Enterprise Architect

User Guide Series

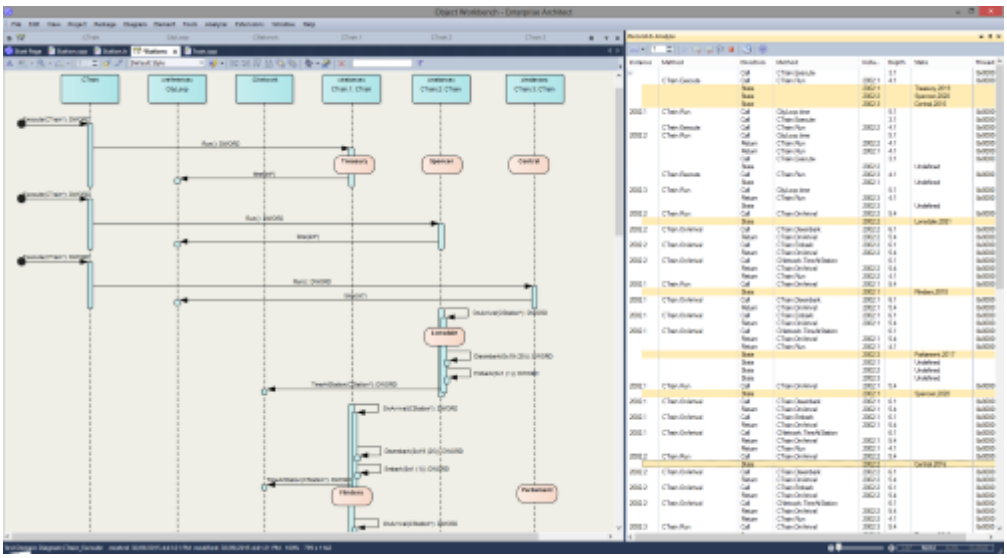
Recording

Recording Application Processes in Sparx Systems Enterprise Architect

Author: Sparx Systems
Date: 2025-05-06
Version: 17.1

Table of Contents

Recording



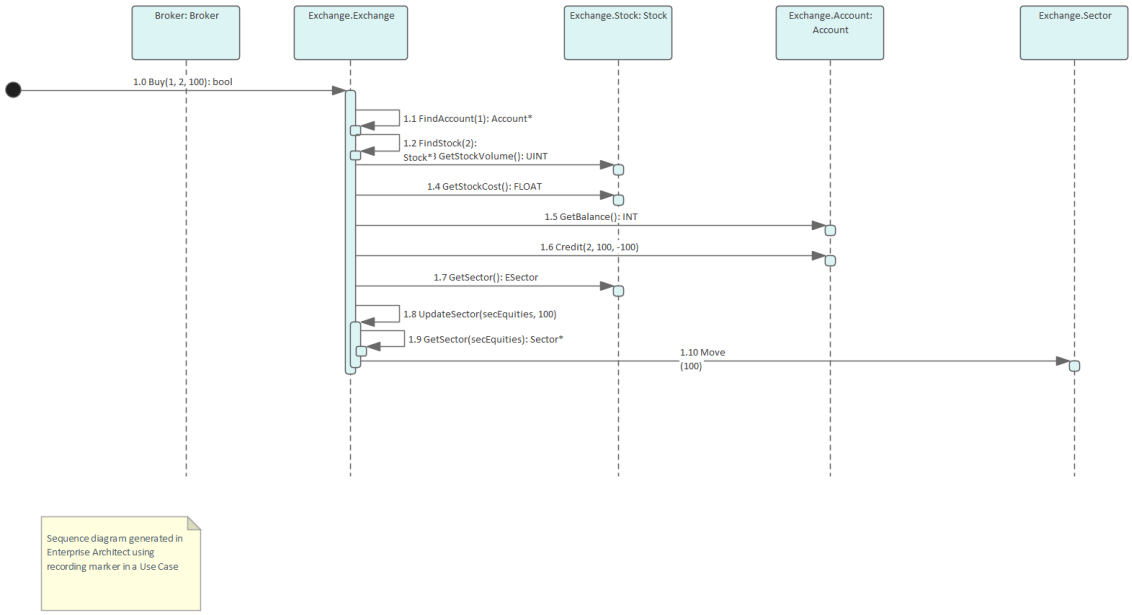
Sequence diagrams are a superb aid to understanding behavior. Class Collaboration diagrams also can be helpful. In addition to these, sometimes a Call Graph is just what we need. Then again, if you have this information available, you could use it to document a Use Case, and why not build a Test domain while you are at it? The Enterprise Architect Analyzer can generate all of these for you and from a single recording. It does this by recording a running program, and it works on all of the most popular platforms.

Access

Ribbon	Execute > Tools > Recorder

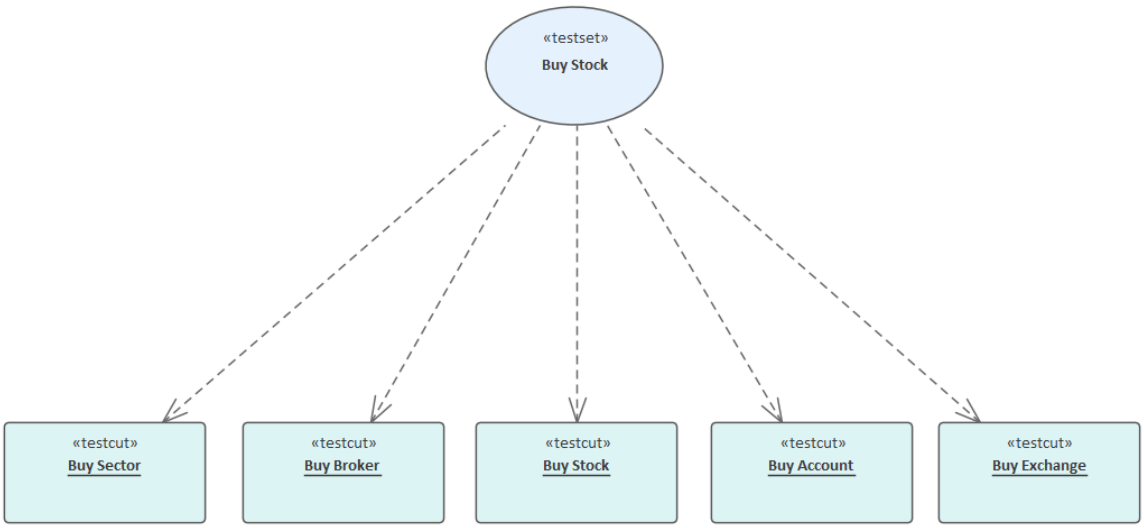
Overview

At its simplest, a Sequence diagram can be produced in very few steps, using even a brand new model. You do not even have to configure an Analyzer Script. Open the Enterprise Architect code editor (Ctrl+Shift+O), place a recording marker in a function of your choice, and then attach the Enterprise Architect Debugger to a program running that code. Any time that function is called, its behavior will be captured to form a recording history. From this history these diagrams can be easily created.



The Sequence diagram from the Example Model recording.

The Class Collaboration diagram from the same recording.

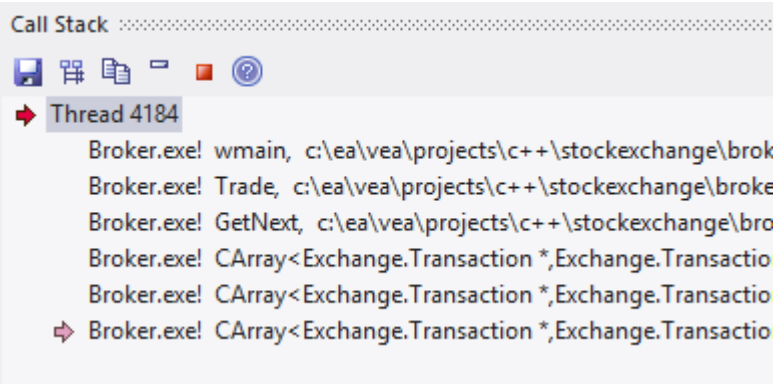


The Test Domain diagram from the same recording.

Of course, an Analyzer Script is still the best idea, and opens up an incredibly rich development environment, but it is worth noting that significant results can be obtained without one. This is also true of the Enterprise Architect Debugger and Profiler tools.

