DevTest Solutions

Using CA Continuous Application Insight

Version 8.0

This Documentation, which includes embedded help systems and electronically distributed materials, (hereinafter referred to as the “Documentation”) is for your informational purposes only and is subject to change or withdrawal by CA at any time.

This Documentation may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA. This Documentation is confidential and proprietary information of CA and may not be disclosed by you or used for any purpose other than as may be permitted in (i) a separate agreement between you and CA governing your use of the CA software to which the Documentation relates; or (ii) a separate confidentiality agreement between you and CA.

Notwithstanding the foregoing, if you are a licensed user of the software product(s) addressed in the Documentation, you may print or otherwise make available a reasonable number of copies of the Documentation for internal use by you and your employees in connection with that software, provided that all CA copyright notices and legends are affixed to each reproduced copy.

The right to print or otherwise make available copies of the Documentation is limited to the period during which the applicable license for such software remains in full force and effect. Should the license terminate for any reason, it is your responsibility to certify in writing to CA that all copies and partial copies of the Documentation have been returned to CA or destroyed.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS DOCUMENTATION “AS IS” WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NONINFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS DOCUMENTATION, INCLUDING WITHOUT LIMITATION, LOST PROFITS, LOST INVESTMENT, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED IN ADVANCE OF THE POSSIBILITY OF SUCH LOSS OR DAMAGE.

The use of any software product referenced in the Documentation is governed by the applicable license agreement and such license agreement is not modified in any way by the terms of this notice.

The manufacturer of this Documentation is CA.

Provided with “Restricted Rights.” Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7014(b)(3), as applicable, or their successors.

Contact CA Technologies

Contact CA Support

For your convenience, CA Technologies provides one site where you can access the information that you need for your Home Office, Small Business, and Enterprise CA Technologies products. At http://ca.com/support, you can access the following resources:

- Online and telephone contact information for technical assistance and customer services
- Information about user communities and forums
- Product and documentation downloads
- CA Support policies and guidelines
- Other helpful resources appropriate for your product

Providing Feedback About Product Documentation

If you have comments or questions about CA Technologies product documentation, you can send a message to techpubs@ca.com.

To provide feedback about CA Technologies product documentation, complete our short customer survey which is available on the CA Support website at http://ca.com/docs.

Contents

Chapter 1: Getting Started with CA Continuous Application Insight

To view a tutorial, see CA Continuous Application Insight Tutorial in *Getting Started*.

This section contains the following topics:

CA Continuous Application Insight Overview

CA Continuous Application Insight (CAI) is designed to enhance the process of delivering applications to market.

The following roles are the intended users of CAI:

- Architects
- Developers
- Quality assurance engineers
- Data architects
- Release engineers

CAI captures the data that flows through an application. CAI uses this data to generate paths that can be viewed in the web-based DevTest Portal. From the DevTest Portal, users can perform the following tasks:

- Drill down into business transactions and analyze abnormal behavior
- Generate architecture diagrams for cross-functional team collaboration
- Generate baseline test cases for regression testing
- Generate virtual services
- Isolate a defective component and generate the test cases and virtual services for reproducing the defect

Users can also create defect tickets from the user interface of an application. The tickets can be managed in the DevTest Portal.

CA Continuous Application Insight Prerequisites

There are no specific system requirements for CAI. Review the DevTest Server requirements.

For information about the system requirements, see *Installing*.

You can use the following methods to add transactions to the CA Continuous Application Insight database:

- Install and configure the DevTest Java Agent for a Java-based application.
- Run the CAI Agent Light.
- Run Native MQ CAI.

For detailed information about the DevTest Java Agent, including the role of the broker, see *Agents*.

In the LISA Bank application, the DevTest Java Agent is installed by default.

Once you add transactions to the database, you can view and analyze them in the DevTest Portal.

CA Continuous Application Insight Supported Platforms

CA Continuous Application Insight (CAI) is supported on the following platforms:

- IBM WebSphere Application Server 7.0
- IBM WebSphere Application Server 8.5
- JBoss 4.2
- JBoss 7
- Jetty 8
- Jetty 9.*x*
- Oracle WebLogic Server 10.3
- Oracle WebLogic Server 12.1.1
- TIBCO BusinessWorks 5.*x*
- TIBCO Enterprise Message Service 6.3
- webMethods Integration Server 9.0
- webMethods Integration Server 9.5

The following JDBC drivers and databases are supported for JDBC virtualization (see page 160):

- Apache Derby
  - Apache Derby 4.2.3 JDBC driver to Apache Derby database
- Oracle
  - Oracle 11*g* JDBC driver to Oracle 11*g* database
  - Oracle 11*g* JDBC driver to Oracle 12*c* database
- IBM DB2:
  - IBM DB2 9.5 JDBC driver to IBM DB2 9.5 database
  - IBM DB2 9.5 JDBC driver to IBM DB2 9.8 database
  - IBM DB2 9.5 JDBC driver to IBM DB2 10.5 database
- Microsoft SQL Server
  - Microsoft JDBC Driver 4.0 to Microsoft SQL Server 2008 R2 database
  - Microsoft JDBC Driver 4.0 to Microsoft SQL Server 2012 database
  - jTDS 1.3.1 JDBC driver to Microsoft SQL Server 2008 R2 database
  - jTDS 1.3.1 JDBC driver to Microsoft SQL Server 2012 database

Open the DevTest Portal

You open the DevTest Portal from a web browser.

Note: For information about the server components that must be running, see Start the DevTest Processes or Services in *Installing*.

Follow these steps:

1. Complete one of the following actions:
   - Enter **http://localhost:1507/devtest** in a web browser. If the registry is on a remote computer, replace **localhost** with the name or IP address of the computer.
   - Select View, DevTest Portal from DevTest Workstation.
2. Enter your user name and password.
3. Click Log in.

Business Transactions and Transaction Frames

Business transactions and transaction frames are key concepts in CA Continuous Application Insight. These terms are also referred to in the documentation as transactions, paths, and frames.

A *business transaction* is a representation of how a user request is serviced by an application. Each transaction encompasses a code path in which one or more servers are run as the result of a client request.

A *transaction frame* encapsulates data about a method call that the DevTest Java Agent or a CAI Agent Light intercepted. The data includes such information as:

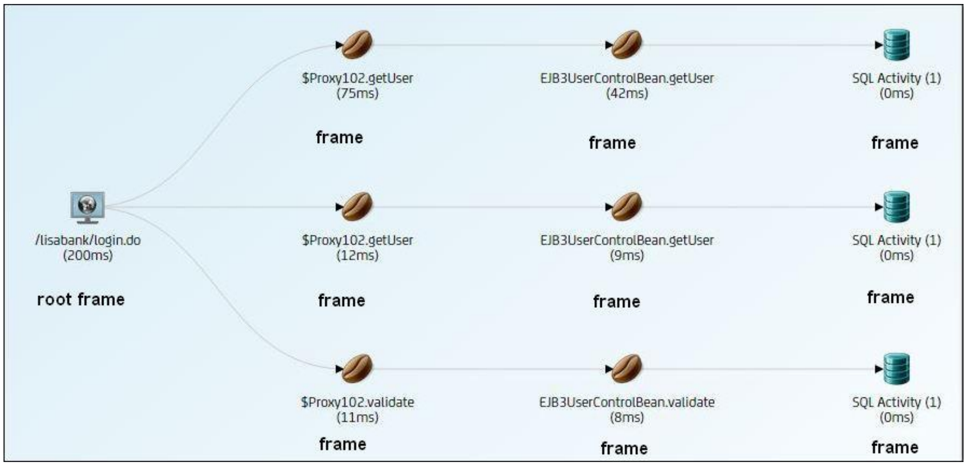
- The name of the method
- The name of the class to which the method belongs
- The arguments that were passed to the method
- The value that the method returned
- Log4j messages that are written
- Exceptions that are thrown by the Java classes

Each transaction frame has a unique identifier and a category. Typically, the category represents a protocol or operating environment. Examples include EJB, JDBC, JMS, REST, SOAP, and WebSphere MQ. A transaction frame is a node in the transaction path. Each node represents a step in the overall business transaction and the code or business logic that executed. A collection of transaction frames shows the order of the code that executed to fulfill the business transactions.

Each transaction contains a hierarchical set of transaction frames. The *root transaction frame* is the top-level frame in the hierarchy. Each transaction has a unique identifier.

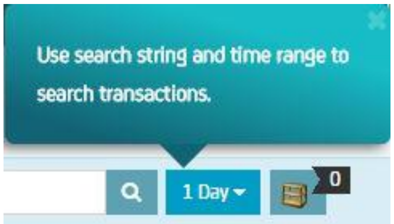
Transactions are also referred to as *paths*. A *transaction path* is a visual representation that shows the flow of how the user request is serviced. For example, the path reveals all the front-end and back-end services that a transaction contacts and the relationship between the services.

The following graphic displays a transaction path in the DevTest Portal.



Enable and Disable Help Tips

CAI provides tips on how to perform key tasks such as searching for transactions and creating artifacts. The following graphic displays tips for entering search criteria:



Use search string and time range to search transactions.

You can enable or disable the help tips.

Follow these steps:

1. Open the DevTest Portal.
2. Select admin, Preferences.

   The Preferences dialog opens.
3. To enable help, select Enable help for first time user.
4. To disable help, clear Enable help for first time user.
5. Click OK.

Disable CA Continuous Application Insight

To disable CA Continuous Application Insight, add the following line to the **local.properties** file:

```
lisa.pathfinder.on=false
```

Follow these steps:

1. Open the **local.properties** file.
2. Add the **lisa.pathfinder.on** property and set the value to false.
3. Save the **local.properties** file.

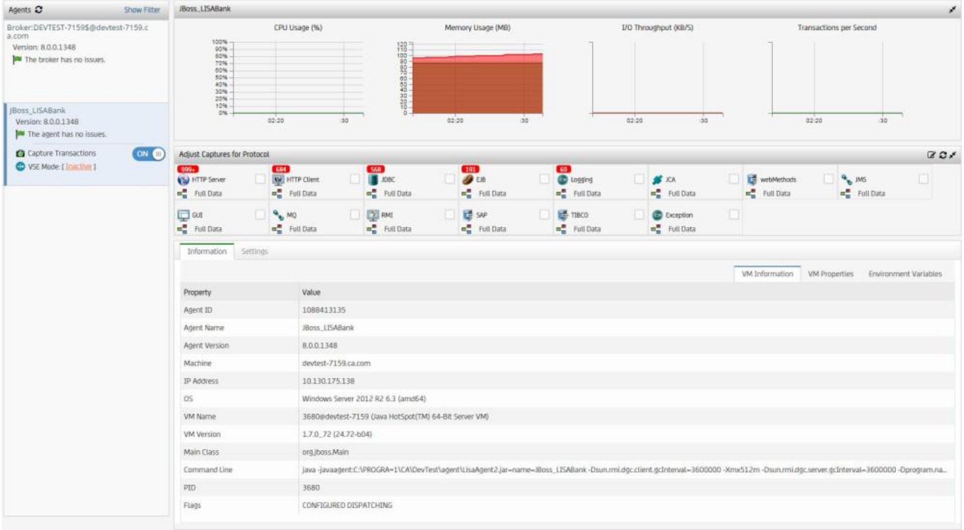
Chapter 2: Configuring Agents

The Agents window lets you perform the following tasks:

- View the agent status (see page 22)
- View the VSE status (see page 23)
- Filter agents (see page 24)
- View performance data (see page 25)
- Configure the capture levels for an agent (see page 26)
- View agent and broker information (see page 28)
- Stop and start agent capture data (see page 28)
- View and update configuration properties (see page 29)
- Upgrade an agent (see page 29)
- Delete an offline agent (see page 30)

You view the Agents window by selecting Settings, Agents from the left navigation menu of the DevTest Portal.

The following graphic displays the Agents window:



Broker:DEVTEST-7159$@devtest-7159.ca.com
Version: 8.0.0.1348
The broker has no issues.

JBoss_LISABank
Version: 8.0.0.1348
The agent has no issues.
VSE Mode: [Inactive]

Property	Value
Agent ID	1088413135
Agent Name	JBoss_LISABank
Agent Version	8.0.0.1348
Machine	devtest-7159.ca.com
IP Address	10.130.175.138
OS	Windows Server 2012 R2 6.3 (amd64)
VM Name	3680@devtest-7159 (Java HotSpot(TM) 64-Bit Server VM)
VM Version	1.7.0_72 (24.72-b04)
Main Class	org.jboss.Main
Command Line	java -javaagent:C:\PROGRA~1\CA\DevTest\agent\LisaAgent2.jar=name=JBoss_LISABank -Dsun.rmi.dgc.client.gcInterval=3600000 -Xmx512m -Dsun.rmi.dgc.server.gcInterval=3600000 -Dprogram.na...
PID	3680
Flags	CONFIGURED DISPATCHING

The Agents window contains the following components:

- Agents pane

  This pane contains a broker and one or more agents. The broker appears at the top. When you select an agent or broker, the right pane of the Agents window displays detailed information. You can delete agents and upgrade agents to the latest version from this pane.

- Performance pane

The performance details display at the top, right pane. The broker or agent that is selected in the Agents pane determines the name that appears in the title bar of pane.

- Adjust Captures for Protocols pane

  This pane contains the protocols for agent that is capturing data. This pane is not available when the broker is selected in the Agents pane.

- Information tab

  This tab contains detailed information about the broker or agent.

- Settings tab

  This tab contains the configuration properties for the broker or agent.

This section contains the following topics:

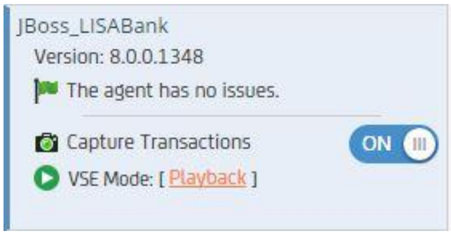
Agent Status Indicator

In the left pane of the Agents window, each agent has a flag to indicate the status information.

