, Create 53 Unit Test, Introduction 53 Script Actions Define, Visual Execution Analyzer 10 Scripts Build 12 Search Debugger File Search 51 File, Introduction 51 Scripts 51 Sequence Diagram **Diagram Features, Generate Sequence** Diagrams 75 Generate From Debugger Call Stack 49 Generate From Recording, Execution Analysis 74 **Generate In Execution Analysis** 57 Generate, Execution Analysis 75 Recording History, Execution Analysis 74 Save Recording History, Execution Analysis Sequence Diagram Recording Tab Build Script, Options 59 Sequence Recording Option 59 Advanced Techniques 64 **Enable Diagnostic Messages** 64 **Enable Filter** 60 Limit Auto Recording 63 Record Activity For Class 64 **Record Activity For Method** 65 **Record Arguments To Function Calls** 61 Record Calls To Dynamic Modules 62 **Record Calls To External Modules** 61 Server Apache Tomcat, Debugging 25 JBOSS, Debugging 25 Tomcat, Debugging 25 Server Configuration JBOSS 27 Tomcat 28 Service Configuration Tomcat 29 Set Up **Debug Session** 15

For .NET 30 State Changes Capture, Execution Analysis 76 Map, Visual Execution Analyzer 78 Record, Visual Execution Analyzer 78 Set Up To Capture, Execution Analysis 76 State Machine Diagram **Execution Analysis** 77 In Visual Execution Analyzer 77 State Transitions Add, Visual Execution Analyzer 75 Step Into **Function Calls** 41 Step Out Of Functions 42 Step Over Lines Of Code 41 Step Through **Function Calls** 49 Stop Recording Execution Analysis, Recording Sequence Diagrams 73

- T -

75

Team Review Save Profiler Report As Resource 87 Test Unit, In Execution Analysis 80 Unit, Record Results In Execution Analysis 82 Unit, Run In Execution Analysis 81 Unit, Set Up In Execution Analysis 80 Test Script Introduction 53 JUnit 80 NUnit 80 The Debug Window 35 Tomcat Server, Configuration 28 Server, Debugging 25 Service Configuration 29 Toolbar (File) Search, Debugging 51

- U -

UAC And Debugging 17 Unit Test Command Introduction 53 Unit Test Script Unit Test Script Create -53 Unit Testing Create Test Scripts, Execution Analysis 80 Define Tests, Execution Analysis 80 Introduction, Execution Analysis 80 JUnit 80 NUnit 80 Record Test Results, Execution Analysis 82 Run, Execution Analysis 81 Set Up, Execution Analysis 80

- V -

Variable Debug, Break On Change In Value 46 Visual Execution Analyzer Access 3 Add State Transitions 75 Advanced Debug Techniques, Java 22 Automatic Recording, Record Sequence Diagrams 72 Availability 2 Break On Variable Changing Value 46 **Breakpoint Management** 37 Breakpoint Storage 38 Breakpoints And Markers Window, Record Sequence Diagrams 71 Build Script, Create 12 Control Recording, Record Sequence Diagrams 72 Create Sequence Diagram, Call Stack 49 Debug .NET 29 Debug .NET Assembly 30 Debug .NET CLR Versions 31 Debug .NET With COM Interop Process 32 Debug Another Process 40 Debug Apache Tomcat Server Configuration 28 **Debug Apache Tomcat Windows Service** 29 Debug ASP .NET 32 Debug Java 21 Debug Java Applets In Internet Browsers 23 Debug Java Web Servers 25 **Debug JBOSS Server Configuration** 27 Debug Symbols, C++ And Native Applications 20 **Debugger Frameworks** 15 **Debugger System Requirements** 16 Debugger Windows, Display 39 Debugger, Overview 15 **Debugging Actions** 39