A point of interest: you can view a thread's behavior while it is recorded. Showing the Call Stack during a recording will

show updates to a thread's stack in real-time, much like an animation. It is a good feedback tool and in some circumstances it might be all that is required.



Features at a glance

Diagram Generation

- Sequence diagram
- Class Collaboration diagram
- Test Domain diagram
- State Transition capture
- Call Graph

Control

- Support multi-threaded and single-threaded models
- Support stack depth control 

- Support filters to restrict capture
- Filter wildcard support
- Real-time stack update

Integration

- Class Model

- Test Domain
- StateMachine
- Executable StateMachines
- Unit Tests

Platforms

- Microsoft .NET
- Microsoft Native
- Java
- PHP
- GDB
- Android

Requirements

- Recording is available to users of all editions of Enterprise Architect

Notes

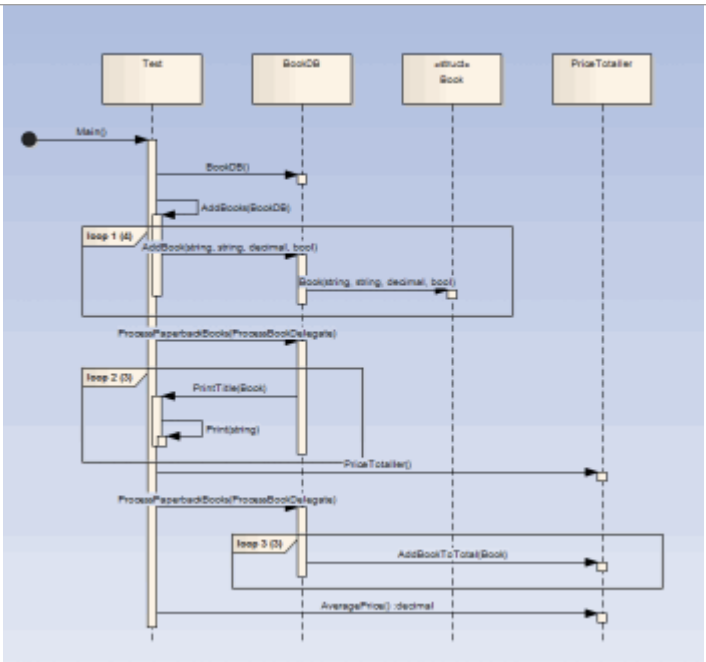
- The debug and record features of the Visual Execution Analyzer are not supported for the Java server platform 'Weblogic' from Oracle

How it Works

This topic explains how the Visual Execution Analyzer generates Sequence diagrams.

Explanation

Points	Detail
Usage	The Visual Execution Analyzer enables you to generate a Sequence diagram from recordings of the live execution of an application. As the application runs, the history of each thread is recorded. This history can be used to generate the Sequence diagram. This is a Sequence diagram generated from a program that calculates the price of books:

	 How does the recorder know what to record? • The recorder works from recording markers; these are placed by you in the functions of interest Call Stacks in Java can stretch further than the eye can see. How can we restrict the recording to just ten frames? • The recorder is controlled by the depth either set on the recorder toolbar or associated with a Marker Set stored in the model
Its the real thing	In recording, the target application is not modified; no instrumentation of any image or module occurs at all. A recording produced using a 'Release'

	build of a program is a trustworthy document of what a program did.
Where do you start	We have a very large server application; so where do we start? If you have little or no understanding of the program you intend to record and little or no model to speak off, you might be best starting with the Profiler. Running the Profiler whilst using a program in a specific manner can quickly identify Use Cases from the entry points and Call Graphs presented. Having that knowledge can enable you to focus on areas that are uncovered and record those functions. If you have the source code, all you need to do is place a recording marker in a function that interests you. We recommend against placing multiple recording markers in multiple functions at the same time. In practice this has shown to be less helpful. Where do you place a recording marker? For windows UI programs, and in relation to some business use case, you might start by placing one in the event handlers for a message that seems most pertinent. If you are investigating a utility function, just set a method recording marker at or

	somewhere near the start. For services, daemons and batch processes you might want to profile the program once for each behavior of interest and use the report to explore those areas uncovered.
Tip	It's a good idea to have a quick glance at the Breakpoints and Markers window before debugging, and check that the markers listed here are what you are expecting.
Scenarios	• Microsoft Native C and C++, VB (Windows programs, Window Services, Console programs, COM servers, IIS ISAPI modules, Legacy) • Microsoft .NET (ASP.NET, Windows Presentation Foundation (WPF), Windows Forms, Workflow Services, devices, emulators) • Java (Apps, Applets, Servlets, Beans) • Android (using Android debug bridge for devices and emulators) • PHP (Web site scripts)

	• GDB (Windows / Linux interopability)

The Recording History

When the execution analysis of an application encounters user-defined recording markers, all information recorded is held in the Record & Analyze window.

Access

Ribbon	Execute > Tools > Recorder > Open Recorder

Facilities

Facility	Information/Options
Information Display	The columns in the Record & Analyze window are: • Sequence - The unique sequence number • 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 the function it is returning to • 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, if a return, for a current frame; the second shows the source filename and line number of the function called or function returning • Instance - There are two Instance columns, which 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
Operations on Information	The Record & Analyze window toolbar provides a range of facilities for controlling the recording of the execution of an Analyzer script. You can perform a number of operations on the results of a recording, using the Record & Analyze window context menu, once the recording is complete.

Notes

- The checkbox against each operation is used to control whether or not this call can be used to create a Sequence, Test Domain Class or Collaborative Class 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

Diagram Features

When you generate a Sequence diagram, it includes these features:

Features

Feature	Detail
References	When the Visual Execution Analyzer cannot match a function call to an operation within the model, it still creates the Sequence but also 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 Visual Execution Analyzer attempts to match function scope with method calls to as accurately as possible represent the execution visually.
States	If a StateMachine 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.

Setup for Recording

This section explains how to prepare to record execution of the application.

Steps

Step
Prerequisites - To set up the environment for recording Sequence diagrams you must: • Have completed the basic set up for Build & Debug and created Execution Analysis scripts for the Package • Be able to successfully debug the application
Narrow the focus of a recording by applying filters.
Control the detail of a recording by adjusting the stack depth.

Control Stack Depth

When recording particularly high-level points in an application, the Stack Frame count can result in a lot of information being collected; to achieve a quicker and clearer picture, it is better to limit the stack depth on the toolbar of either:

- The Breakpoint and Markers window or
- The Record & Analyze window

Access

Ribbon	Execute > Tools > Recorder > Open Recorder

Set the recording stack depth

You set the recording stack depth in the numerical field on the toolbar of the Breakpoints & Markers window or the Record & Analyze window:

By default, the stack depth is set to three frames. The maximum depth that can be entered is 30 frames.

The depth is relative to the stack frame where a recording marker is encountered; so, when recording begins, if the

stack frame is 6 and the stack depth is set to 3, the Debugger records the frames 6 through 8.

For situations where the stack is very large, it is recommended that you first use a low stack depth of 2 or 3. From there you can gradually increase the stack recording depth and insert additional recording markers to expand the picture until all the necessary information is displayed.

Place Recording Markers

This section explains how to place recording markers, which enable you to silently record code execution between two points. The recording can be used to generate a Sequence diagram.

As this process records the execution of multiple threads, it can be particularly useful in capturing event driven sequences (such as mouse and timer events).

Access

Ribbon	Execute > Windows > Breakpoints

Actions

Action
Different recording markers can be used for recording the execution flow; see the related links for information on the properties and usage of these markers.
Manage breakpoints in the Breakpoint & Markers window.

Activate and deactivate markers.
Working with Marker Sets - when you create a breakpoint or marker, it is automatically added to a marker set, either the Default set or a set that you create for a specific purpose.

Notes

- The *Breakpoint and Marker Management* topic (Software Engineering) describes a different perspective

Set Record Markers

Markers are set in the source code editor. They are placed on a line of code; when that line of code executes, the Execution Analyzer performs the recording action appropriate to the marker.