The following list describes the possible states:

- **Green:** The agent has no issues.
- **Yellow:** Exception thrashing or flooding.
- **Orange:** CPU or garbage collection thrashing.
- **Blink between orange and red:** Heap or perm gen exhaustion.
- **Red:** JVM alarm state or deadlock.
- **Gray:** The agent is offline.

The following graphic displays an agent with a status of no issues:



JBoss_LISABank

Version: 8.0.0.1348

The agent has no issues.

Capture Transactions ON

VSE Mode: [Playback]

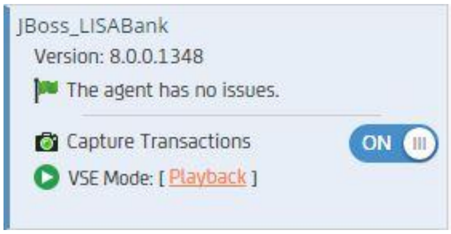
VSE Mode Indicator

In the left pane of the Agents window, each agent has an indicator for the status of the VSE functionality.

The following list describes the possible states:

- **Red with white dot:** The agent is in record mode.
- **Green with white arrow:** The agent is in playback mode.
- **Blue:** The agent is in validate mode.
- **Gray:** The agent is in passthrough mode.

The following graphic displays an agent in playback mode:



JBoss_LISABank

Version: 8.0.0.1348

The agent has no issues.

Capture Transactions

VSE Mode: [Playback]

To change the state, click the state in red text next to VSE Mode and select an option from the menu.

Filter Agents

The Agents window lets you find an agent by narrowing down the list of agents that display in the left pane.

You can sort and filter agents by name and can enter the search text criteria.

Follow these steps:

1. Select Settings, Agents from the left navigation menu.

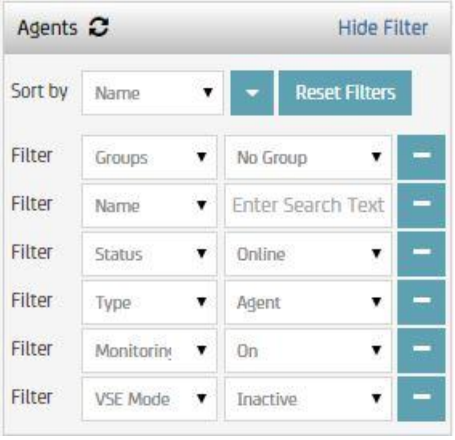
   The Agents window opens with the Agents pane at the left of the window. By default, the filter is hidden.

2. Click Show Filter.
3. To sort agents by name, select an option from the Sort by, Name list.

   The default type is Name.

4. To change the filter type, click the arrow and select from list of options.
5. To filter by name, select an option from the Filter, Name list.
6. Use the up and down arrows to list the agents in ascending or descending order in the left pane.
7. To search agents by name, enter the search criteria in the Enter Search Text field.
8. (Optional) To add more filters, click +.

   An additional filter displays:



Sort by Name

Filter Groups No Group

Filter Name Enter Search Text

Filter Status Online

Filter Type Agent

Filter Monitoring On

Filter VSE Mode Inactive

9. To remove a filter, click -.
10. To hide the filter, click Hide Filter.
11. To reset the filters, click Reset Filters.

View Performance Data

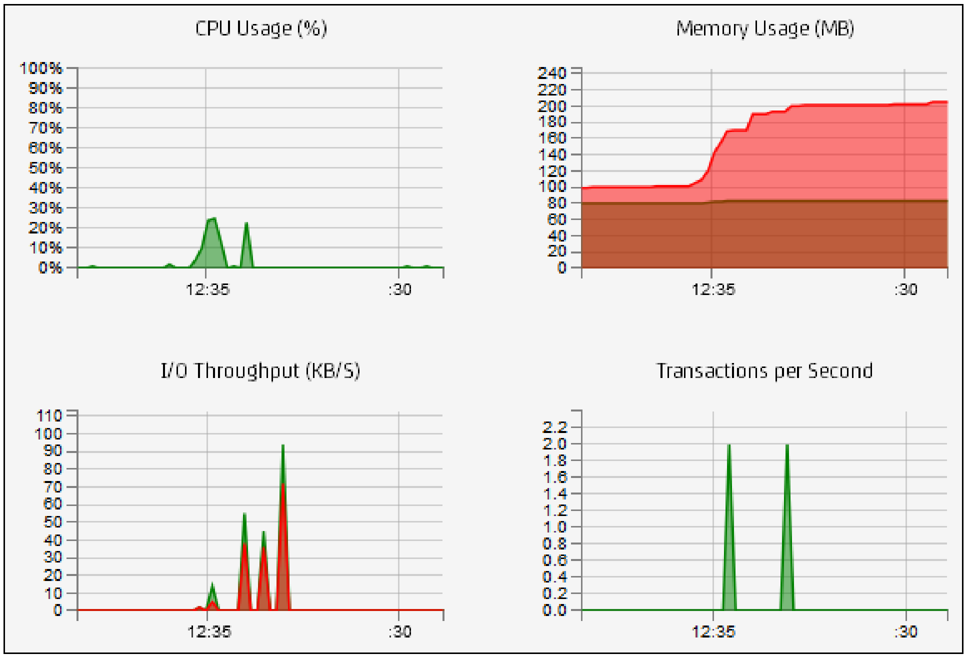
You can view a set of performance graphs for an agent or broker.

The graphs display the following information:

- CPU usage
- Memory usage
- I/O throughput
- Transactions per second

The I/O throughput graph is not applicable to the broker.

The following graphic displays the performance graphs for an agent.



Follow these steps:

1. Select Settings, Agents in the left navigation menu.

   The Agents window opens.

2. In the Agents pane, select an agent or broker.

   The performance data displays in the right pane.

3. (Optional) Click a performance graph to view a legend.

Configure Capture Levels

Each protocol that the DevTest Java Agent can capture has a *capture level*.

The following capture levels are available:

Counts

The agent collects only count information. This level has the least impact on the system under test.

Counts and Paths

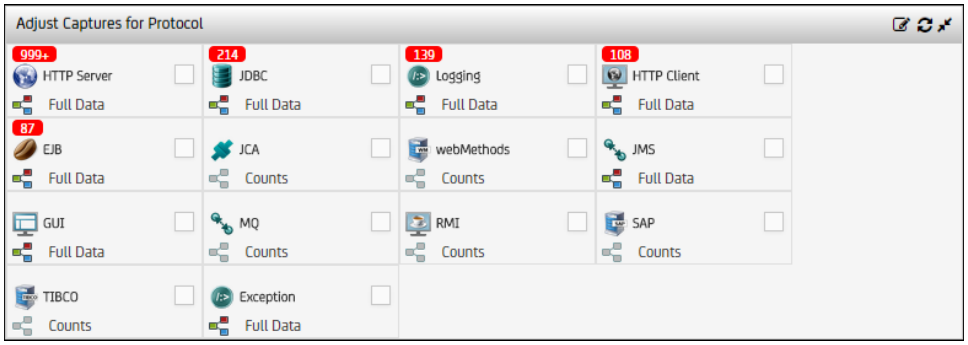
The agent collects transaction frames, but not full requests or responses.

Full Data

The agent collects all data. This level has the most impact on the system under test.

The default capture level is Counts.

The following graphic shows the Adjust Captures for Protocol pane, which lists the protocols and their capture levels.



Adjust Captures for Protocol

999+ HTTP Server □ Full Data	214 JDBC □ Full Data	139 Logging □ Full Data	108 HTTP Client □ Full Data
87 EJB □ Full Data	JCA □ Counts	webMethods □ Counts	JMS □ Full Data
GUI □ Full Data	MQ □ Counts	RMI □ Counts	SAP □ Counts
TIBCO □ Counts	Exception □ Full Data		

We recommend the following workflow:

1. Start with all the protocols in Counts mode.
2. Exercise the application and view the number of counts.
3. Decide which protocols you are interested in and increase the capture levels accordingly.

If you want to create tickets (see page 61), ensure that the capture level for the HTTP Server protocol is set to Full Data.

If you want to create baselines (see page 81) or virtual services (see page 129), ensure that the capture levels for the appropriate protocols are set to Full Data.

The information is updated once a minute.

A red badge displays the number of times that the agent has observed the protocol since the agent was started.

If the count is 0, the red badge does not appear.

If the count is more than one thousand, the red badge displays the text **999+**. The tooltip for the icon shows the exact number.

Note: Setting different capture levels is not supported for queue-based client and server communication, for example, WebSphere MQ and JMS.

Follow these steps:

1. Select Settings, Agents in the left navigation menu.

   The Agents window opens.

2. From the Agents pane select an agent.
3. Select the check box for one or more protocols. Selecting more than one protocol lets you change to the same capture level for several protocols at once.
4. To select all protocols, click ✎ and select Select All.
5. Click the current capture level and select an option from the drop-down list.
6. To switch between displaying all protocols and displaying only the protocols whose count is greater than 0, click ✎ and select Show Protocols with Counts.
7. To reset all of the counts to 0, click ✎ and select Reset All Counts.
8. To refresh the counts and capture levels, click ✎ and select Enable Auto fresh.

View Agent and Broker Information

You can view the following categories of information about an agent or broker:

- VM Information

  The VM Information tab displays basic information about the agent and the virtual machine in which it is running. The values are read only.

  If a broker is selected, the agent name is the user name, followed by an ampersand, followed by the computer name.

  The main class is the Java class that contains the main method that was invoked for the agent.

  The PID is the process ID in which the agent is running.

- VM Properties

  The VM Properties tab displays the system properties and the environment variables. The values are read only.

  Note: The VM properties tab cannot display information for an agent that is offline.

- Environment Variables

  The Environment Variables tab displays the system environment variables on the computer where the agent is running.

Follow these steps:

1. Select Settings, Agents in the left navigation menu.

   The Agents window displays.

2. In the Agents pane, select an agent or broker.
3. Select the Information tab from the right pane.

Stop and Start Agent Transaction Capture

The Agents window lets you control whether an agent is capturing transactions.

Follow these steps:

1. Select Settings, Agents in the left navigation menu.

   The Agents window displays.

2. To stop the agent transaction capture, move the slider to Off for the agent you want to stop capturing data.
3. To start the agent transaction capture, move the slider to On for the agent you want to start capturing data.

View and Update Properties

You can view and update the configuration properties for an agent or broker.

The properties are grouped into categories for example, General and Capture.

Follow these steps:

1. Select Settings, Agents in the left navigation menu.

   The Agents window displays.

2. In the left portion, select an agent or broker.
3. Click the Settings tab.
4. Change the value of one or more properties.
5. To discard any changes that have not saved, click Revert.
6. To save the changes, click Save.

Upgrade an Agent in the DevTest Portal

From the Settings, Agents window, you can upgrade agents to the latest version in the DevTest Portal. The Download button appears next to the agent when a new version is available. You hover over the Download and Pending button to see the version information.

Note: Not all agents can be upgraded in the DevTest Portal. Native agents and agents using JRE 1.5 and **LisaAgent2.jar** must be manually upgraded (see page 30).

Only pure agents that have the following requirements can be upgraded:

- JRE 1.6 or later
- agent must be started from the **InsightAgent.jar** file.

Follow these steps:

1. Save the latest **LisaAgent.jar** file to the **LISA_HOME\lib\core\agentupgrade** folder.
2. Select Settings, Agents from the left navigation menu.

   The Agents window displays.

3. From the Agents pane, click Download button.

   The Download button changes to Pending text.

4. Restart the agent to complete the upgrade process.

Upgrade an Agent Manually

If an agent is not eligible to be upgraded in the DevTest Portal (see page 29), you can use this procedure to upgrade the agent.

Follow these steps:

1. Stop the agent.
2. Replace the existing **LisaAgent.jar** file with the new version.
3. Start the agent.

Delete an Offline Agent

You can delete an offline agent from the Agents pane of the Agents window. When an agent is deleted, all transactions and frames of the agent are removed.

Note: The Delete button displays when an agent is offline.

Follow these steps:

1. Select Settings, Agents from the left navigation menu.

   The Agents window displays.
2. From the Agents pane, locate the agent that you want to delete.
3. (Optional) Click Show Filter to use the filter criteria to search for an agent.
4. Click Delete.

Chapter 3: Analyzing Business Transactions

CAI lets you view the business transactions (see page 15) that have been captured. These transactions are also referred to as *paths*.

CAI exposes information about the transactions and their frames. This information includes execution times, thread utilization, log messages, SQL statements, request and response data, and stack traces.

You view transactions from the Analyze Transactions window. Transactions display in graphical or list view. By default, transactions are displayed in list view.

Note: Frames for each transaction follow a specific execution order and are not arranged by a particular agent.

Graphical view is a visual representation of the transactions. This view displays up to a maximum of 100 transactions. You can do the following operations in this view:

- Search for paths by full text or by a duration of time
- View the transaction details by clicking the frame. The transaction details display a visual path of transactions and tabs that contain the details of the frame. For example, frame information, tags, request, response, thread name, log messages, XML, and exception when applicable.
- View paths for transactions that have repeated paths
- Filter paths by category
- Shelve and unshelve a path
- Export a path diagram to a PDF file
- Show and hide an outline of a path
- Disable capture

In list view, transactions are listed in the Results pane. You can do the following operations in this view:

- Search for transactions by full text or by a duration of time
- View information of a path. For example, the path and agent name, start time, wall time, and CPU time
- Import and export transaction capability
- View the transaction details by clicking ▤ in the Actions column. The transaction details display a visual path of transactions in the path graph and tabs that contain the details of the frame. For example, frame information, tags, request, response, thread name, log messages, XML, and exception when applicable.
- Import and export transaction capability
- View paths for transactions that have repeated paths

Follow these steps:

1. Select Application Insight, Analyze Transactions from the left navigation menu.

   The Analyze Transactions window displays. By default transactions display in list view.

2. To open graphical view, click .

   The transactions display in graphical view.

3. (Optional) Click Show Outline to view a portion of a large transaction at a time.