56 Deploy Script, Create New **Deploy Script, Introduction** 56 **Diagram Features, Generate Sequence** Diagrams 75 Difference Between Recording Marker And Breakpoint 72 Execution Analysis, Introduction 57 File Search, Introduction 51 File Search, Use 51 For C++ Applications 19 For Microsoft Native Applications 19 For WINE Applications 18 General Debug Setup, Java 21 Generate Sequence Diagram 74.75 Inspect Process Memory 45 Java Debug Session, Attach To VM 22 Manual Recording, Record Sequence Diagrams 73 Map State Changes 78 Marker Management 37 Marker Storage 38 Marker Types, Record Sequence Diagrams 66 MDDE Basic Setup 8 **MDDE External Tools** 8 MDDE, Build Scripts 12 MDDE, Code Editors 11 MDDE, Default Script, Set 11 MDDE, Generate Code 11 MDDE, Package Build Scripts, Manage 9 MDDE, Script Actions, Define 10 MDDE, Synchronize Code 11 **Object Workbench, Create Variables** 89 **Object Workbench**, Introduction 87 **Object Workbench, Invoke Methods** 90 Object Workbench, Overview 88 **Object Workbench**, Variables 88 Outputs 2 Overview 2 Pause Recording, Record Sequence Diagrams 73 Pin/Unpin A Package 11 Place Markers, Recording Sequence Diagrams 66 **Profiler Overview** 82 **Record Activity For Class** 64 Record Activity For Method 65 Record Sequence Diagrams, Advanced Techniques 64 Record Sequence Diagrams, Enable Filter 60 **Record Sequence Diagrams, Introduction** 57 Record Sequence Diagrams, Overview 57 **Record Sequence Diagrams, Prerequisites** 59

Index 102

Visual Execution Analyzer Record Sequence Diagrams, Recording Options 59 Record Sequence Diagrams, Set Up 59 **Record State Changes** 78 **Record Unit Test Results** 82 **Recording Actions** 49 Recording History 74 Recording Markers, Activate, Record Sequence Diagrams 71 Recording Markers, Disable, Record Sequence Diagrams 71 **Recursive Builds** 14 Resume Recording, Record Sequence Diagrams 73 Run Script, Create New 55 Run Script, Introduction 55 Run Unit Test 81 Save Call Stack 50 Save Recording History 75 Script Search 51 Search Window 51 Sequence Diagrams, Enable Diagnostic Messages 64 Sequence Diagrams, Limit Auto Recording 63 Sequence Diagrams, Record Arguments To Function Calls 61 Sequence Diagrams, Record Calls To Dynamic Modules 62 Sequence Diagrams, Record Calls To External Modules 61 Set Code Breakpoint 38 38 Set Data Breakpoint Set Recording Markers, Record Sequence Diagrams 70 Set Up Debug Session 15 Set Up Debug Session For .NET 30 Set Up To Capture State Changes 76 Show Loaded Modules 47 Show Output 47 Start Debugger 40 State Machine Diagram 77 State Transition Diagram 77 41 Step Into Function Calls Step Out Of Functions 42 Step Over Lines Of Code 41 Step Though Function Calls 49 Stop Debugger 40 Stop Recording, Record Sequence Diagrams 73 Structure 4 **Tooltips In Code Editor** 48 **UAC-Enabled Operating Systems** 17 Unit Test Script, Create 53,80

53 Unit Test Script, Introduction Unit Test, Record Results 82 Unit Testing, Introduction 80 Uses Of 3 View Call Stack 42 43 **View Local Variables** View Local Variables, Long Values 43 View Variables In Other Scopes 44 Work With Marker Sets, Record Sequence Diagrams 72 Workspace Layouts 8 Visual Execution Profiler Attach To Process 85 Getting Started 84 Launch 85 85 Operation Overview 82 Prerequisites 84 Report, Example 82 Report, Load 86 Report, Save 86 Report, Save As Resource In Team Review 87 Set Options 86 Set Sample Intervals 86 Start 85 Stop 85 Supported Platforms 84 System Requirements 84 Team Review, Save Report As Resource 87 Toolbar 84 Visual Execution Sampler Attach To Process 85 **Getting Started** 84 Launch 85 Operation 85 Overview 82 Prerequisites 84 Report, Example 82 Report, Load 86 Report, Save 86 Report, Save As Resource In Team Review 87 Set Options 86 Set Sample Intervals 86 Start 85 Stop 85 Supported Platforms 84 System Requirements 84 Team Review, Save Report As Resource 87 Toolbar 84 VM Attach To In Java Debug Session 22

- W -

Watched Items Debugger 44 Watches Window 44 Break On Variable Changing Value 46 Web Server Java, Debug 25 Window Breakpoints And Markers 71 Call Stack 42, 48 35 Debug Locals 43 43 Locals, View Long Values Memory Viewer 45 Modules 47 Output, Debugger 47 Record & Analyze 74 Search, Debugging 51 Watches 44 Watches, Break On Variable Changing Value 46 Workbench 88 Windows Service, Apache Tomcat 29 Windows 7 Use Debugger 17 Windows Vista Use Debugger 17 WINE Debugging 18 **DIB Data Access Violation** 18 Workbench Variables Constraints 88 Constructors 89 Create 89 Delete 88 Requirements 88 Workbench Window 88 Workspace Layout For Execution Analysis 8

Visual Execution Analyzer in Enterprise Architect

www.sparxsystems.com