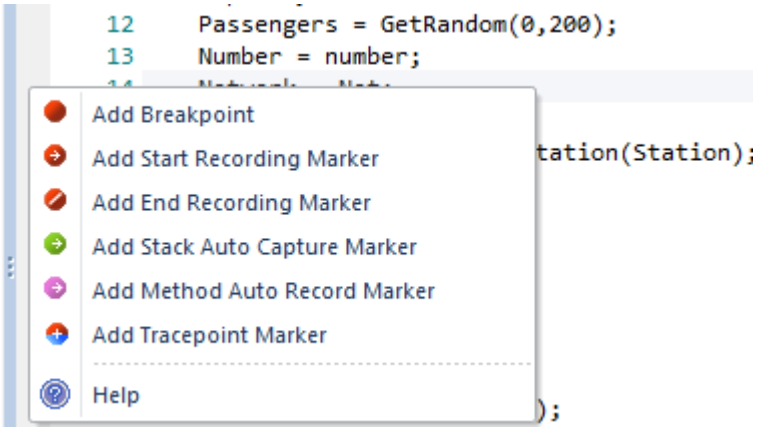
Access

Use one of the methods outlined here, to display the Code Editor window and load the source code associated with the selected Class or Class element.

Ribbon	Execute > Source > Edit > Edit Element Source Execute > Source > Edit > Open Source File
Keyboard Shortcuts	On an element press Ctrl+E or F12 To bring up the 'Open Source File' browser press Ctrl+Alt+O

Set a recording marker

Step	Action

1	Open the source code to debug, in the integrated source code editor.
2	Find the appropriate code line and right-click in the left (Breakpoint) margin to bring up the breakpoint/marker context menu; select the required marker type: 
3	If a Start Recording Marker has been set, you must also set an End Recording Marker.

Marker Types

Markers are really fantastic. Unusual by their very light footprint when used with care, their impact on the performance of the programs being recorded can be negligible. Markers come in several flavors (well colors actually) and more are always being added. They are placed and are visible in the left margin of the editor, so you will need to have some source code.

Use to

- Record a single function
- Record parts of a function
- Use Cases spanning multiple functions
- Record call stacks
- Generate Sequence diagrams
- Generate Test Domain diagrams
- Generate Class Collaboration diagrams

Reference

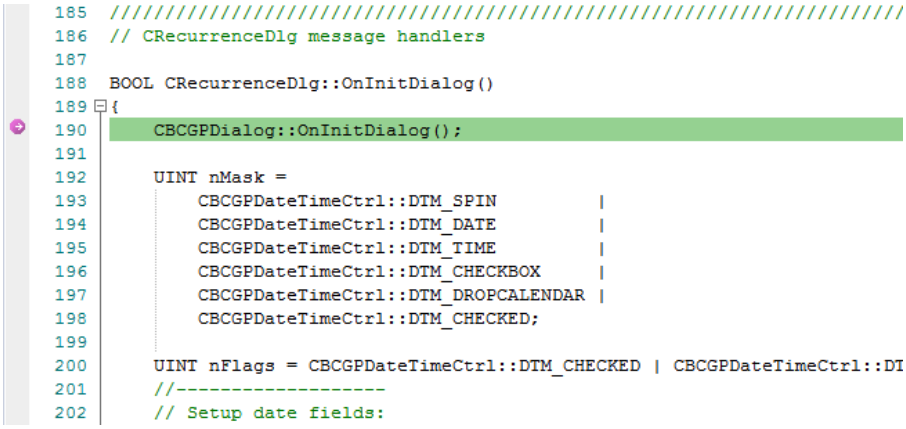
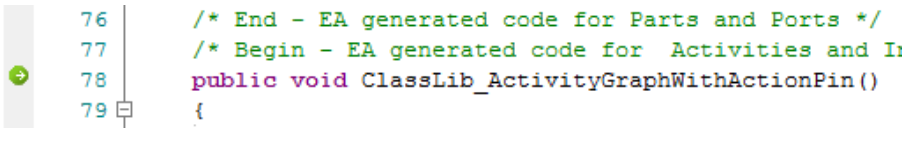
Marker	Detail
Start / End Recording markers	Place the markers at the start and end lines of the code to record. These need not be within the same function.

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

When the program encounters a start recording marker, a new recording is initiated (*the camera starts rolling!).* When an end marker is encountered, the current recording ends *(it's a take*). How you use these markers is up to you and your knowledge of the system under your care.

Advanced Stuff (nested markers):

If a Start recording marker is encountered while a recording is in progress, but where *capture is inhibited by the Stack depth value in use*, a separate recording will be initiated. Each recording is kept on a stack. When one ends, it is removed. This technique can be used in Enterprise Architect to record and render scenes in very complex systems. It resembles splicing short scenes from a video to create a trailer. If you only want to record a single function, you should use an Auto record marker.

Method Auto Record marker	A Method Auto Record marker enables you to record a particular function. The debugger will automatically end the recording when the function completes. This is good because recording is an intensive operation. The function marker combines a Start Recording marker and an End Recording marker in one, so recording is executed after the marker point, and always stops when this function exits.  Recording markers can be nested. When a new Method Auto Record marker is hit while recording, the stack depth to record to will be extended to include the current method and the required depth from that function.
Stack Auto-Capture marker	

```
185 ////////////////////////////////////////////////////////////////////////
186 // CRecurrenceDlg message handlers
187
188 BOOL CRecurrenceDlg::OnInitDialog()
189 {
190     CBCGPDialog::OnInitDialog();
191
192     UINT nMask =
193         CBCGPDateTimeCtrl::DTM_SPIN         |
194         CBCGPDateTimeCtrl::DTM_DATE         |
195         CBCGPDateTimeCtrl::DTM_TIME         |
196         CBCGPDateTimeCtrl::DTM_CHECKBOX     |
197         CBCGPDateTimeCtrl::DTM_DROPCALENDAR |
198         CBCGPDateTimeCtrl::DTM_CHECKED;
199
200     UINT nFlags = CBCGPDateTimeCtrl::DTM_CHECKED | CBCGPDateTimeCtrl::DT
201     //--------------------
202     // Setup date fields:
```

```
76      /* End - EA generated code for Parts and Ports */
77      /* Begin - EA generated code for  Activities and I
78      public void ClassLib_ActivityGraphWithActionPin()
79      {
```

	Stack markers enable you to capture any unique stack traces that occur at a point in an application; they provide a quick and useful picture of where a point in an application is being called from. To insert a marker at the required point in code, right-click on the line and select the 'Add Stack Auto Capture Marker' 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, and the application continues running.
Limiting the recording depth	You can limit the depth of frames in any recording using the stack depth control on either the recorder and breakpoints toolbars.

The Breakpoints and Markers Window

Using the Breakpoints & Markers window, you can apply control to Visual Execution Analysis when recording execution to generate Sequence diagrams; for example, you can:

- Enable, disable and delete markers
- Manage markers as sets
- Organize how markers are displayed, either in list view or grouped by file or Class

Access

Ribbon	Execute > Windows > Breakpoints

Working with Marker Sets

Marker sets enable you to create markers as a named group, which you can reapply to a code file for specific purposes.

You can perform certain operations from the Breakpoints & Markers window alone, but to understand and use markers and marker sets you should also display the appropriate code file in the 'Source Code Viewer' (click on the Class element and press F12).

Access

Ribbon	Execute > Windows > Breakpoints :  toolbar icon

Using Marker Sets

Action	Details
Example of Use	You might create a set of Method Auto Record markers to record the action of various functions in the code, and a set of Stack Capture markers to record the sequence of calls that cause those functions to be called.

	You could then create Sequence diagrams from the recordings under each set.
Create a Marker Set	To create a marker set from the Breakpoints & Markers window, click on the drop-down arrow on the  icon and select the 'New Set' option. The 'New Breakpoint Marker Set' dialog displays; in the 'Enter New Set Name' field, type a name for the set, and click on the Save button. The set name displays in the text field to the left of the 'Set Options' icon. Alternatively, you can select the 'Save as Set' option from the 'Set Options' drop-down to make an exact copy of the currently-selected set, which you can then edit.
Accessing Sets	To access a marker set, click on the drop-down arrow on the text field to the left of the 'Set Options' icon, and select the required set from the list. The markers in the set are listed in the Breakpoints & Markers window. You would normally load a marker set prior to the point at which an action is to be captured. For example, to record a sequence

	involving a particular dialog, when you begin debugging you would load the set prior to invoking the dialog; once you bring up the dialog in the application, the operations you have marked are recorded.
Add Markers to Set	To add markers to a marker set, add each required marker to the appropriate line of code in the 'Source Code Viewer'. The marker is immediately added to whichever set is currently listed in the Breakpoints & Markers window. Each marker listed on the dialog has a checkbox in the 'Enabled' column; newly-added markers are automatically enabled, but you can disable and re-enable the markers quickly as you check the code.
Storage of Sets	When you create a marker set it is immediately saved within the model; any user using the model has access to that set. However, the Default set, which always exists for a model, is a personal workspace, is not shared and is stored external to the model.
Delete a	Right-click on the marker and select the

Marker from a Set	'Delete Breakpoint' option.
Delete a Set	If you no longer require a marker set, access it on the Breakpoints & Markers window and select the 'Delete Selected Set' option from the 'Set Options' drop-down list. You can also clear all user-defined marker sets by selecting the 'Delete all sets' option; a prompt displays to confirm the deletion.