This section contains the following topics:

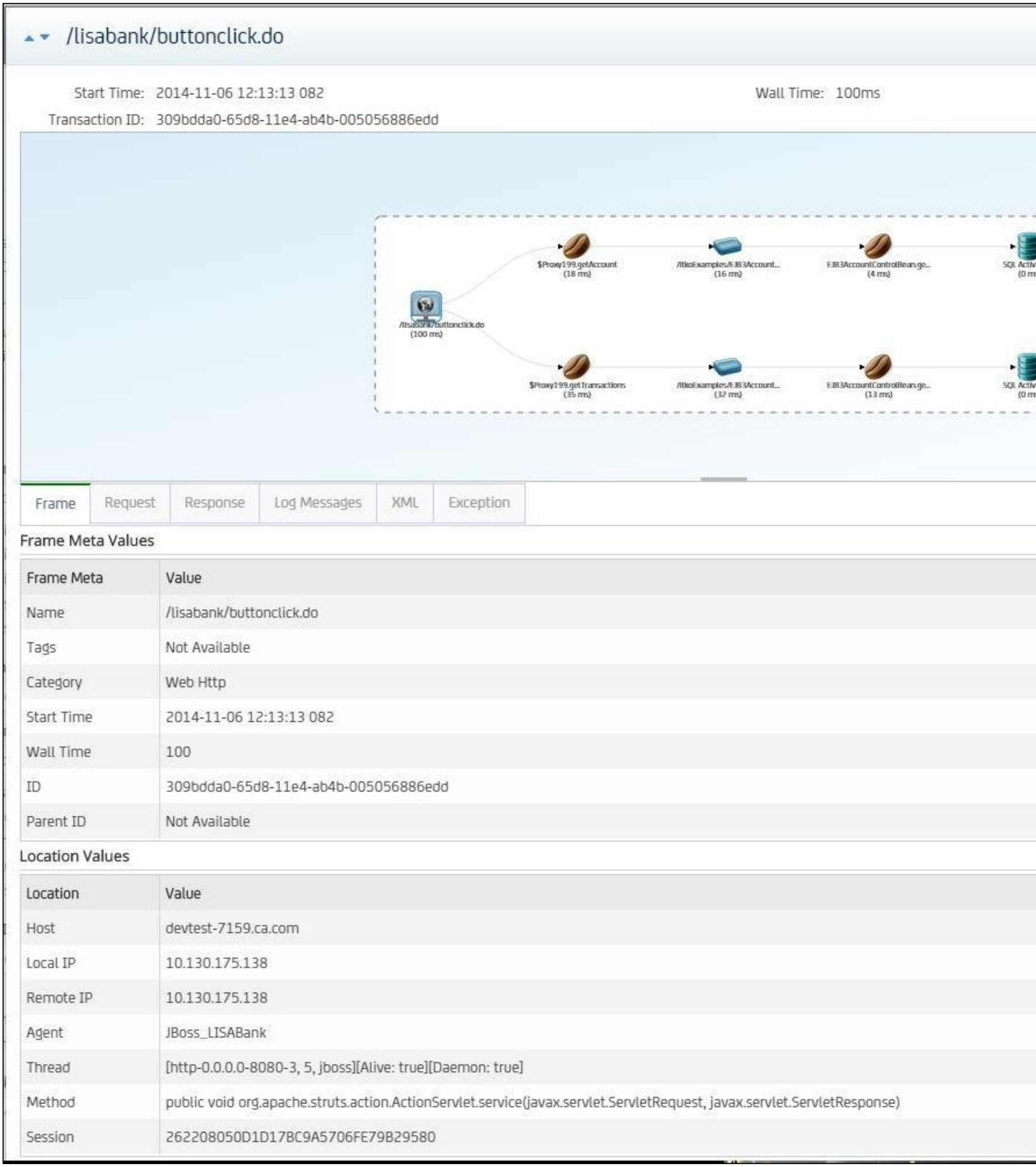
View Transaction Details (see page 34)
Export Path Diagram to PDF (see page 43)
Merge Repeated Paths (see page 44)
Exclude Data from Agent Capture (see page 45)
Search and Filter Transactions (see page 45)
Exporting and Importing Paths (see page 48)

View Transaction Details

The transaction detail dialog contains information about each transaction in a path. The dialog is divided into the following areas:

- The upper area contains the path graph (see page 38).
- The lower area contains the frame information (see page 40).

The following graphic shows the transaction detail dialog:



Start Time: 2014-11-06 12:13:13 082

Wall Time: 100ms

Transaction ID: 309bdda0-65d8-11e4-ab4b-005056886edd

Frame Meta Values

Frame Meta	Value
Name	/lisabank/buttonclick.do
Tags	Not Available
Category	Web Http
Start Time	2014-11-06 12:13:13 082
Wall Time	100
ID	309bdda0-65d8-11e4-ab4b-005056886edd
Parent ID	Not Available

Location Values

Location	Value
Host	devtest-7159.ca.com
Local IP	10.130.175.138
Remote IP	10.130.175.138
Agent	JBoss_LISABank
Thread	[http-0.0.0.0-8080-3, 5, jboss][Alive: true][Daemon: true]
Method	public void org.apache.struts.action.ActionServlet.service(javax.servlet.ServletRequest, javax.servlet.ServletResponse)
Session	262208050D1D17BC9A5706FE79B29580

Path Graph Information

The path graph displays the path and its frames in a graphical representation.

In the path graph, you can do the following operations:

- shelve or unshelve transactions by right-clicking the frame and selecting an option, Shelve or Shelve All Occurred Transactions
- disable capture (see page 45) by right-clicking the frame and selecting an option, Disable Capture or Disable URL Capture

Agent name

The name of the agent that generated the path.

Start time

The date and time when the frame began running.

Wall Time

The execution time for the frame.

Transaction ID

The unique identifier of the path.

Frame Information

The frame information displays the details of the selected frame in the path graph.

Frame

The frame information which includes meta and location values.

Tags

Any frame tags that the frame has.

Request

The actual request that the selected frame sent. This information does not appear for JDBC frames.

Response

The actual response that the selected frame received. This information does not appear for JDBC frames.

SQL

Information about the SQL call. This information includes the SQL statement, the results, and the output. This information appears only for JDBC frames.

Thread Name

The name of the thread on which the frame ran.

CPU time

The amount of time that the processor spent running the frame.

Utilization

The percentage of thread utilization.

Log Messages

Any log messages. By default, the minimum level of log messages is **Warning**. To change the minimum level, configure the Java logging level property.

You can configure this property from the Agents window (see page 19) of the DevTest Portal. The property appears in the Settings tab.

XML

The raw XML of the instrumentation response in the frame. This information does not appear for JDBC frames.

Exception

Displays detailed information when a transaction contains an exception.

Follow these steps:

1. Select Application Insight, Analyze Transactions from the left navigation menu.

   The Analyze Transactions window displays.

2. (Default) In list view, click ▤ in the Actions column.
3. In graphical view, double-click the parent frame.

   The transaction detail dialog opens. The title bar contains the full name of the frame.

4. Review the information in the path graph and the frame information.
5. Click the up and down arrows at the left of the title bar to navigate to the previous and next transaction.
6. Click X to close the dialog.

Path Graph

A *path graph* contains a graphical representation of a path and its frames. The path graph displays in the upper area of the transaction detail dialog.

When you select a root transaction frame (see page 15) in graphical or list view from the Analyze Transactions window, the transaction detail dialog displays with the path graph at the top.

You view the path graph from the Analyze Transactions window by:

- clicking a path in graphical view.
- clicking Open transaction details from the Actions column in list view.

The icons indicate the type of frame. Examples of the frames types are EJB, HTTP, JMS, REST, web service, DevTest, and unknown. The DevTest frame type represents a step executing inside a test case or virtual service model.

The text below the icon provides the following information:

- Frame name
- Execution time

Use Zoom In and Zoom Out to size the path display.

Frame Name

The format of the frame name depends on the type of frame.

The frame name for HTTP is the path portion of the URL. The name also contains the method. Paths that are created using the VS Traffic to CAI Companion are indicated with the virtualized label in the component name.

An example is **POST/itkoExamples/EJB3AccountControlBean(virtualized).**

See the VS Traffic to CAI Companion (see page 193) topic for more information about using this companion.

The frame name for JDBC is SQL Activity (*N*), where *N* is the number of SQL statements that were used to generate the frame.

The frame name for JMS consists of the following parts:

- A string that identifies the type of operation. If the operation involves the production of a message, the string is **send:**. If the operation involves the consumption of a message, the string is **recv:**.
- The name of the queue
- (Optional) The name of the connection factory

An example is **recv:queue/ORDERS.REQUEST@ConnectionFactory**.

The frame name for JMS can also be **DevTest**. This name is used when a test case receives a JMS message.

The frame name for RMI is the Java class and method that was executed.

The frame name for webMethods is the flow service name.

The frame name for WebSphere MQ consists of the following parts:

- The string **put** or **get**
- The WebSphere MQ host name
- The queue manager name
- The queue name

An example is **put-172.24.255.255-QueueManager-ORDERS.REQUEST**.

The frame name for a web service is the path portion of the URL.

Execution Time

In certain scenarios, the execution time below the leftmost component is the time for the entire transaction.

If the execution time is so low that it rounds down to zero, the value **0 ms** is displayed.

Frame Information

The transaction detail dialog displays information about a frame that is selected in the path graph. The frame information displays in the lower portion of the transaction detail dialog.

The frame information contains the following tabs:

- Frame
- Request
- Response
- Log Messages
- XML
- Exception
- Payload
- SQL Summary

The Frame tab and the Log Messages tab appear for all frame types.

Frame

The Frame tab shows the metadata values and the location values of the selected frame. The following details are listed:

Frame Meta Data

- Name: The name of the frame.
- Tags: Any frame tags that are associated with the frame.
- Category: The category of the frame.
- Start time: The time at which the frame began running.
- Wall time: How much time it took to run the frame from start to finish. Also known as root frame response time.
- ID: The unique frame ID
- Parent ID: The parent frame ID of the current frame.

Location Values

- Host: The name of the computer on which the frame ran.
- Agent: The name of the agent which captured the frame.
- Thread name: The thread name that the frame is running.
- Method: The java methods captured in the frame
- Session: Session ID of the session this frame is associated with.

If the execution time for a frame is so low that it rounds down to zero, the value **0 ms** is displayed.

Request Tab

The Request tab displays the actual request that the selected frame sent. This tab appears only for non-JMS and non-JDBC frames.

If a WebSphere MQ transaction includes a binary payload, the Request tab displays a human-readable version of the payload. The nontext characters are removed. You can view the original binary payload in the XML tab.

Response Tab

The Response tab displays the actual response that the selected frame received. This tab appears only for non-JMS and non-JDBC frames.

If a WebSphere MQ transaction includes a binary payload, the Response tab displays a human-readable version of the payload. The nontext characters are removed. You can view the original binary payload in the XML tab.

Log Messages Tab

The Log Messages tab contains any log messages. The following details are listed:

- Level of information
- The logger name (if any)
- The text of the log message

By default, the minimum level of log messages is **Warning**. To change the minimum level, configure the **Java logging level** property.

You can configure this property from the Agents window (see page 19) of the DevTest Portal. The property appears in the Settings tab.

To sort the columns, click the drop-down arrow in a column heading and select Sort Ascending or Sort Descending. To show or hide columns, click the drop-down arrow in a column heading, point to Columns, and select or clear the check boxes.

XML Tab

The XML tab displays the raw XML of the instrumentation response in the frame. This tab appears for all frames other than JDBC frames.

The XML tab is additive. The frames to the right contain less information than the frames on the left. The first frame contains the complete response.

Exception Tab

If the frame contains an exception, the Exception tab displays the stack trace.

This feature has the following prerequisites:

- The capture level (see page 26) of the Exception protocol must be set to Full Data for the agent.
- The Capture all the supported exception frames property must be enabled for the agent. To access this property in the Settings tab, select the Transactions category.

Payload Tab

The Payload tab displays the JMS payload. This tab appears only for JMS frames.

SQL Summary Tab

The SQL Summary tab provides a summary of the SQL information. This tab appears only for JDBC frames.

The following details are listed:

- A sequential identifier that is assigned to each SQL statement
- SQL statement
- Number of rows that were returned
- Number of invocations
- Average time (milliseconds)
- Total time (milliseconds)

Clicking the SQL statement displays the results.

Double-clicking the SQL statement opens a dialog that displays the entire statement.

To sort the columns, click the drop-down arrow in a column heading and select Sort Ascending or Sort Descending. To show or hide columns, click the drop-down arrow in a column heading, point to Columns, and select or clear the check boxes.

Export Path Diagram to PDF

You can export a path diagram into a PDF file so that you can view, print, and save the path information. You export a path into a PDF file from list or graphical view.

Follow these steps:

In list view:

1. Select Application Insight, Analyze Transactions in the left navigation menu.
2. Click ☰ in the Actions column.

   The transaction detail dialog displays.

3. Click PDF.

   The path diagram displays in the preview browser.

4. Move the mouse to the bottom, right of the browser to display the task bar.

   The task bar contains options to zoom, save, and print.

5. To save the PDF file, click Save.
6. Enter the file name and click Save.
7. To print the PDF, click the print button.

   The print setup displays.

8. Click Print.

In graphical view:

1. In graphical view, click PDF.

   The path diagram displays in the preview browser.

2. Move the mouse to the bottom, right of the browser to display the task bar.
3. To save the PDF file, click Save.
4. Enter the file name and click Save.
5. To print the PDF, click the print button.

   The print setup displays in the browser.

6. Click Print.

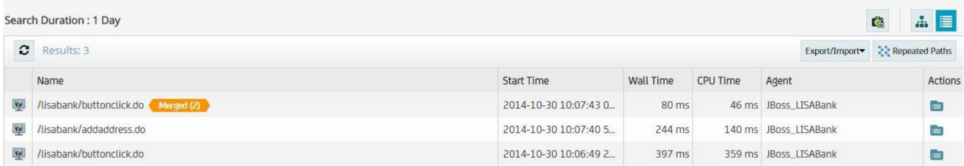
Merge Repeated Paths

A repeated path contains duplicate transactions that occur in a path. You can merge these paths to identify the same transactions and reduce the number of transactions that are listed in the Results pane. Merging repeated paths helps you to identify the cause of the defect. For example, you can see where all of the occurred, repeated login transactions that fail for an application.