Notes

- Marker Sets are very simple and flexible but, as they are available for use by any user of the model, they can be easily corrupted; consider these guidelines:
  - When naming a set, use your initials in the name and try to indicate its use, so that other model users can recognize its owner and purpose
  - When using a set other than Default, avoid excessive experimentation so that you don't add lots of ad-hoc markers to the set
  - Make sure you are aware of which marker set is exposed in the Breakpoints & Markers window as you can easily inadvertently add markers to the set that are not relevant to the code file the

set was created for

- In any set, if you have added markers that don't have to be kept, delete them to maintain the purpose of the set; this is especially true of the Default set, which can quickly accumulate redundant ad-hoc markers

Control the Recording Session

The Record & Analyze window enables you to control a recording session. The control has a toolbar, and a history window that displays the recording history as it is captured. Each entry in this window represents a call sequence made up of one or more function calls.

Access

Open the Record & Analyze window using one of the methods outlined here.

You must also open the Execution Analyzer window ('Execute > Analyzer | Analyzer Scripts'), which lists all the scripts in the model; you must select and activate the appropriate script for the recording.

Ribbon	Execute > Tools > Recorder > Open Recorder

Recorder Toolbar

You can access facilities for starting, stopping and moderating an execution analysis recording session through the Record & Analyze toolbar.

Access

Ribbon	Execute > Tools > Recorder > Open Recorder Explore > Portals > Show Toolbar > Record

Buttons

Button	Description
	Display a menu of options for defining what the recording session operates on. • Attach to Process - enabled even if no Analyzer Script exists, this option displays a dialog through which you select a process to record and a debugging platform to use; you can also optionally select a record marker

	set and/or a StateMachine to use during the recording • Generate Sequence Diagram from Recording - generate a Sequence/State diagram from the Execution Analyzer trace • Generate Testpoint Diagram from History - generate a Test Domain diagram from the Execution Analyzer trace, that can be used with the Testpoint facility • Generate Class Diagram from History - generate a Collaboration Class diagram from the Execution Analyzer trace, depicting only those Classes and operations involved in the recorded action (Use Case) • Generate Call Graph from History - generate a dynamic Call Graph from the recording history, as you might see in the VEA Profile workspace execution analysis layout; this can be more useful than the Sequence diagram in identifying the unique call stacks involved • Generate All - generate the Sequence, Testpoint and Collaboration Class diagrams together from the Execution Analyzer trace

	• Save as Artifact - create an Artifact element that contains the current recording history, under the currently-selected Package in the Browser window; if you subsequently drag this Artifact element onto a Class diagram and double-click on it, the history recorded in the Artifact is copied back into the Record & Analyze window • Load Sequence History from file - select an XML file from which to restore a previously-saved recording history • Save Sequence History to file - save the recording history to an XML file
	Select the recording stack depth for the marker set; that is, the number of frames from the point at which recording began.
	Launch and record the application described in the script; you can optionally select a record marker set and/or a StateMachine to use during the recording. The icon is enabled when the active Analyzer Script is configured for debugging.

	Perform ad-hoc manual recording of the current thread during a debug session. Use this function with the 'step' buttons of the debugger; each function that is called due to a step command is logged to the history window. The icon is enabled if no recording is taking place and you are currently at a breakpoint (that is, debugging).
	Perform ad-hoc auto-recording during a debug session. When you click on this icon, the Analyzer begins recording and does not stop until either the program ends, you stop the debugger or you click on the Stop icon. This icon is enabled if no recording is taking place and you are currently at a breakpoint (that is, debugging).
	Step into a function, record the function call in the History window, and step back out. Enabled for manual recording only.
	Stop recording the execution trace.

	Display the 'Synchronize Model' dialog through which you can synchronize the model with the code files generated during a Record Profile operation.

Working With Recording History

You can perform a number of operations on or from the results of a recording session, using the Record & Analyze window context menu.

Options

Option	Action
Show Source for Caller	Display the source code, in the Source Code Viewer, for the method calling the sequence.
Show Source for Callee	Display the source code, in the Source Code Viewer, for the method being called by the sequence.
Generate Diagram for Selected Sequence	Generate a Sequence diagram for a single sequence selected in the recording history.
Generate Sequence Diagram	Generate a Sequence diagram including all sequences in the recording history.
Clear	Clear the recording history currently

	displayed in the Record & Analyze window.
Save Recording History to File	Save the recording history to an XML file. A browser window displays, on which you specify the file path and name for the XML file.
Load Recording History From File	Load a previously saved recording history from an XML file. A browser window displays, on which you specify the file path and name for the XML file to load.
Disable All Calls	Disable every call listed in the Record & Analyze window.
Disable This Call	Disable the selected call.
Disable This Method	Disable the selected method.
Disable This Class	Disable the selected Class.
Disable All Calls Outside	Disable every call listed in the Record & Analyze window except for the selected

This Call	call.
Enable All Calls	Enable every call listed in the Record & Analyze window.
Enable This Call	Enable the selected call.
Enable This Method	Enable the selected method.
Enable This Class	Enable the selected Class.
Help	Display the Help topic for the Record & Analyze window.

Start Recording

When you are recording execution flow as a Sequence diagram, you start the recording by selecting the 'Recording' icon on the Record & Analyze window toolbar. The 'Record' dialog displays with the recording options set to the defaults; that is, the current Breakpoint and Markers Set, the filters defined in the current Analyzer Script and the recording mode as basic.

Access

Ribbon	Execute > Tools > Recorder > Open Recorder : 

Record Dialog Options

Field/Button	Detail
Recording Set	Recording markers determine what is recorded. If you have a recording set to use, click on the drop-down arrow and select it.
Additional	Filters are used by the debugger to

Filters	exclude matching function calls from the recording history. Recording filters are defined in the Analyzer Script. In the 'Additional Filters' field you can add other filters for this specific run. if you specify more than one filter, separate them with a semi-colon.
Basic Recording Mode	In basic mode the debugger records a history of the function calls made by the program whenever it encounters an appropriate recording marker.
Track Instances of Named Classes	In Track Instances mode the debugger also captures the creation of instances of the Classes you specify. It then includes that information in the history. The resulting Sequence diagram can then show lifelines for each instance of that Class with, where appropriate, function calls linked to the lifeline.
Track State Transitions	The recording can also capture changes in State using a specified StateMachine diagram. The StateMachine diagram must exist as a child of a Class. The Execution Analyzer captures instances of that Class and calculates the State of each instance whenever a

	function in the current recording sequence returns.
OK	Click on this button to start the debugger.

Step Through Function Calls

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

Alternatively, press Shift+F6 or select the 'Execute > Run > Step In' ribbon 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.

Nested Recording Markers

When a recording marker is first encountered, recording starts at the current stack frame and continues until the frame pops, recording additional frames up to the depth defined on the Recording toolbar. Consider this call sequence:

A -> B -> C -> D -> E -> F -> G -> H -> I -> J -> K -> L -> M -> N -> O -> P -> Q -> R -> S -> T -> U -> V -> W -> X -> Y -> Z

If you set a recording marker at K and set the recording depth to 3, this would record the call sequence:

K -> L -> M

If you also wanted to record the calls X, Y and Z as part of the Sequence diagram, you would place another recording marker at X and the analyzer would record:

K -> L -> M -> X -> Y -> Z

However, when recording ends for the X-Y-Z component (frame X is popped), recording will resume when frame M of the K-L-M sequence is re-entered. Using this technique can help where information from the recorded diagram would be excluded due to the stack depth, and it lets you focus on the particular areas to be captured.