You can merge repeated paths in the list or graphical view.

In graphical and list view, repeated paths are indicated with an orange badge. The number in the badge is the total number of paths that are repeated for the transaction.

The following graphic displays the list view for a transaction that has two merged, repeated paths:

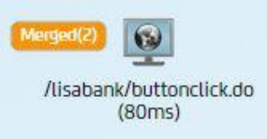


Search Duration : 1 Day

Results: 3

	Name	Start Time	Wall Time	CPU Time	Agent	Actions
	/lisabank/buttonclick.do Merged (2)	2014-10-30 10:07:43 0...	80 ms	46 ms	JBoss_LISABank	
	/lisabank/addaddress.do	2014-10-30 10:07:40 5...	244 ms	140 ms	JBoss_LISABank	
	/lisabank/buttonclick.do	2014-10-30 10:06:49 2...	397 ms	359 ms	JBoss_LISABank	

The following graphic displays the graphical view for a transaction with two merged, repeated paths:



Follow these steps:

1. Select Application Insight, Analyze Transactions from the left navigation menu.

   The Analyze Transactions window opens.

2. In list or graphical view, click Repeated Paths.

   The repeated paths merge into one transaction. An orange badge displays with a count of the repeated paths.

3. To unmerge the transactions, click Repeated Paths.

Exclude Data from Agent Capture

The DevTest Java Agent can capture a large amount of data, including data that is not relevant or valuable. For example, you might not be interested in the data about logging in to an application.

You can specify that the agent no longer capture a certain type of transaction frame.

Some frames do not support being excluded. If you try to exclude a frame that is not supported, the following message appears:

```
Failed to create disable capture rule for {{name}}. {{reason}}
```

Follow these steps:

1. Select Application Insight, Analyze Transactions from the left navigation menu.

   The Analyze Transactions window opens.

2. Do one of the following actions:

   - Display the graphical view.
   - Display the list view and click Open transaction details from the Actions column.

3. Right-click the frame that you want to exclude and select Disable Capture or Disable URL Capture.

4. To view the frames that are currently excluded, click the Manage exclude frame capture rules button.

5. To remove a frame from the list of excluded frames, do the following actions:

   a. Click the Manage exclude frame capture rules button.

   b. Select the frame.

   c. Click Include.

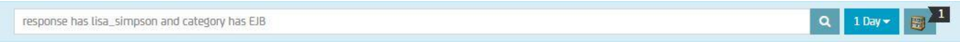
Search and Filter Transactions

You can search and filter transactions by entering a full payload text search (see page 46), restricting the period of time (see page 46), and refining the search using filter options (see page 47).

Search for Transactions

CAI lets you find transactions using an intelligent search capability. You enter a payload text search from the Enter Search Text field. You can enter a payload search string for example, **response has lisa_simpson and category has EJB**.

You can also use keywords for example, a transaction ID, a category like EJB, and a local or remote IP address.



response has lisa_simpson and category has EJB

Follow these steps:

1. Select Application Insight, Analyze Transactions in the left navigation menu.
2. Enter the text search criteria in the search text field and click 🔍.

   The Results list returns a list of paths that are based on the search criteria.
3. (Optional) Refine your search further by selecting options in the Refine by pane.

Search for Transactions by Time

The Analyze Transactions window can contain hundreds of transaction paths. You can narrow down the paths that you want to analyze by filtering the transactions by time and date. The results of the search are listed in the Results pane. You can view up to 25 transactions at a time in the list.

You specify the following time-range by:

- Minutes (1 through 30)
- Hours (1 through 12)
- Days (1 through 15)
- All Time

You specify the following date-range criteria in the Advanced time option:

- Start date and time
- End date and time

The default date range is 1 Day.

Filter Transactions

You use the filter options in the Refine By pane to narrow down the number of transactions that appear in the Results list.

You enter a filter option from the search text field or select the check boxes. For example, if you know the class name you want to filter, enter the name and press enter. The filter automatically selects the specific check box for that class name. The Results pane returns a list of transactions containing that class name.

The following filter options are available:

- Agent
- Category
- Class Name
- Execution Time
- Local IP
- Operation
- Point of Interest
- Remote IP
- Tags
- Transaction ID

Follow these steps:

1. Select Application Insight, Analyze Transactions from the left navigation menu.

   The Analyze Transactions window displays in list view by default.

2. In the Refine By pane, select the appropriate check box or enter text.

   The Search Duration pane returns a list of transactions for the specified filter option.

Filter Graph by Category

You can filter transactions that have a frame in a specific category from the graphical view. The options that display in the category list depend on which categories are supported for that transaction.

Follow these steps:

1. Select Application Insight, Analyze Transactions in the left navigation menu.

   The Analyze Transactions window displays in list view by default.

2. Click from the top of the Search Duration pane.

3. Click and select or clear the appropriate category check box.

   The Results pane returns a list of transactions for the specified category.

Exporting and Importing Paths

You can export one or more paths from the CAI database using the Analyze Transactions window. The paths can be imported into other CAI installations or can be imported from Splunk. You can export paths to be debugged by CA Support.

Export Paths

You can export transaction paths that were captured into a ZIP file.

Follow these steps:

1. Select Application Insight, Analyze Transactions in the left navigation menu.
2. From the Results pane, select one or more paths that you want to export.
3. Click Export/Import and select one of the options:
   - Export Selected - to export the selected paths in the Results list.
   - Export Page - to export all the transactions in the Results list.

   The Export Path wizard opens.
4. Enter a name for the export zip file.
5. Click Export and download path.

   A ZIP file is created and is downloaded.

Import Paths

Transaction paths can be imported using the Analyze Transctions window.

Follow these steps:

1. Select Application Insight, Analyze Transactions in the left navigation menu.
2. Click Export/Import and select Import.

   The Import Path wizard opens.
3. Click Choose File, select the zip file, and click Open.
4. Click Import.

   The Imported column displays:

   - Yes when the path is imported into the database.
   - Duplicate when the path already exists in the database.
   - Failed when the paths were not imported.
5. Click OK.

   A Confirmation dialog displays.
6. Click OK and click Refresh in the Results pane.

   The imported paths display in the Results list.

Import Paths from Splunk

From the Analyze Transactions window, you can import paths that were captured from Splunk. See the Agent Light (see page 195) topic for more information about using Splunk and the CAI Agent Light. The Search field in the Import from Splunk dialog is the agent light built-in Splunk search string. You can enter different search strings that generate the Splunk HTTP transactions. See the Agent Light for HTTP (see page 200) topic for more information about the HTTP search query language syntax.

Follow these steps:

1. Select Application Insight, Analyze Transactions in the left navigation menu.

   The Analyze Transactions window displays.

2. Click Export/Import and select Import from Splunk.

   The Import from Splunk dialog opens.

3. Enter the fields and click Import.

   A confirmation dialog indicates that the import was successful.

4. Click OK to close the dialog.

5. Click Refresh ⟳ from the Results pane to display the Splunk paths.

   The imported paths from Splunk display at the top of the list in the Results pane.

Chapter 4: Using the Shelf

CAI allows you to add a collection of transaction frames into the shelf to generate artifacts. Adding a transaction frame to the shelf is known as *shelving.* From the shelf, you generate artifacts by creating virtual services, baselines, and documentation on multiple transaction frames instead of working with one frame at a time.

Note: When generate an artifact, ensure you delete any transaction frames that you do not want to include. For more information about deleting transactions, see the Delete Frames in the Shelf (see page 56) topic.

Transactions can be shelved from:

- Analyze Transactions window in graphical view
- Transactions detail dialog (see page 34)

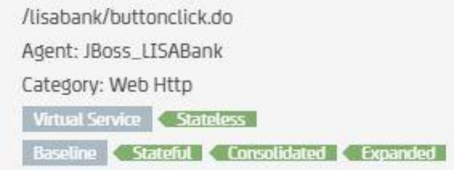
See the Add a Transaction Frame to the Shelf (see page 54) procedure for more information about how to add transactions to the shelf.

The shelf icon is at the top of the Search Duration bar in the Analyze Transactions window. The number that is located in the right corner of the icon indicates the total number of shelved frames.

You can generate the following artifacts:

- Create VS
- Create a Baseline
- Create Document

The types of artifacts that can be generated for each transaction are indicated with colored labels. The following graphic displays an example of the types of artifacts that can be generated for a transaction. From this transaction, you can create a stateless virtual service and a stateful, consolidated, or expanded baseline.



/lisabank/buttonclick.do

Agent: JBoss_LISABank

Category: Web Http

Virtual Service | Stateless

Baseline | Stateful | Consolidated | Expanded

The shelf has two methods of generating artifacts:

- One-click method

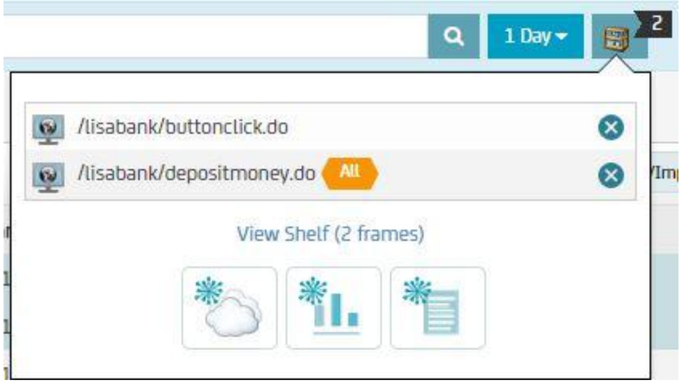
  This method provides a quick way to create baselines, virtual services, and documentation on transactions that are in the shelf without opening the shelf dialog.

- Shelf dialog

  This method allows you to access the shelf dialog to search for shelved transactions and create baselines, virtual services, and documentation.

One-click method

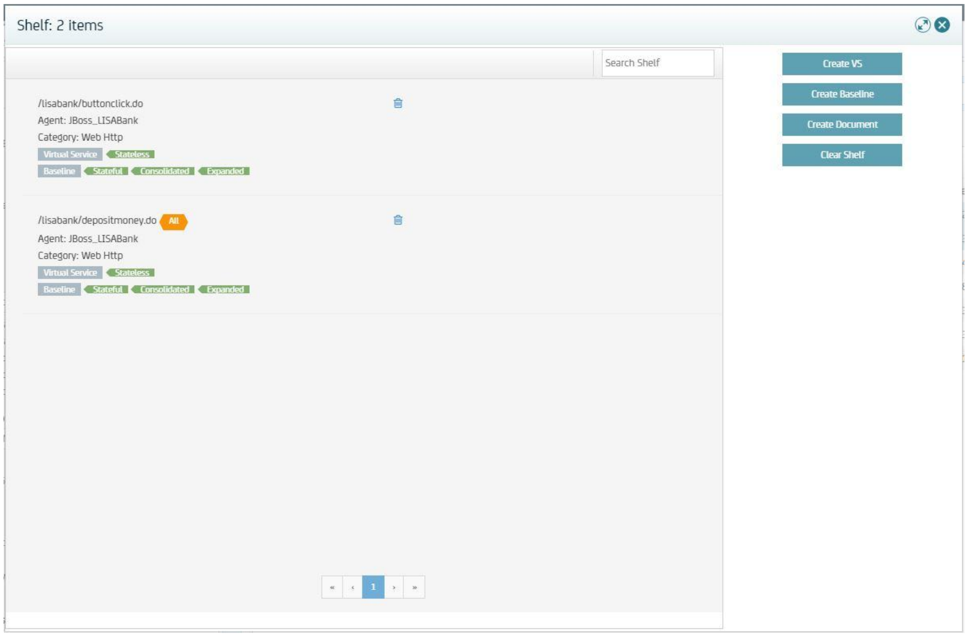
The shelf provides a one-click method of generating artifacts. By hovering over the shelf icon , you can view the shelved transactions and can generate artifacts without opening the shelf dialog. The following graphic displays the one-click popup:



A maximum of five transactions display at a time. When you delete a transaction, the next transaction in the shelf displays.

Shelf dialog

To open the shelf dialog, click the shelf icon or click View Shelf (X frames) from the one-click method. The following graphic displays the shelf dialog:



Shelf: 2 items

/lisabank/buttonclick.do
Agent: JBoss_LISABank
Category: Web Http

/lisabank/depositmoney.do
Agent: JBoss_LISABank
Category: Web Http

From the shelf dialog, you can perform the following operations on the transactions that are listed:

- Search for transactions (see page 57)
- Delete a transaction frame (see page 56)
- Create artifacts: virtual service, baseline, and documentation (see page 58)
- Clear the entire contents of the shelf (see page 59)

Note: For more information about creating virtual services and baselines, see the Creating Virtual Services (see page 129) and Creating Baselines (see page 81) topics.

This section contains the following topics:

How to Work with Transactions in the Shelf (see page 54)

How to Work with Transactions in the Shelf

This scenario describes how to put a collection of transaction frames in the shelf to generate artifacts.

1. Add a Transaction Frame to the Shelf (see page 54).
2. Add All Occurred Transactions to the Shelf (see page 55).
3. Add Merged Transactions to the Shelf (see page 55).
4. Delete Frames in the Shelf (see page 56).
5. Search for Paths in the Shelf (see page 57).
6. Generate Artifacts (see page 58).
7. Clear All Transactions from the Shelf (see page 59).

Add a Transaction Frame to the Shelf

You shelve transactions and frames into the shelf to create virtual services, baselines, and documentation.

Follow these steps:

1. Select Application Insight, Analyze Transactions from the left navigation menu.

   Analyze Transactions window displays captured transactions in list view by default.

2. In list view, click in the Actions column.

3. In graphical view, click .

   The transactions display in the path graph.