Generating Sequence Diagrams

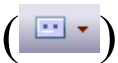
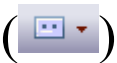
This topic describes what you might do with the recording of an execution analysis session.

Access

Ribbon	Execute > Tools > Recorder > Open Recorder

Reference

Action	Detail
Generate a diagram	Select the appropriate Package in the Browser window, in which to store the Sequence diagram. To create the diagram from all recorded sequences, either: • Click on the 'Recorder Menu' icon () in the Record & Analyze window toolbar, and select the 'Generate Sequence Diagram from Recording' option, or • Right-click on the body of the window

	and select the 'Generate Sequence Diagram' option To create the diagram from a single sequence, either: • Click on the 'Recorder Menu' icon () in the Record & Analyze window toolbar, and select the 'Generate Sequence Diagram from Recording' option, or • Right-click on the sequence and select the 'Generate Diagram from Selected Sequence' option
Save a recorded sequence to an XML file	Click on the sequence, click on the 'Recorder Menu' icon () in the Record & Analyze window toolbar, and select the 'Save Sequence History to File' option.
Access an existing sequence XML file	Either: • Click on the  in the Record & Analyze window toolbar, and select the 'Load Sequence History from File' option, or • Right-click on a blank area of the screen and click on the 'Load Sequence From File' option The 'Windows Open' dialog displays,

	from which you select the file to open.

Use to

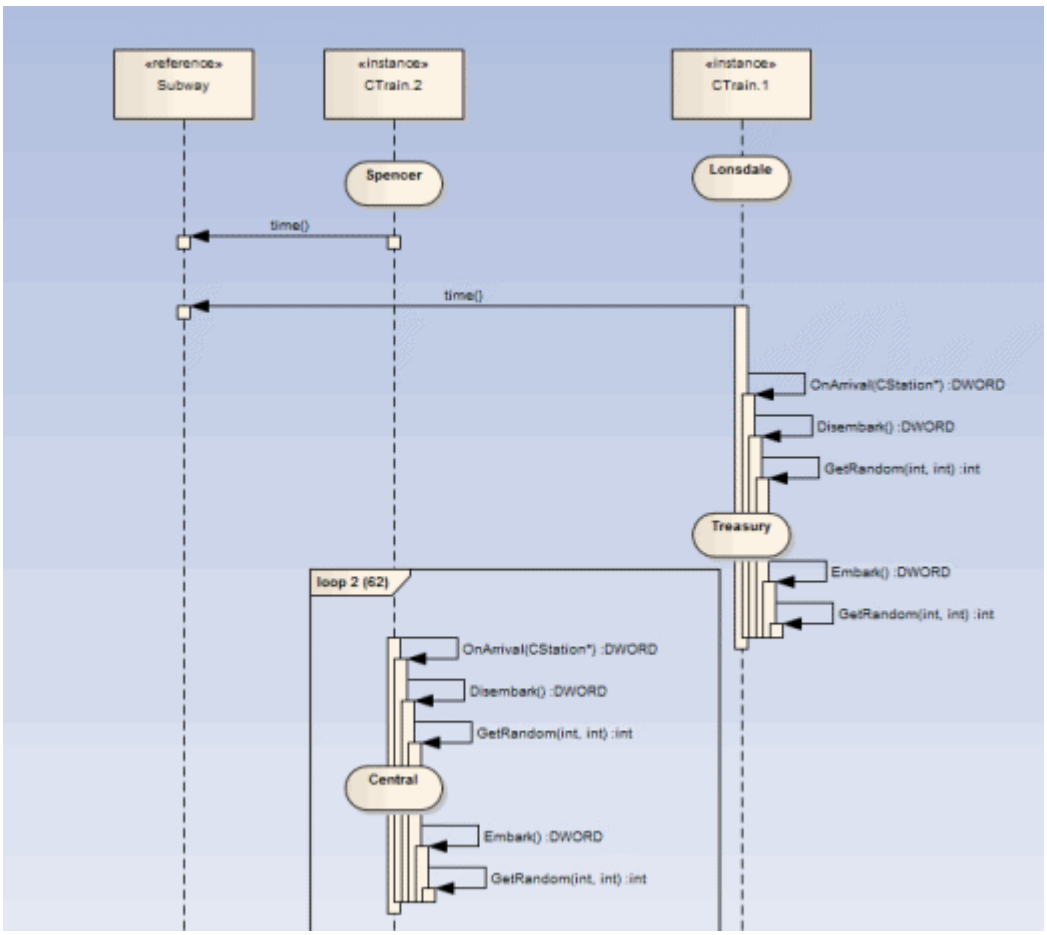
- Generate a Sequence diagram from a recorded execution analysis session, for:
- all recorded sequences or
- a single sequence in the session
- Save the recorded sequence to file
- Retrieve the saved recording and load it into the Record & Analyze window

Reporting State Transitions

This section describes how you can generate Sequence diagrams that show transitions in state as a program executes.

Use to

Generate Sequence diagrams that report user-defined transitions in state as a program executes (as shown in the example generated diagram)



Topic

Create a StateMachine under the Class to be reported.
Set the constraints against each State to define the change in state to be reported.

Reporting a StateMachine

The Execution Analyzer can record a Sequence diagram, we know that. What you might not know is that it can use a StateMachine at the same time to detect State transitions that might occur along the way. These States are represented at the point in time on the lifeline of the object. The transitions also are apparent from the lifelines. Any invalid or illegal transition will be highlighted with a red border. Have a look.

Process

Firstly you model a StateMachine for the appropriate Class element.

You then compose the expressions that define each State using the 'Constraints' tab of each State.

These simple expressions are formed using attribute names from Class model and actual code base. They are not OCL statements. Each expression should appear on a separate line.

```
m_strColor ==  "Blue"
```

You then use the Recorder window to launch the debugger.

The Recorder window Run button is different from the button on other debugger toolbars.

The Recorder window will allow you to browse for a StateMachine if you do not know the StateMachine name. The 'State Transition' dialog presents a list of StateMachines

for the entire model, in which you locate and select the appropriate diagram (see the example).

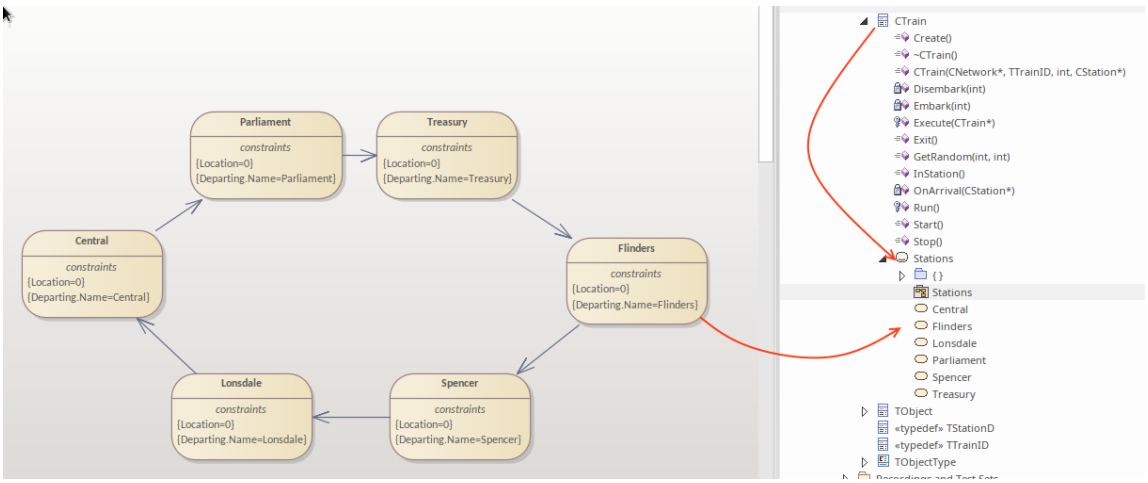
When you generate the Sequence diagram, it depicts not only the sequence but changes in State at the various points in the sequence; each Class instance participating in the detection process is displayed with its own lifeline.