4. Right-click a root transaction or frame and select Shelve.

   In graphical and list view, a blue pin icon displays above the shelved transaction frame. The frame is added to the shelf. The total number of shelved transactions in the shelf increases by one.

Add All Occurred Transactions to the Shelf

You can shelve all the transactions and frames that occurred during a captured transaction into the shelf instead of shelving them separately. The shelf displays all the occurred transactions as one transaction. You create virtual services, create baselines, and create documentation as a single shelved item. For example, when you shelve all the transactions occurring in the same path at the root transaction level then all the frames are added to the shelf as one transaction.

Follow these steps:

1. Select Application Insight, Analyze Transactions from the left navigation menu.

   Analyze Transactions window displays in list view by default.

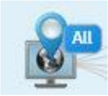
2. Click ▤ in the Actions column.
3. The transaction details displays with the path graph at the top.
4. In graphical view, click 🖧.

   The transactions display in the path graph.

5. Right-click a transaction and select Shelve All Occurred Transactions.

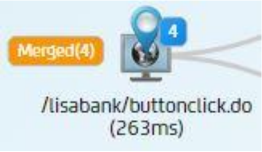
   In graphical view and in the path graph, a blue pin and the word **All** displays above the shelved transaction frame. The frame is added to the shelf. The total number of shelved transactions in the shelf increases by one.

   The following graphics displays a transaction that has been shelved with all the occurred transactions in the path:



Add Merged Transactions to the Shelf

You can shelve merged transactions with repeated paths. Adding a group of repeated paths into the shelf lets you create artifacts for several transactions at once instead of one transaction at a time. When merged transactions are shelved, the number of merged repeated paths display in a blue badge.



Note: Stateful baselines are not supported for merged transactions. You can create consolidated and expanded baselines only.

Follow these steps:

1. Select Application Insight, Analyze Transactions from the left navigation menu.
2. In graphical or list view, click Repeated Paths.
3. The path displays an orange Merged badge in **both** views.
4. From graphical view, right-click the merged path and select Shelve (x number) Merged Transactions.

   The merged transactions are placed in the shelf.

Delete Frames in the Shelf

You can delete transaction frames that you do not want to generate artifacts for. Transaction frames can be deleted using the shelf dialog and using the one-click method. See the topic, Using Shelf (see page 51) for information about the one-click method.

Follow these steps:

For the shelf dialog

1. In the Analyze Transactions window, click 2.

   The shelf dialog displays a list of shelved frames.

2. Locate the frame that you want to delete and click 🗑.

   The frame is removed from the shelf.

For the one-click method popup

1. In the Analyze Transaction window, hover the mouse over the shelf icon.
2. Locate the frame that you want to delete and click X.

Search for Paths in the Shelf

The search capability lets you search for paths in the shelf by entering full text search criteria.

Follow these steps:

1. In the Analyze Transactions window, click 2.

   The shelf displays a list of all the shelved components.

2. Enter the text search criteria in the Search Shelf field.

   The shelf returns a list of components that are based on the search criteria.

Generate Artifacts

You generate the following business artifacts for transaction frames that are in the shelf:

- Virtual services
- Baselines
- Documentation

The types of artifacts that can be generated are indicated with colored labels in the list of transactions.

For detailed information about creating artifacts for specific types of transactions, see Creating Baselines (see page 81) and Creating Virtual Services (see page 129).

The following procedure assumes that you have shelved a transaction frame. See the Add a Transaction Frame to the Shelf (see page 54) topic for information about how to shelve a transaction frame.

Follow these steps:

One-click method:

1. Open the Analyze Transactions window.
2. Hover the mouse over the shelf icon .

   The shelf displays with a list of shelved transaction frames.
3. (Optional) Click X to delete any frames for which you do not want to create an artifact.
4. Click an artifact option.
5. Follow the prompts.

Shelf dialog method:

1. Open the Analyze Transactions window.
2. Click .

   The shelf displays with a list of shelved transaction frames.
3. (Optional) Click X to delete any frames for which you do not want to create an artifact.
4. Click an artifact option.
5. (Optional) Click Locate agent in the topology map to view the agent map.
6. (Optional) Select Show All Agents to display all the agents for a transaction.
7. Click Create.

Clear All Transactions from the Shelf

You remove all the transactions that are currently in the shelf by clicking Clear Shelf from the shelf dialog.

Chapter 5: Creating and Managing Tickets

CAI lets you capture detailed information about application behavior, including defects. This feature can help speed up the resolution of defects by enhancing collaboration between testers and developers.

The application must be running on an agent-enabled computer. A ticket is created in the application. The ticket is viewed in the DevTest Portal.

Tickets include such information as a title, severity level, and description. Tickets also include the corresponding paths (see page 31), which expose the actions of the underlying components.

This section contains the following topics:

Create Tickets in an Application

This procedure assumes that the application is running on an agent-enabled computer.

For the email functionality to work, the email settings (see page 63) must be configured.

To enable the HP ALM - Quality Center functionality, see Send Tickets to HP ALM - Quality Center (see page 64).

Before you begin, open the Settings, Agents window and ensure that the capture level (see page 26) for the HTTP Server protocol is set to Full Data.

Follow these steps:

1. While pressing the Alt key, click anywhere in the application window.

   The ticket submission dialog opens.

2. Click Create a Ticket.
3. Enter the following information:
   - Title: A title for the ticket.
   - Email: A message about the new ticket is sent to this email address. This field appears only if the email settings are configured.
   - External Defect Tracking Num
   - Severity: The options are Low, Medium, High, and Critical.
   - Description: This field is optional.
4. To include a screen shot of the application at the time of capture, ensure that the Take screenshot check box is selected.
5. If the functionality for sending tickets to HP ALM - Quality Center is enabled, select the Raise this as a defect in Quality Center check box.
6. Click Submit.
7. If the functionality for sending tickets to HP ALM - Quality Center is enabled, do the following actions:
   a. Type the user name, password, domain, and project of HP ALM - Quality Center and click Save.
   b. If an additional set of fields appears, enter the required information and click Save.

   A message indicates that the ticket was submitted.

8. To display the ticket in the Manage Tickets window, click View Ticket.
9. Click the Close to close the ticket submission dialog.
10. To view the ticket in the New Ticket Alerts portlet of the Home page, click Refresh in the Home page.

Email Settings for New Tickets

When you create a ticket, CAI sends an email notification to a specified email address. The email includes a link to the ticket that is accessible from the Manage Tickets window by clicking Link in the Actions column.

Before you create a ticket, configure the following properties (see page 29) for the broker. To access these properties in the Settings tab, expand the Tickets category and select Email.

The following properties display:

SMTP server

Contains the name of the SMTP server to use for sending the email. You must use a server that allows unauthentication connection to send email.

From

Specifies the sender that appears in the email.

Subject

Specifies the subject line of the email. In the default value, the characters **%1** are replaced with the title of the ticket.

Body

Specifies the body of the email. In the default value, the characters **%1** are replaced with the link to the ticket.

Send Tickets to HP ALM - Quality Center

When you create a ticket (see page 62), you can specify that the ticket is sent to HP ALM - Quality Center.

HP ALM - Quality Center 11 and 12 are supported.

To enable this functionality, complete the following tasks:

- Install the defect submitter
- Set the configuration properties for HP ALM - Quality Center

Installing the Defect Submitter

The installation process creates the following log files in the **LISA_HOME\bin** directory:

- **InstallUtil.InstallLog**
- **LisaQCServices.InstallLog**

The installation process also creates a configuration file named **LisaQCServices.exe.config**. You can use this file to change the default port number of 10017.

Note: If you run the **lisa-qc-defect-submitter-install.bat** file and an error indicates that the specified service already exists, a previous installation of the defect submitter might not have been removed. Run the **lisa-qc-defect-submitter-uninstall.bat** file, and then rerun the **lisa-qc-defect-submitter-install.bat** file.

Follow these steps:

1. Install the HP Quality Center Client Side Components.
2. Install DevTest Solutions.
3. Go to the **LISA_HOME\addons\qc-defect-submit** directory and run the **qc_defect_submitter.exe** file.

   The setup wizard opens.
4. Click Next.

   The Select Destination Directory step opens.
5. Set the destination directory to the **LISA_HOME** directory, and click Next.

   A message indicates that the directory already exists.
6. Click Yes.

   The files are extracted. The files include batch files for installing and uninstalling.
7. Click Finish.

8. Open a command prompt as an administrator.
9. Navigate to the **LISA_HOME\bin** directory and run the **lisa-qc-defect-submitter-install.bat** file.

   A message indicates that the installation has completed.
10. Open the Control Panel and display the services administrative tool.
11. Locate the **LisaQCService.QCDefectSubmitter** service.
12. Configure the service to run automatically and then start the service.

Setting the Configuration Properties for HP ALM - Quality Center

CAI queries the HP ALM - Quality Center instance for the required fields. These fields are added to the ticket submission dialog.

Follow these steps:

1. Open the Agents window.
2. In the left portion, select the broker.
3. Click the Settings tab.
4. Expand the Application Insight category and select Quality Center.
5. Configure the following properties:

   QC host

   The IP address or host name of the HP ALM - Quality Center REST API. For example:

   ```
   172.24.255.255
   ```

   QC port

   The port on which the HP ALM - Quality Center REST API is listening. For example:

   ```
   8080
   ```
6. Save the property changes.
7. Restart the DevTest Java Agent.

Managing Tickets

The Manage Tickets window lets you view all the tickets that were captured and stored in the CAI database.

You can perform the following operations:

- Search Tickets (see page 67)
- View the Transactions for a Ticket (see page 69)
- Edit Ticket Information (see page 70)
- Obtain the Direct URL of a Ticket (see page 69)
- View Ticket Image

You search for tickets by a specific reporter, status, and duration of time. Tickets appear in descending order by date. New and updated tickets also appear in the New Tickets Alert portlet in the Home page.

Tickets can be one of following statuses:

- New
- Identified
- Closed

How to Manage a Ticket

This scenario takes you through the steps of managing a ticket in CAI.

1. Search Tickets (see page 67)
2. Identify Transaction Defects (see page 68)
3. Obtain the Direct URL of a Ticket (see page 69)
4. View the Transactions for a Ticket (see page 69)
5. Edit Ticket Information (see page 70)

Search Tickets

You search for tickets to display the defects that are found during testing from the Manage Tickets window. The Manage Tickets window contains search criteria for finding tickets in the CAI database.

You can find tickets using the following search criteria:

- Reporter - the user who created the ticket
- Status - new, identified, and closed
- Duration of time - by minutes, hours, days, and all time
- Advanced - start and end date, and start and end time
- Search by entering text

The initial status of a ticket is New. When editing the ticket information, you can change the status to Identified or Closed. Tickets that are updated display in the New Ticket Alerts portlet in the Home page.

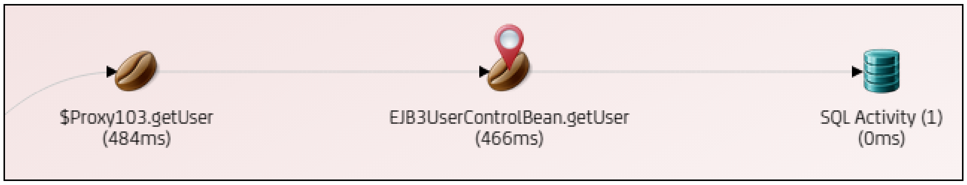
Identify Transaction Defects

You can mark frames in the transactions for a ticket to identify the origin of a defect.

Only tickets with a status of New can be identified.

When you identify a ticket, a red icon is placed on the frame and the background of the path graph turns red. The ticket is removed from the New Ticket Alert portlet of the Home page.

The following graphic shows an identified frame in a path graph.



Follow these steps:

1. Select Application Insight, Manage Tickets in the left navigation menu.

   The Manage Tickets window displays.

2. Click Graphical View from the Actions column.

   The Transactions dialog opens with a visual representation of the transaction paths for the ticket.

3. To mark a frame, right-click the frame and select Identify Problem (Update Ticket Status)

   The status changes from New to Identified in the Actions column.

4. To unmark a frame, right-click the frame and select Unidentify Problem.

   The status changes from Identified to New in the Actions column.

5. (Optional) Go to the Home page and click the Refresh button to update the New Ticket Alerts portlet.

Obtain the Direct URL of a Ticket

CAI lets you obtain a URL that points directly to an existing case.

Follow these steps:

1. Select Application Insight, Manage Tickets in the left navigation menu.

   The Manage Tickets window displays.

2. Click 🔗 for the ticket.

   The case URL displays.

3. Copy the URL that appears in the field.

4. You can now go directly to the ticket by:

   - typing the URL in a web browser or highlighting the URL and right-clicking the URL
   - highlighting the link, right-clicking, and selecting Go to http://<URL>

View the Transactions for a Ticket

The path graph lets you view the corresponding transactions for each ticket. The paths can be displayed with the payload data to determine the cause of the defect.

Follow these steps:

1. Select Application Insight, Manage Tickets in the left navigation menu.

   The Manage Tickets window displays.

2. Click 🖧 in the Actions column.

   The path graph for the transaction displays.