Example

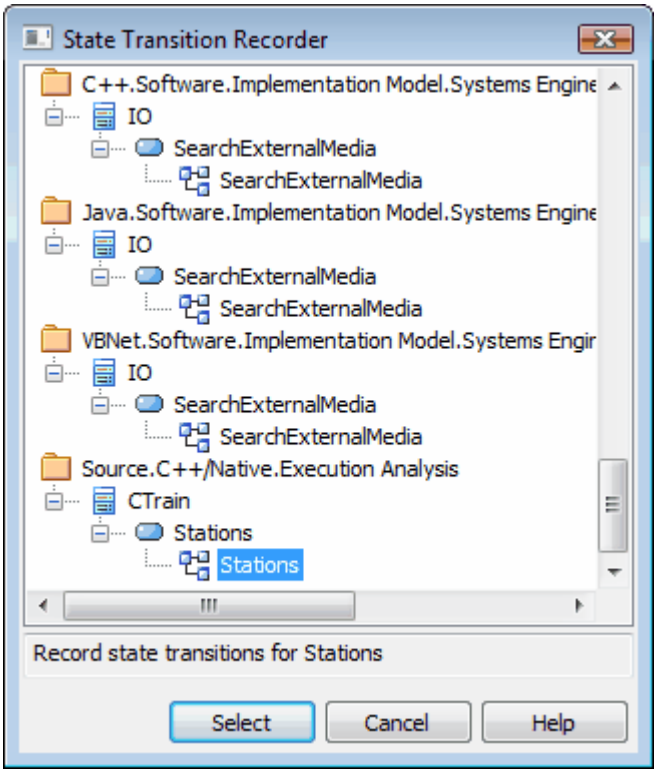
The Stations StateMachine shows the different States within the Melbourne Underground Loop subway system.

A train traveling on the subway network can be stopped at any of the stations represented on the StateMachine.

The Stations StateMachine is a child of the CTrain Class.



When you browse for the diagram in the 'State Transition Recorder' dialog, the hierarchy shows only the root Package, parent Class and child SubMachine and diagram; no other model components are listed.


State Transition Recorder
C++.Software.Implementation Model.Systems Engine
IO
SearchExternalMedia
SearchExternalMedia
Java.Software.Implementation Model.Systems Engine
IO
SearchExternalMedia
SearchExternalMedia
VBNet.Software.Implementation Model.Systems Engir
IO
SearchExternalMedia
SearchExternalMedia
Source.C++/Native.Execution Analysis
CTrain
Stations
Stations
Record state transitions for Stations
Select
Cancel
Help

Recording and Mapping State Changes

This topic discusses how to set constraints against each State in the StateMachine under a Class, to define the change in state to be recorded.

Example

This example of a State 'Properties' dialog is for the State called 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, the Parliament State has two constraints:

The values of constraints can only be compared for elemental, enum and string types

The CXTrain Class has a member called Location of type int, and a member called Departing.Name of type CString; what this constraint means is that this State is evaluated to True 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

Operators in Constraints

There are two types of operator 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 example as:

Location=0 OR

Location=1 AND

Departing.Name!=Central

Here are some examples of using the equivalence operators:

Departing.Name!=Central AND

Location!=1

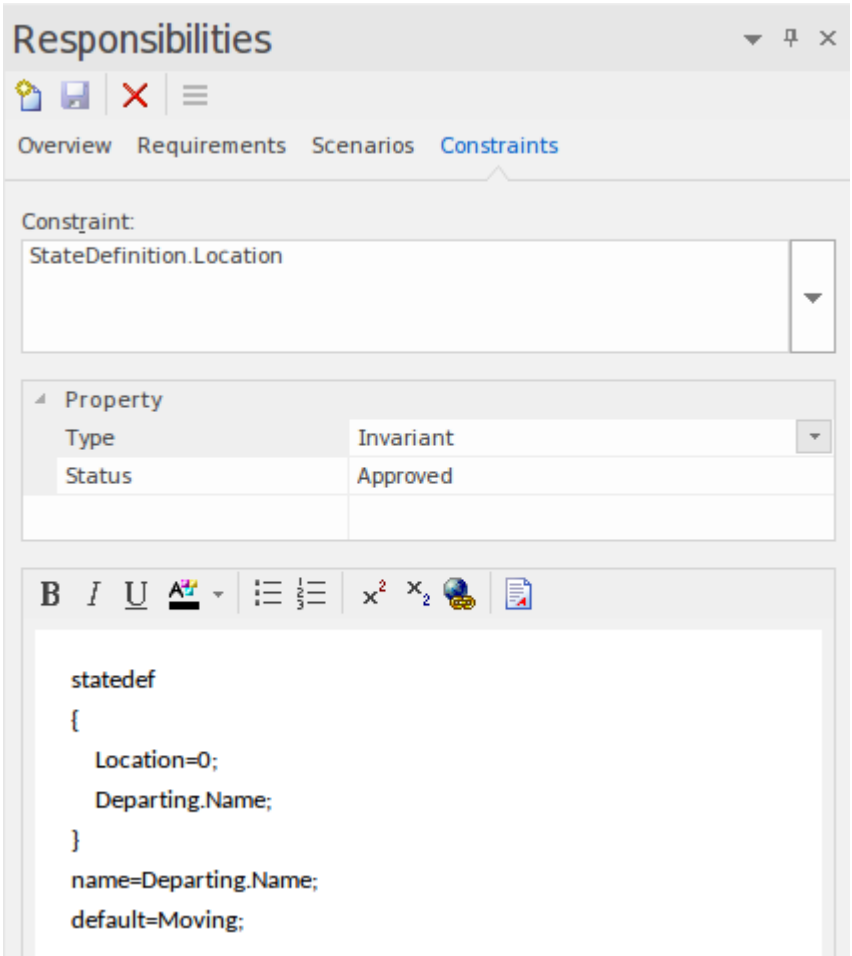
Notes

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

State Analyzer

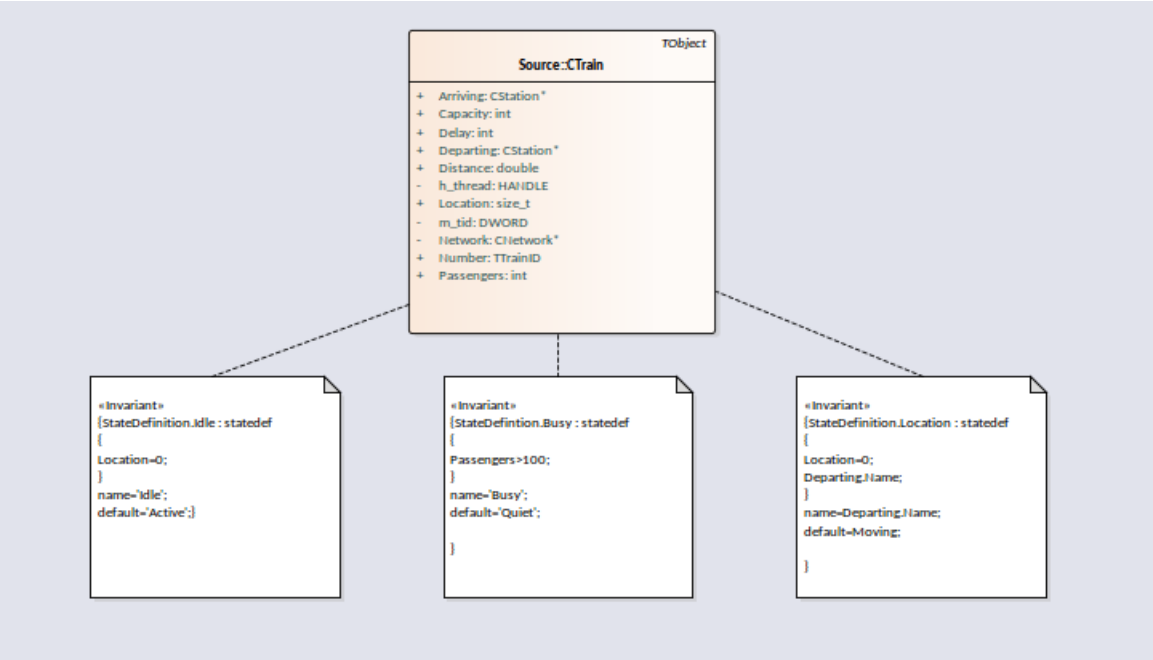
The State Analyzer is a feature that can analyze, detect and record states for instances of a Class. The feature works by combining a state definition (defined on a Class as a constraint) and markers called State points. It is available for any languages supported by the Execution Analyzer, including Microsoft.NET, Mono, Java and native C++.