3. Review the transaction.

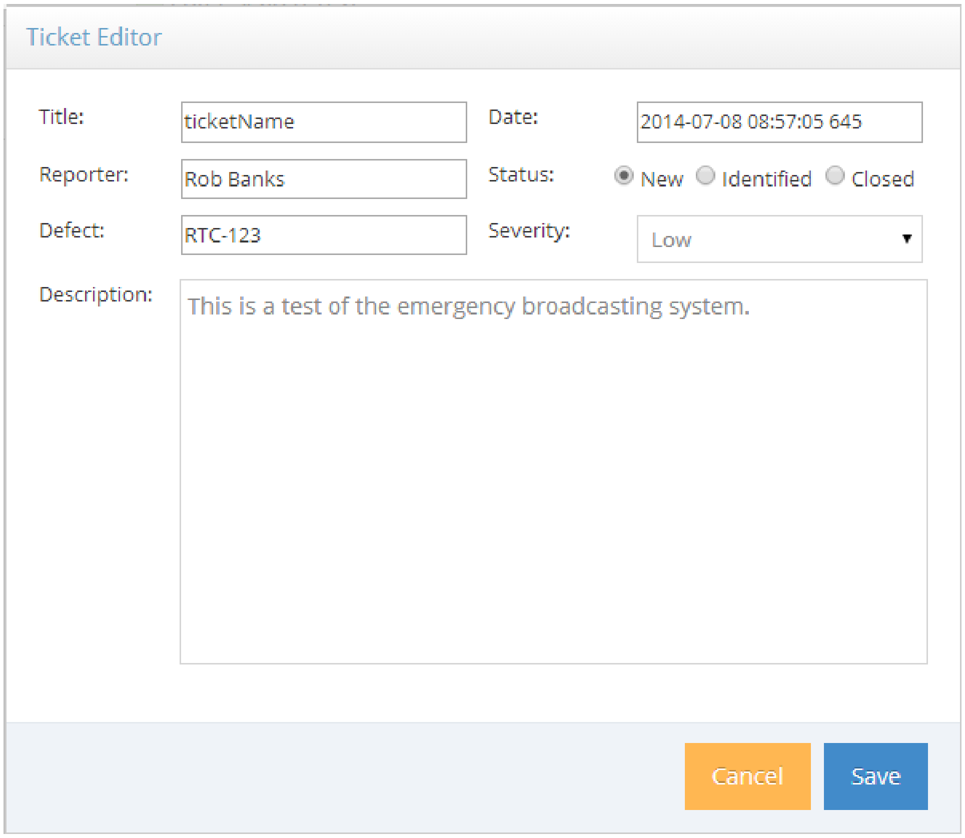
4. (Optional) Right-click a frame and select from the following options:

   - Identify Problem (Update Ticket Status)
   - Generate All Artifacts
   - Generate Document
   - Unidentify Problem

Edit Ticket Information

You can change the information for an existing ticket from the Ticket Editor.

The following graphic displays the Ticket Editor dialog.



Ticket Editor

Title:	ticketName	Date:	2014-07-08 08:57:05 645
Reporter:	Rob Banks	Status:	● New ○ Identified ○ Closed
Defect:	RTC-123	Severity:	Low ▾
Description:	This is a test of the emergency broadcasting system.		

Follow these steps:

1. Click Application Insight, Manage Tickets from the left navigation menu.

   The Manage Tickets pane displays a list of tickets.

2. Click Edit Ticket in the Actions column for the ticket you want to edit.

   The Ticket Editor dialog opens.

3. Edit the appropriate fields and click Save.

   The top of the Manage Tickets window displays a message that the ticket was successfully updated.

   Note: If you change the Status from New to Identified or Closed, the ticket is removed from the New Tickets Alert portlet in the Home page.

View Screen Shot

CAI lets you view the screen shot of the application that was taken at the time of capture.

Follow these steps:

1. Select Application Insight, Manage Tickets in the left navigation menu.

   The Manage Tickets window displays.

2. Click View Screen shot from the Actions column.

   The screen shot displays.

3. Click X to close the screen shot.

Chapter 6: Working with Defects

CAI enables you to identify and analyze defects in transactions, detect performance bottlenecks, and produce defect reporting.

From the Explore Defects window, you can:

- Search for transactions with defects (see page 74)
- Annotate transaction with exceptions, log messages, response time percentage, and view any transactions that have been annotated (see page 75)
- Pin annotated transactions (see page 76)
- Merge Transactions with repeated paths (see page 77)
- Generate Artifacts (see page 78)
- Generate a defect report (see page 79)

This section contains the following topics:

Search for Transactions with Defects

You can find transactions with defects using the search and filter refinement in the Explore Defects window.

Follow these steps:

1. Select Application Insight, Explore Defects from the left navigation menu.

   The Explore Defects window displays.

2. To search by text string, enter the search criteria in the Enter search text field and click Search.
3. To search by time range, click 1 Day and select an option.
4. To perform an advanced search, click 1 Day and select Advanced.
5. Enter or select the search criteria and click Apply.

   The Results pane populates with a list of defects that are based on the search criteria.

6. To filter a search, use the Refined By pane.

   The following filter options are available:

   - Exec Time (ms)
   - Transaction ID
   - Point of Interest

Annotate a Transaction to View Defects

CAI allows you to expose the transactions that have violations and display them in the path.

From the Explore Defects window, you can annotate transactions for:

- Exceptions

  Allows you to identify and view transactions with exception violations.

- Log messages

  Allows you to identity and view log messages for errors and warning.

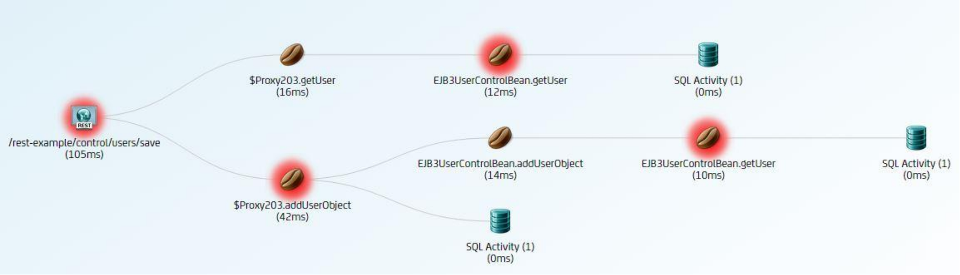
- Response times percentile

  Allows you to set the percentile of the worst performers and view them with in the transaction path.

- Only show annotated transactions

  Allows you to view only transactions that were annotated.

The following graphic displays an example of a transaction with annotated exceptions:



Transactions with log messages that contain warnings are yellow and exceptions, error log messages, and response time violations are red.

Any transactions that are annotated and pinned in the Explore Defects window appear in the Point of Interests list from the Home page.

Follow these steps:

1. Select Application Insight, Explore Defects from the left navigation menu.

   The Explore Defects window displays the transactions in a graphical view.

2. Search for a transaction using the search refinement at the top of the pane.
3. Click Annotation and select one or more options.

   Transactions with defects display.

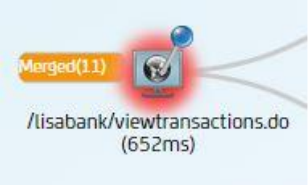
4. (Optional) Select a transaction to view more information about a defect.

   The transaction detail dialog displays.

Pin Annotated Transactions for Point of Interest

After you annotate defect transactions in the Explore Defects window, you can pin them so that they are visible and display in the Points of Interest list of the Home page.

The following graphic displays a defective transaction that is pinned.



Follow these steps:

1. In the Explore Defects window, right-click the node with the violation and select Pin Transaction.
2. Enter a description and click OK.

   The transaction displays a blue pin. The tooltip contains the pin description that was entered.
3. Go to the Home page and click Refresh in the Points of Interest portlet.

   The pinned transaction displays in the Points of Interest list.

To unpin an annotated transaction:

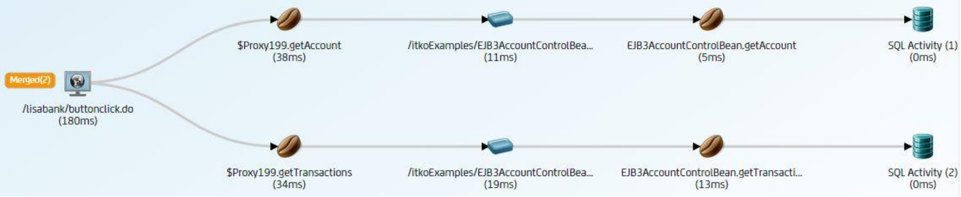
1. In the Explore Defects pane, right-click the blue pinned node and select Remove Pin.

   The pin is removed.
2. Go to the Home page and click Refresh in the Points of Interest portlet.

   The pinned transaction is removed from the Points of Interest list.

View Transactions with Repeated Paths

You can view transactions that have repeated paths in a graphical view.


Merged(2)
/lisabank/buttonclick.do
(180ms)
$Proxy199.getAccount
(38ms)
/itkoExamples/EJB3AccountControlBea...
(11ms)
EJB3AccountControlBean.getAccount
(5ms)
SQL Activity (1)
(0ms)
$Proxy199.getTransactions
(34ms)
/itkoExamples/EJB3AccountControlBea...
(19ms)
EJB3AccountControlBean.getTransacti...
(13ms)
SQL Activity (2)
(0ms)

Follow these steps:

1. Select Application Insight, Explore Defects from the navigation menu.

   The Explore Defects window displays.
2. To find specific transactions, use the search and the refinement filters.
3. Click Repeated Paths.

   Paths with repeated transactions display an orange, merged badge with a count of the number of repeated transactions.

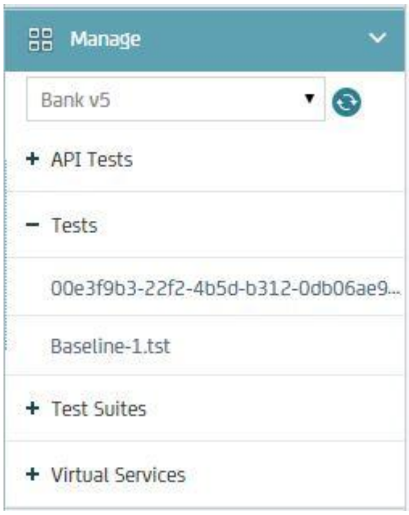
Generate Artifacts in Explore Defects

From the Explore Defects window, you can generate the following artifacts, if applicable:

- Baseline (only expanded is supported)
- Virtual Services

See the Creating Baselines (see page 81) and Creating Virtual Services (see page 129) topics for more information about generating baselines and virtual services in CAI.

The generated artifacts display for the selected project in the Manage menu options. The following graphic displays a baseline that was created in the Bank v5 project from the Explore Defects window:



Follow these steps:

1. Select Application Insight, Explore Defects in the left navigation menu.
2. In the Results pane, right-click a frame and select Generate All Artifacts.

   The Generate All Artifacts dialog opens.
3. Select the type of artifact you want to generate and click Generate.

   Click Advanced from the Generate All Assets dialog if you want to configure the options.
4. In the Select Project dialog, select a project from the drop-down list and click Create.

   A confirmation dialog displays.
5. Click OK.

Generate a Defect Report

You can generate a defect report from the Explore Defects window.

Follow these steps:

1. Select Application Insight, Explore Defects from the left navigation menu.

   The Explore Defects window displays.

2. Search for paths using the Search Text field, using the Refined by filter criteria, or scrolling through the Results list.
3. (Optional) Click Annotate and select one or more options to annotate from the menu.

   Frames with violations appear in red.

4. Right-click the transaction that you want a report and select Generate Document.
5. Enter the title and click OK.

   The report displays in a browser as a PDF file.

6. Click Save in the browser to download and save the report.
7. Enter a file name and click Save.
8. Click Print to print the report.

Chapter 7: Creating Baselines

CAI lets you create baseline test cases and suites from transactions in the CAI database.

When you generate the transactions, ensure that the capture levels (see page 26) for the appropriate protocols are set to Full Data.

You cannot create baselines for transaction frames that have a category of GUI.

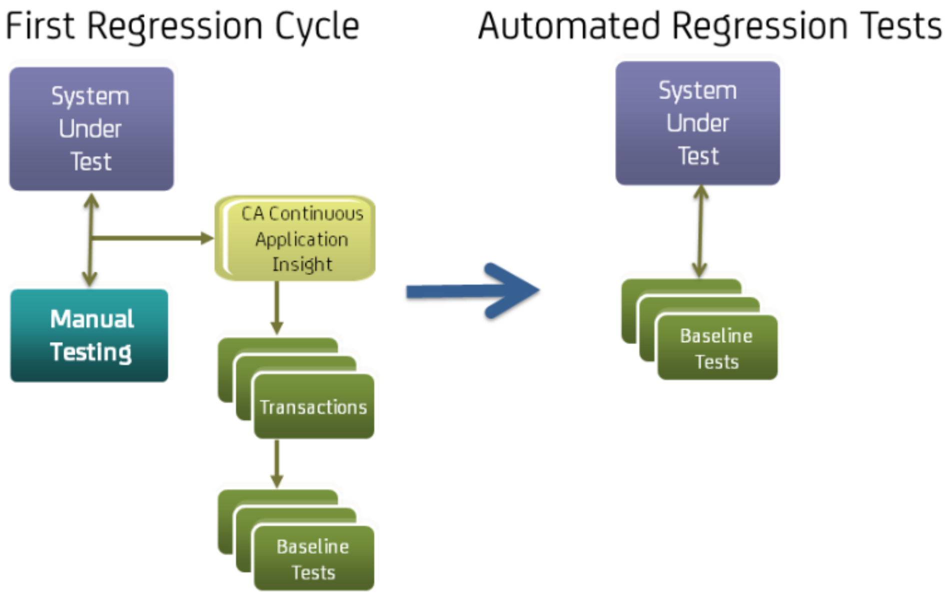
Note: This feature requires a separate license.

This section contains the following topics:

Baselines Overview

Baselines are designed to help automate the testing of enterprise applications.

The following graphic provides a high-level view of how most baselines work.



The left portion shows the initial regression cycle. Manual testing is performed on a system. CA Continuous Application Insight captures the activity and creates transactions that are used to generate baseline test cases.

The right portion shows the subsequent automated regression tests. The baseline test cases are run as necessary against the system under test.

Each baseline test case contains information about the expected response from the system. During an automated regression cycle, the baseline test case compares the expected response with the actual response. If the responses do not match, the differences are highlighted.

Stateful Baselines

A *stateful baseline* is a baseline test case that applies to an entire conversation or session, instead of simply one of its transactions. A conversation is a set of transactions in a session.

Whereas most consolidated baselines contain a single test step, a stateful baseline typically contains multiple test steps.

You can create stateful baselines from the DevTest Portal or the CAI command-line tool.