We begin by selecting a Class and composing our state definition.



We can get a picture of all the state definitions we've defined by placing the Class on a diagram and linking to the Class notes that themselves link to a particular state

definition constraint. We explain how to do that in a later section.

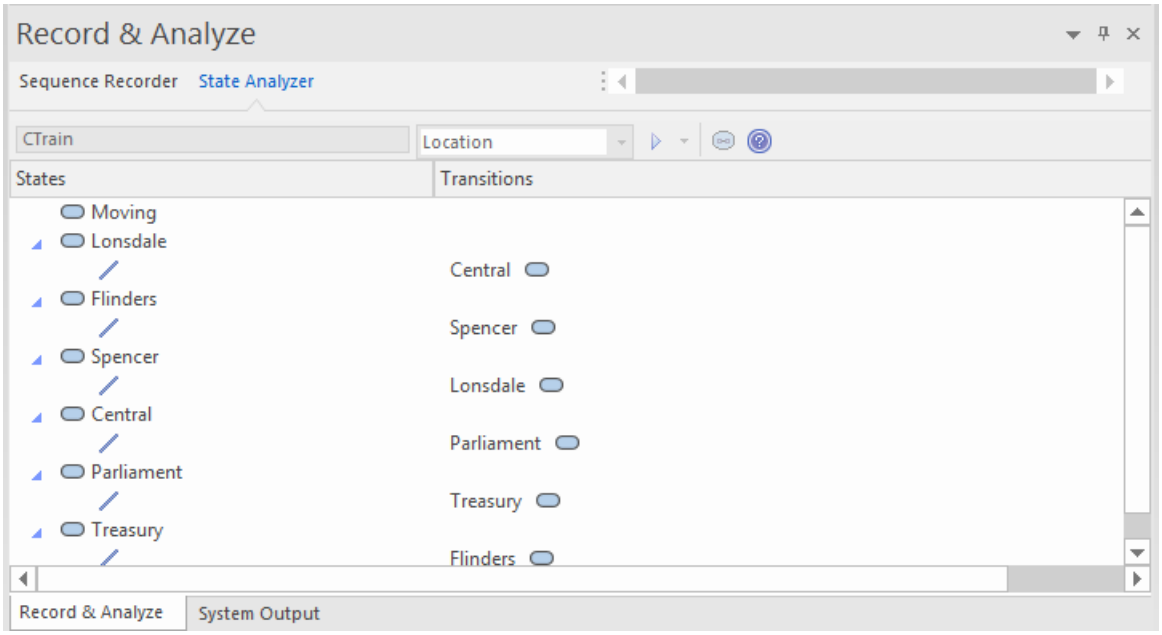


State points are set by placing one or more markers in relevant source code.

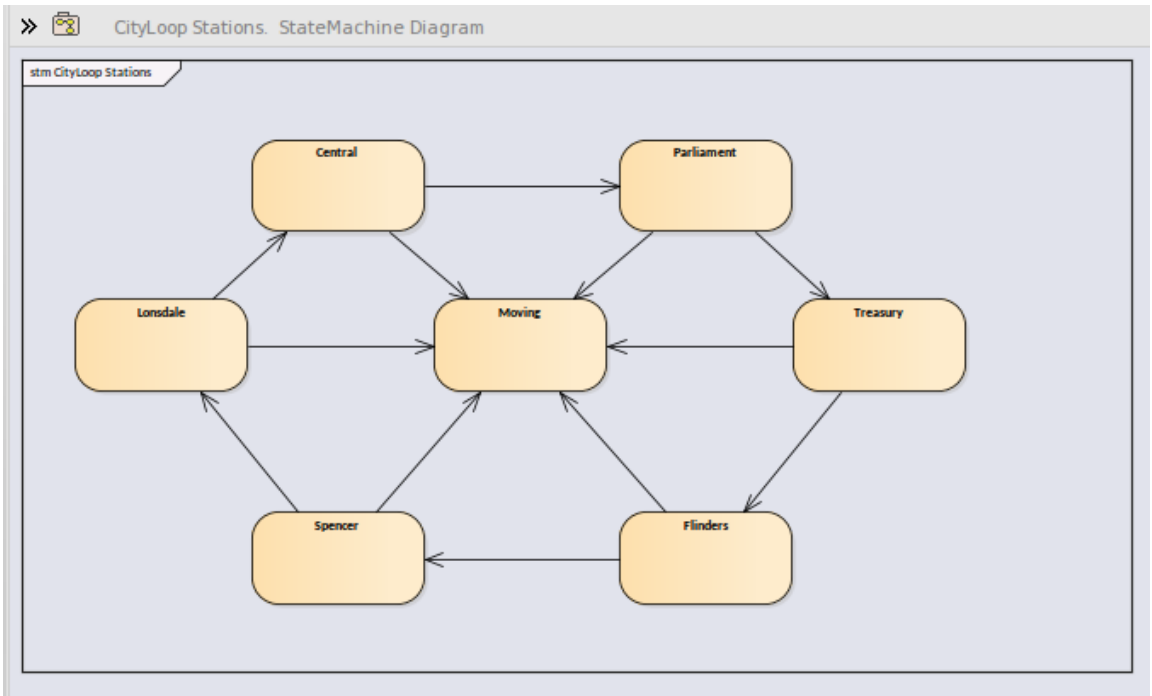
```
73 DWORD   CTrain::OnArrival( CStation* S)
74 {
75     Departing = S;
76     Location = 0;
77     Delay = (Disembark(GetRandom()) + Embark(GetRandom()));
78     DWORD ScheduleTime = Network->TimeAtStation(Departing);
79     if(Delay > (int)ScheduleTime)
80         return Delay;
81     return ScheduleTime;
82 }
```

The program to be analyzed is run using the State Analyzer control. When the Execution Analyzer encounters any State point, the current instance of the Class is analyzed. Where the value domain of the instance matches the state definition, a state is recorded. Each time the instance varies, new states are thus detected. The control lists each state as it is discovered. Under each state the control lists the discrete set of transitions to other states made by instances

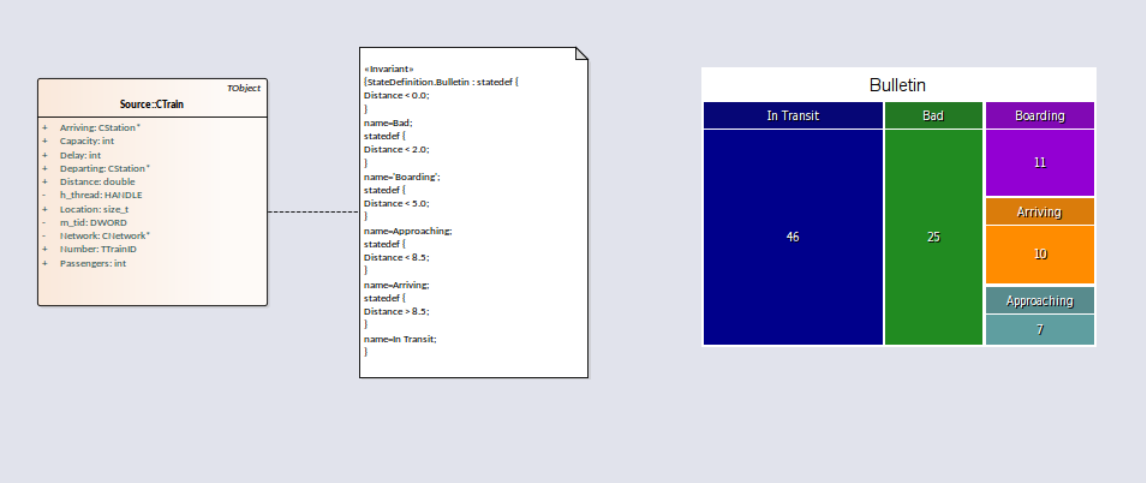
of the class.



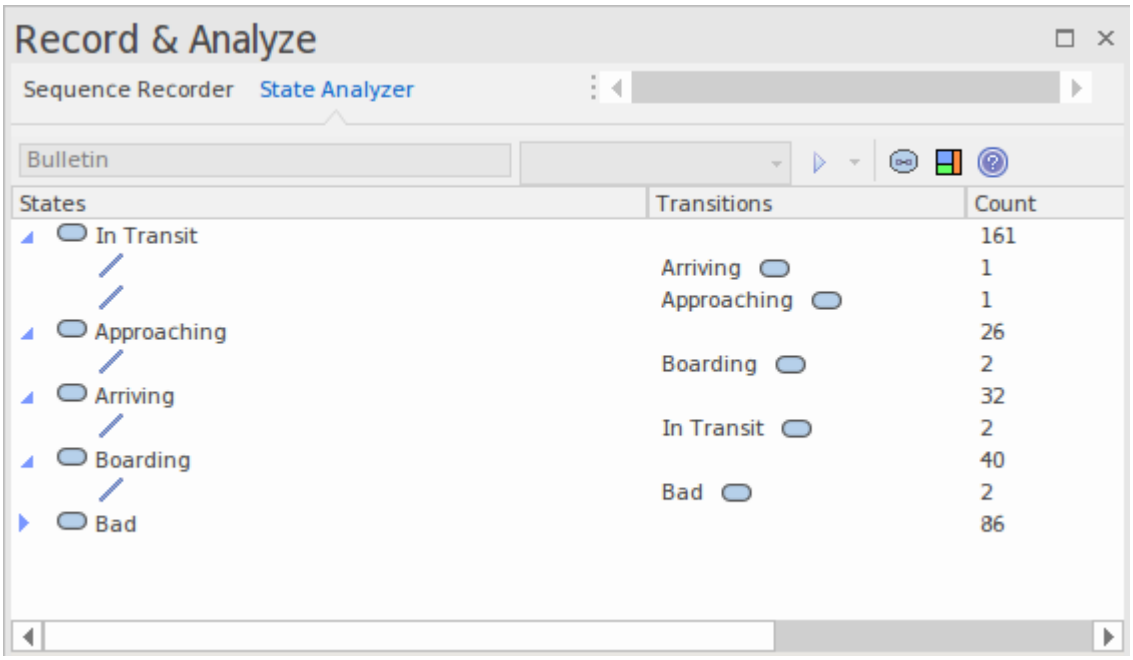
The information can be used to create a StateMachine.



Using the same information we can easily produce a Heat Map. This example shows a 'Train' Class, its 'Bulletin' State Definition (as a linked note), and the Heat Map it produced. The Figures in the map are percentages. From the map we can observe that trains were in the 'In Transit' state 46% of the time.



This is the analysis for the 'Bulletin' State Definition that produced our Heat Map.



Access

Ribbon	Execute > Tools > Recorder > Open Recorder > State Analyzer Design > Element > Editors > Constraints

State Definitions

State Definitions are composed in the Constraints properties of a Class element. The constraint type should be named *StateDefinition.name*, where 'name' is your choice of title for the definition. These titles are listed in the combo box of the State Analyzer whenever a Class is selected. You select a single definition from this combo box prior to running the program. The State Definition in our example is named 'StateDefinition.Location'. It defines states based on the location of instances of the CTrain Class.

State Definitions are composed of one or more specifications. Each state specification begins with the keyword 'statedef' which is then followed by one or more statements. Statements define the constraints that describe the state, and optionally a variable whose value can be used to name the state. Statements are enclosed in curly brackets and are terminated with a semi colon as shown:

```
statedef {
    Location=0;
    Departing.Name;
}
```

Naming states using variables

In this example, 'Location' is a constant and 'Departing.name' is a variable. An additional statement

follows the constraints and instructs the name of the State to be assigned from the variable value. Here is the definition with the naming directive.

```
statedef {
    Location=0;
    Departing.Name;
}
name=Departing.Name;
```

Naming states using literals

In this example the State Definition only contains constants and the state is named using a literal.

```
statedef {
   Location=100;
}
name='Central';
```

A single State Definition defining multiple State specifications.

```
statedef {
   Passengers > 100;
}
```

```
name=Busy;
statedef {
   Passengers >= 50;
}
name=Quiet;
statedef {
   Passengers < 50;
}
name=Very Quiet;
statedef {
   Passengers = 0;
}
name=Idle;
```

Default State

A State definition can specify a default 'catch all' state that will describe the state of an instance when no other state holds true. You define a default state for the definition with a statement resembling this:

```
statedef {
     Location=0;
     Departing.Name;
}
```

name=Departing.Name;
default=Moving;

In this example, while execution is in progress any instance detected having a non-zero 'Location' attribute will be recorded as being in the 'Moving' state.

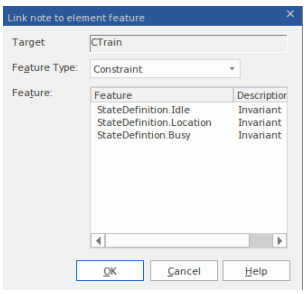
You can choose to exclude the recording of the default state by disabling the 'Include default state' option on the drop down menu of the State Analyzer toolbar. This would exclude transitions to any 'default' state being recorded.

Creating Notes on a Class element that display State Definitions

This section describes how to create the Class diagram that shows all the State Definitions defined for the Class.

Actions

Display a Class diagram	Open an existing Class diagram or create a new one.
Create a link to the Class	Drag the Class of interest on to the diagram as a link.

element	
Create a note element	Create a note element on the diagram and link it to the class.
Link the note to the State Definition	Select the link between the Note and the Class and, using its context menu, select the 'Link Note to Element Feature' option.
Choose the definition to display on the Note 	From the element dialog, choose 'Constraints' from the drop combo. Any defined State Definitions will be listed for you to choose from.
Repeat	Repeat the procedure for any other State Definitions on the class.

Synchronization

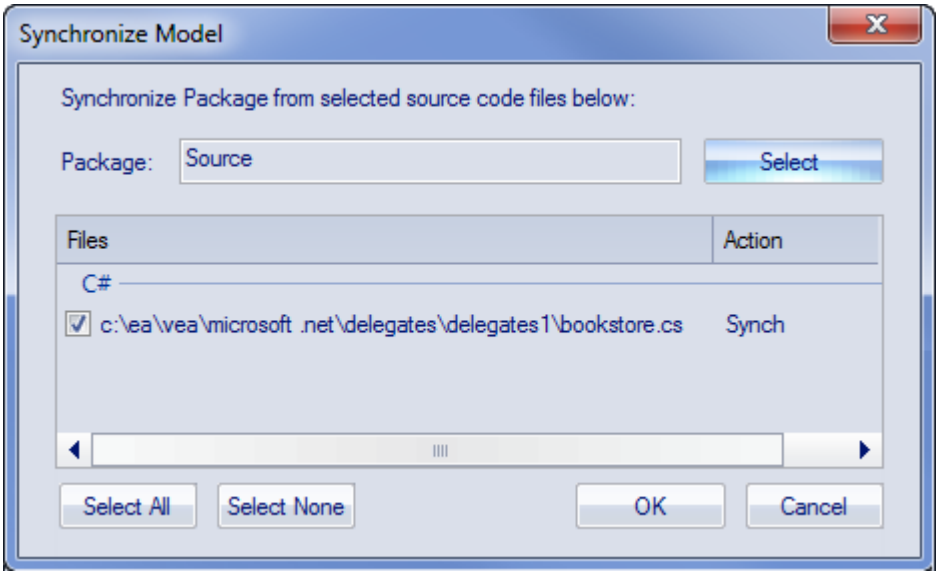
The recording produces a number of assets, the recording history being the main one. Recording also identifies a set of source code files. This set can be used to produce Class and Test Domain diagrams, but can also be used to synchronize your model.

A synchronized model provides quick and accurate navigation between diagram elements and the Class model.

Access

Ribbon	Execute > Tools > Recorder > Open Recorder > Toolbar  button
Context Menu	Right-click on the Record & Analyze window | Synchronize Model with Source Code

Synchronize Model



Field/Button	Action
Package	Click on the Select button and select the target Package into which to reverse-engineer the code files.
Files/Action	Lists the files identified during one or more recording(s). The appropriate action is listed next to each file.
Select All	Click on this button to select the checkbox against every file in the 'Files' list.
Select None	Click on this button to clear the checkbox against every file in the 'Files' list.
OK	Click on this button to start the operation. The progress of the synchronization will be displayed.

Cancel	Click on this button to abort synchronization and close the dialog.