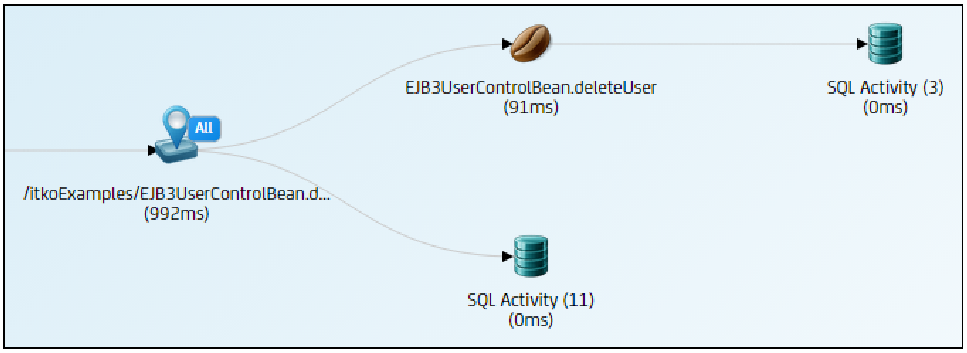
Stateful baselines are supported for the following categories of transaction frame:

- EJB
- REST
- Web HTTP
- Web service

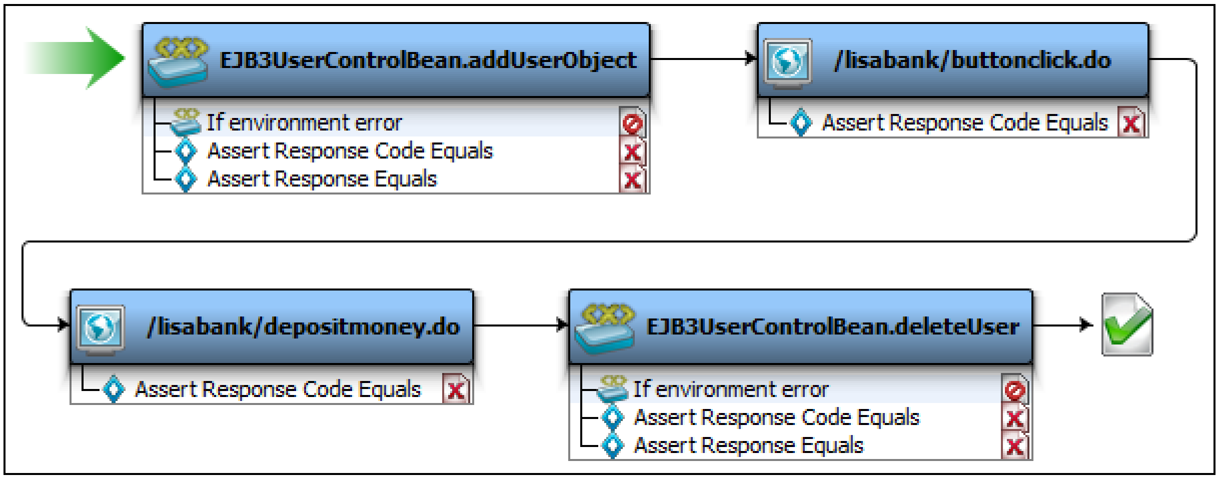
If you add a single transaction frame to the shelf, a stateful baseline is automatically created.

If you add a merged transaction frame to the shelf, you cannot create a stateful baseline. You can create consolidated and expanded baselines.

Assume that you run the **multi-tier-combo** test case in the **examples** project. Multiple paths are generated. The following graphic shows the main portion of one path. The Shelve All Occurred Transactions action has been applied to the web service frame.


EJB3UserControlBean.deleteUser
(91ms)
SQL Activity (3)
(0ms)
All
/itkoExamples/EJB3UserControlBean.d...
(992ms)
SQL Activity (11)
(0ms)

The following graphic shows a stateful baseline that was generated from the shelved frame.


EJB3UserControlBean.addUserObject
If environment error
Assert Response Code Equals
Assert Response Equals
/lisabank/buttonclick.do
Assert Response Code Equals
/lisabank/depositmoney.do
Assert Response Code Equals
EJB3UserControlBean.deleteUser
If environment error
Assert Response Code Equals
Assert Response Equals

EJB Baselines

This section contains the following topics:

Generate an EJB Baseline

Use the following procedure to generate an EJB baseline from a set of transactions in the CAI database.

Follow these steps:

1. Add one or more EJB transaction frames to the shelf.
2. Open the shelf.
3. Click Create Baseline.
4. To change the default name, select the name and make your edits, then click ✓ to save.
5. Click the downward-facing arrow.
6. If the creation mode fields are available, select an option:

   Stateful

   Create a baseline that contains one or more test cases that apply to an entire conversation or session, instead of simply one of its transactions. This option is not available for merged transactions.

   Consolidated

   Create a baseline that contains a single test case and a data set.

   Expanded

   Create a baseline that contains a suite of tests (one for each transaction) and a suite document for running the tests.

7. Specify the type of step that is included in the test case:

   Use Application Test Steps

   The test case includes a step that corresponds to the transaction frame that you added to the shelf. For example, selecting a SOAP frame results in a test case that includes a Web Service Execution (XML) step.

   Use Transaction Frame Step

   The test case includes an Execute Transaction Frame step.

8. (Optional) Configure the baseline to use magic dates.

   Apply magic dates to test cases

   Converts the date strings in the baseline test case or suite to variable definition strings. For example, instead of a string that contains a specific date and time, the string can contain a function that specifies seven days from the current date and time. This behavior is equivalent to the **doDateDeltaFromCurrent** form of magic dates in CA Service Virtualization. If this option is unsupported, the option does not appear.

9. (Optional) Specify a test case to be used as a template for the baseline.

10. If the following fields are available for editing, configure them:

    JNDI Factory

    The JNDI factory class to use when generating EJB baseline tests.

    JNDI URL

    The JNDI URL to use when generating EJB baseline tests.

    Security Principal

    The JNDI user to use when generating EJB baseline tests.

    Security Credentials

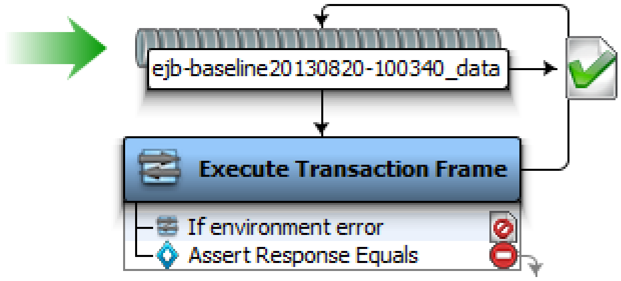
    The JNDI password to use when generating EJB baseline tests.

11. Click Create.
12. Select the project where the baseline will be created.
13. Click Create.

EJB Baseline Test Cases

Consolidated EJB baseline test cases have an Execute Transaction Frame step that reads from a Large Data data set.

The following graphic shows an example of a consolidated EJB baseline test case.



The data set contains the following information:

- Agent name
- Transaction frame
- Expected response

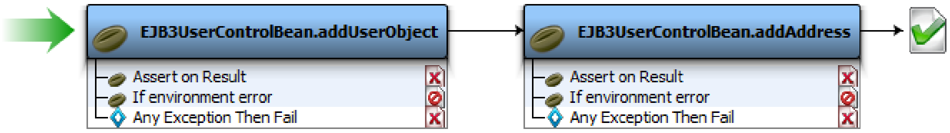
By default, the data set is local, rather than global.

The Execute Transaction Frame step includes a Graphical XML Side-by-Side Comparison assertion with the name **Assert Response Equals**. This assertion verifies that the actual response matches the expected response.

Stateful EJB baseline test cases have one or more Enterprise JavaBean Execution steps. Each step includes the following assertions:

- Embedded assertion with the name **Assert on Result**
- Embedded assertion with the name **If environment error**
- Assert on Invocation Exception assertion with the name **Any Exception Then Fail**

The following graphic shows an example of a stateful EJB baseline test case.



To run the test case, do one of the following actions:

- Stage a quick test
- Stage a test case

You can monitor the results in the Consolidated Baseline tab (see page 125).

Note: For information about how to stage a quick test or a test case, see Running Test Cases and Suites in *Using CA Application Test*.

EJB Baseline Suites

The test cases in an EJB baseline suite are located in the Baselines folder, instead of the Tests folder.

The suite document is located in the Suites folder.

The test cases in an EJB baseline suite have an Enterprise JavaBean Execution step or an Execute Transaction Frame step.

When you run the suite, the outcome of each test is shown in the Baseline Results panel (see page 127).

Note: For information about how to run a suite, see Run a Test Suite in *Using CA Application Test*.

JMS Baselines

JMS baselines are supported for configurations in which JNDI is used to obtain the JMS connection factory.

JMS baselines are also supported for configurations in which a direct API from IBM WebSphere MQ, Java CAPS, SonicMQ, or TIBCO is used to obtain the JMS connection factory.

Note: If a service or client reuses or forwards JMS message objects without changing them, the behavior of CA Continuous Application Insight is undefined.

This section contains the following topics:

Generate a JMS Baseline

Use the following procedure to generate a JMS baseline from a set of transactions in the CAI database.

Follow these steps:

1. Add a JMS transaction frame (see page 90) to the shelf.
2. Open the shelf.
3. Click Create Baseline.
4. To change the default name, select the name and make your edits, then click ✓ to save.
5. Click the downward-facing arrow.
6. If the creation mode fields are available, select an option:

   Consolidated

   Create a baseline that contains a single test case and a data set.

   Expanded

   Create a baseline that contains a suite of tests (one for each transaction) and a suite document for running the tests.

7. Specify the type of step that is included in the test case:

   Use Application Test Steps

   The test case includes a step that corresponds to the transaction frame that you added to the shelf. For example, selecting a SOAP frame results in a test case that includes a Web Service Execution (XML) step.

   Use Transaction Frame Step

   The test case includes an Execute Transaction Frame step.

8. (Optional) Configure the baseline to use magic dates:

   Apply magic dates to test cases

   Converts the date strings in the baseline test case or suite to variable definition strings. For example, instead of a string that contains a specific date and time, the string can contain a function that specifies seven days from the current date and time. This behavior is equivalent to the **doDateDeltaFromCurrent** form of magic dates in CA Service Virtualization. If this option is unsupported, the option does not appear.

9. (Optional) Specify a test case to be used as a template for the baseline.
10. Click Create.
11. Select the project where the baseline will be created.
12. Click Create.

JMS Baseline Transaction Frames

The procedure for generating a JMS baseline includes a step where you add a JMS transaction frame to the shelf.

For a scenario with one request and one response, the path graph from which you select a transaction frame to shelve contains four transaction frames. These frames represent the following activities:

- The client produces the request message.
- The service consumes the request message.
- The service produces the response message.
- The client consumes the response message.

Select the frame that represents the client producing the request message. For an exception to this rule, see the following section about duplicate nodes.

CAI searches for transactions that contain the same name as the frame that you select. The baseline contains all transactions that involve sending a message to or receiving a message from the same queue.

Duplicate Nodes

Under some circumstances, the path graph contains two adjacent frames that represent the same produce or consume operation. The names of the adjacent frames both begin with either **send:** or **recv:**.

If the first two frames in the path graph begin with **send:**, select the first of the frames.

If the first two frames in the path graph begin with **recv:**, select the second of the frames.

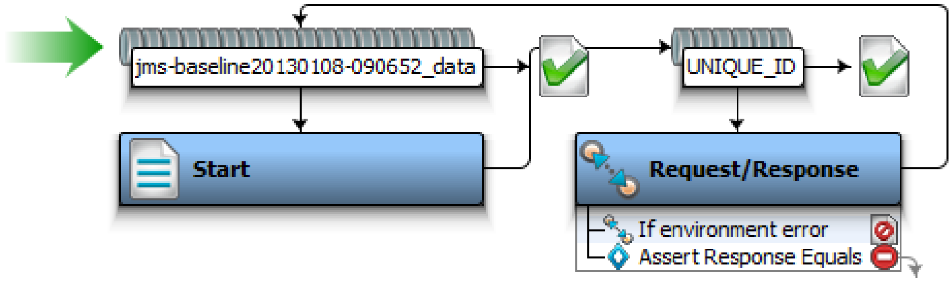
JMS Baseline Test Cases

The test case for most single-request, single-response scenarios have two steps:

- A do-nothing step that reads from a Large Data data set
- A messaging step that combines the request and response. The messaging step can be any of the following steps: JMS Messaging (JNDI), IBM WebSphere MQ, JCAPS Messaging (Native), SonicMQ Messaging (Native), or TIBCO Direct JMS. In the WebSphere MQ step, the client mode is set to JMS.

The test case can also include a Unique Code Generator data set for generating a correlation ID.

The following graphic shows an example test case. The first step is a do-nothing step. The second step is a messaging step with the name **Request/Response**.

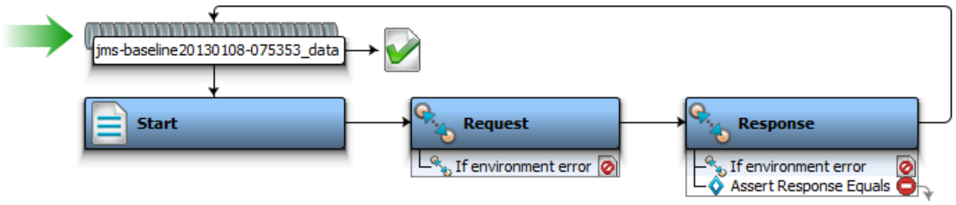


The data set contains information that changes for each transaction in the baseline. This information includes the request payload, response payload, and JMS message properties. By default, the data set is local, instead of global.

The messaging step includes a Graphical XML Side-by-Side Comparison assertion with the name **Assert Response Equals**. When you run the baseline test case, this assertion verifies that the actual response matches the expected response. If the responses do not match, then the test generates an error.

Other scenarios divide the request and response into separate steps.

The following graphic shows an example test case. The first step is a do-nothing step. The second step is a messaging step with the name **Request**. The third step is a messaging step with the name **Response**.



Configuration properties are generated for destination information, connection information, and XML diff options. If the client and service use different JNDI connection settings, then two sets of properties are generated for the connection information.

To run the test case, do one of the following actions:

- Stage a quick test
- Stage a test case

You can monitor the results in the Consolidated Baseline tab (see page 125).

Note: For information about how to stage a quick test or a test case, see Running Test Cases and Suites in *Using CA Application Test*.

JMS Baseline Suites

The test cases in a JMS baseline suite are located in the Baselines folder, instead of the Tests folder.

The suite document is located in the Suites folder.

The test cases in a JMS baseline suite are similar to JMS baseline test cases. However, they do not include a Large Data data set. The data is stored in the step instead.

As with JMS baseline test cases, configuration properties are generated.

When you run the suite, the outcome of each test is shown in the Baseline Results panel (see page 127).

Note: For information about how to run a suite, see Run a Test Suite in *Using CA Application Test*.

REST Baselines

This section contains the following topics:

Generate a REST Baseline

Use the following procedure to generate a REST baseline from a set of transactions in the CAI database.

Follow these steps:

1. Add one or more REST transaction frames to the shelf.
2. Open the shelf.
3. Click Create Baseline.
4. To change the default name, select the name and make your edits, then click ✓ to save.
5. Click the downward-facing arrow.
6. If the creation mode fields are available, select an option:

   Stateful

   Create a baseline that contains one or more test cases that apply to an entire conversation or session, instead of simply one of its transactions. This option is not available for merged transactions.

   Consolidated

   Create a baseline that contains a single test case and a data set.

   Expanded

   Create a baseline that contains a suite of tests (one for each transaction) and a suite document for running the tests.

7. Specify the type of step that is included in the test case:

   Use Application Test Steps

   The test case includes a step that corresponds to the transaction frame that you added to the shelf. For example, selecting a SOAP frame results in a test case that includes a Web Service Execution (XML) step.

   Use Transaction Frame Step

   The test case includes an Execute Transaction Frame step.

8. (Optional) Configure the baseline to use magic dates.

   Apply magic dates to test cases

   Converts the date strings in the baseline test case or suite to variable definition strings. For example, instead of a string that contains a specific date and time, the string can contain a function that specifies seven days from the current date and time. This behavior is equivalent to the **doDateDeltaFromCurrent** form of magic dates in CA Service Virtualization. If this option is unsupported, the option does not appear.

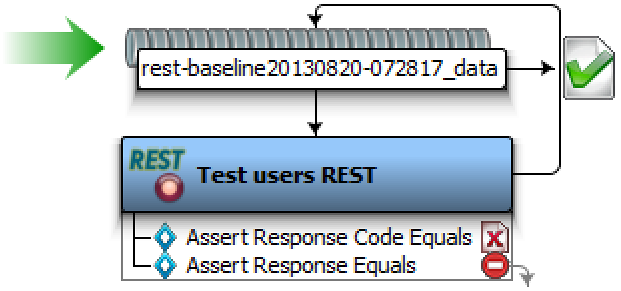
9. (Optional) Specify a test case to be used as a template for the baseline.

10. Click Create.
11. Select the project where the baseline will be created.
12. Click Create.

REST Baseline Test Cases

Consolidated REST baseline test cases have a test step that reads from a Large Data data set. The test step is either a REST step or an Execute Transaction Frame step.

The following graphic shows an example of a consolidated REST baseline test case. The test case has a REST step.


rest-baseline20130820-072817_data
REST
Test users REST
Assert Response Code Equals
Assert Response Equals

If the test case has a REST step, the data set contains such information as the URL, request body, expected response, and response code.

If the test case has an Execute Transaction Frame step, the data set contains the agent name, transaction frame, and expected response.

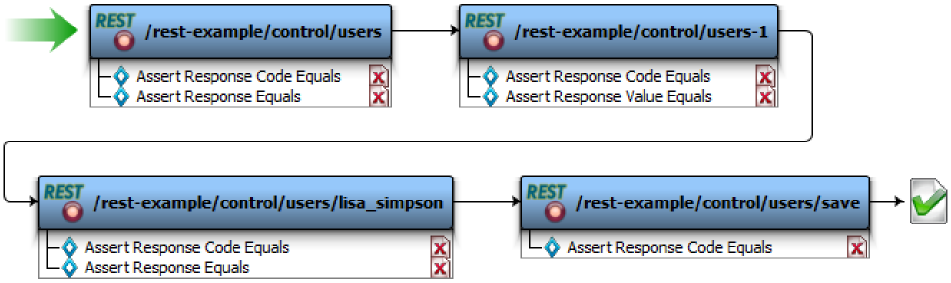
By default, the data set is local, instead of global.

The test step includes one or more of the following assertions:

- **Assert Response Code Equals.** This assertion verifies that the actual response code matches the expected response code.
- **Assert Response Equals.** This assertion verifies that the actual response matches the expected response. The type of assertion depends on whether the response format is XML, JSON, or raw text.

Stateful REST baseline test cases have one or more REST steps. Each step includes the **Assert Response Code Equals** assertion. Each step may include the **Assert Response Equals** assertion or the **Assert Response Value Equals** assertion.

The following graphic shows an example of a stateful REST baseline test case.


REST
/rest-example/control/users
Assert Response Code Equals
Assert Response Equals
REST
/rest-example/control/users-1
Assert Response Code Equals
Assert Response Value Equals
REST
/rest-example/control/users/lisa_simpson
Assert Response Code Equals
Assert Response Equals
REST
/rest-example/control/users/save
Assert Response Code Equals

To run the test case, do one of the following actions:

- Stage a quick test
- Stage a test case

You can monitor the results in the Consolidated Baseline tab (see page 125).

Note: For information about how to stage a quick test or a test case, see Running Test Cases and Suites in *Using CA Application Test*.

REST Baseline Suites

The test cases in a REST baseline suite are located in the Baselines folder, instead of the Tests folder.

The suite document is located in the Suites folder.

The test cases in a REST baseline suite are similar to REST baseline test cases. However, they do not include a Large Data data set. The data is stored in the test step instead.

When you run the suite, the outcome of each test is shown in the Baseline Results panel (see page 127).

Note: For information about how to run a suite, see Run a Test Suite in *Using CA Application Test*.

HTTP Client Calls

When an application makes an HTTP client call using the Apache HttpClient API for POST, GET, PUT, and DELETE, CAI assigns the REST category to the transaction frame.

To force the POST and GET calls to be categorized as HTTP instead, add the **lisa.agent.rest.enabled** property to the **agent** element of the **rules.xml** file. Set the value to false.

```
<property key="lisa.agent.rest.enabled" value="false"/>
```

If the registry is running when you update the **rules.xml** file, restart the registry.

Forcing the PUT and DELETE calls to be categorized as HTTP is not supported.

TIBCO BusinessWorks Baselines

This section contains the following topics:

Generate a TIBCO BusinessWorks Baseline

Use the following procedure to generate a TIBCO ActiveMatrix BusinessWorks baseline from a set of transactions in the CAI database.

Follow these steps:

1. Add a TIBCO ActiveMatrix BusinessWorks transaction frame to the shelf.
2. Open the shelf.
3. Click Create Baseline.
4. To change the default name, select the name and make your edits, then click ✓ to save.
5. Click the downward-facing arrow.
6. If the creation mode fields are available, select an option:

   Consolidated

   Create a baseline that contains a single test case and a data set.

   Expanded

   Create a baseline that contains a suite of tests (one for each transaction) and a suite document for running the tests.
7. (Optional) Configure the baseline to use magic dates.

   Apply magic dates to test cases

   Converts the date strings in the baseline test case or suite to variable definition strings. For example, instead of a string that contains a specific date and time, the string can contain a function that specifies seven days from the current date and time. This behavior is equivalent to the **doDateDeltaFromCurrent** form of magic dates in CA Service Virtualization. If this option is unsupported, the option does not appear.
8. (Optional) Specify a test case to be used as a template for the baseline.
9. Click Create.
10. Select the project where the baseline will be created.
11. Click Create.

TIBCO BusinessWorks Baseline Test Cases

TIBCO ActiveMatrix BusinessWorks baseline test cases have an Execute Transaction Frame step that reads from a Large Data data set.

The data set contains the following information:

- Agent name
- Transaction frame
- Expected response

By default, the data set is local, instead of global.

The Execute Transaction Frame step includes a Graphical XML Side-by-Side Comparison assertion with the name **Assert Response Value Equals**. When you run the baseline test case, this assertion verifies that the actual response matches the expected response. If the responses do not match, then the test generates an error.

To run the test case, do one of the following actions:

- Stage a quick test
- Stage a test case

You can monitor the results in the Consolidated Baseline tab (see page 125).

Note: For information about how to stage a quick test or a test case, see Running Test Cases and Suites in *Using CA Application Test*.

TIBCO BusinessWorks Baseline Suites

The test cases in a TIBCO ActiveMatrix BusinessWorks baseline suite are located in the Baselines folder, instead of the Tests folder.

The suite document is located in the Suites folder.

The test cases in a TIBCO ActiveMatrix BusinessWorks baseline suite have an Execute Transaction Frame step.

When you run the suite, the outcome of each test is shown in the Baseline Results panel (see page 127).

Note: For information about how to run a suite, see Run a Test Suite in *Using CA Application Test*.

TIBCO Enterprise Message Service Baselines

The baseline feature is supported for TIBCO Enterprise Message Service (EMS).

The procedure is the same as for JMS baselines. However, before you start, copy the **tibjms.jar** file into the **LISA_HOME\lib** directory.

Web HTTP Baselines

This section contains the following topics:

Generate a Web HTTP Baseline

Use the following procedure to generate a Web HTTP baseline from a set of transactions in the CAI database.

Signed Web HTTP transactions are supported. However, CAI does not save SSL information to the baseline. If the SSL information is required, configure the following fields in the generated baseline:

- SSL Keystore File
- SSL Keystore Password
- SSL Key Alias
- SSL Key Password

These fields are located in the HTTP/HTML Request step.

Follow these steps:

1. Add one or more Web HTTP transaction frames to the shelf.
2. Open the shelf.
3. Click Create Baseline.
4. To change the default name, select the name and make your edits, then click ✓ to save.
5. Click the downward-facing arrow.
6. If the creation mode fields are available, select an option:

   Stateful

   Create a baseline that contains one or more test cases that apply to an entire conversation or session, instead of simply one of its transactions. This option is not available for merged transactions.

   Consolidated

   Create a baseline that contains a single test case and a data set.

   Expanded

   Create a baseline that contains a suite of tests (one for each transaction) and a suite document for running the tests.

7. Specify the type of step that is included in the test case:

   Use Application Test Steps

   The test case includes a step that corresponds to the transaction frame that you added to the shelf. For example, selecting a SOAP frame results in a test case that includes a Web Service Execution (XML) step.

   Use Transaction Frame Step

The test case includes an Execute Transaction Frame step.

8. (Optional) Configure the baseline to use magic dates.

   Apply magic dates to test cases

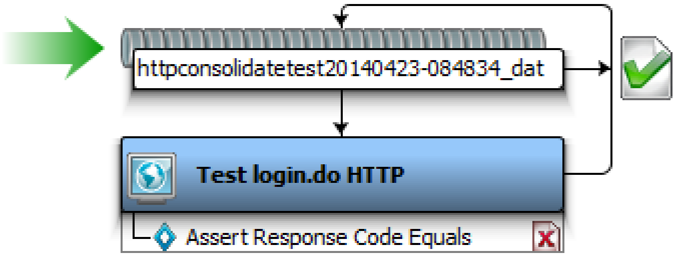
   Converts the date strings in the baseline test case or suite to variable definition strings. For example, instead of a string that contains a specific date and time, the string can contain a function that specifies seven days from the current date and time. This behavior is equivalent to the **doDateDeltaFromCurrent** form of magic dates in CA Service Virtualization. If this option is unsupported, the option does not appear.

9. (Optional) Specify a test case to be used as a template for the baseline.
10. Click Create.
11. Select the project where the baseline will be created.
12. Click Create.

Web HTTP Baseline Test Cases

Consolidated Web HTTP baseline test cases have a test step that reads from a Large Data data set. The test step is an HTTP/HTML Request step.

The following graphic shows an example of a consolidated Web HTTP baseline test case. The test case has an HTTP/HTML Request step.


httpconsolidatetest20140423-084834_dat
Test login.do HTTP
Assert Response Code Equals

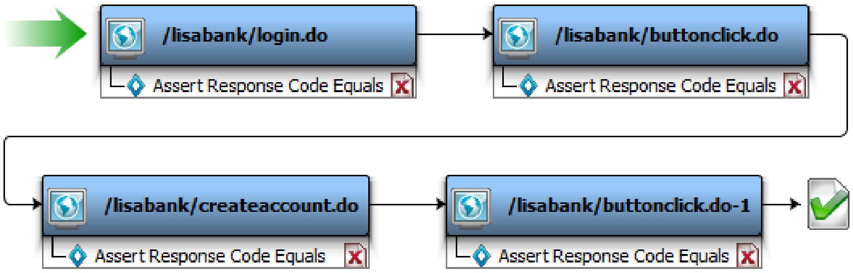
The data set contains such information as the URL, request body, expected response, and response code.

By default, the data set is local, rather than global.

The test step includes the **Assert Response Code Equals** assertion**.** This assertion verifies that the actual response code matches the expected response code.

Stateful Web HTTP baseline test cases have one or more HTTP/HTML Request steps. Each step includes the **Assert Response Code Equals** assertion.

The following graphic shows an example of a stateful Web HTTP baseline test case.


/lisabank/login.do
Assert Response Code Equals
/lisabank/buttonclick.do
Assert Response Code Equals
/lisabank/createaccount.do
Assert Response Code Equals
/lisabank/buttonclick.do-1
Assert Response Code Equals

To run the test case, do one of the following actions:

- Stage a quick test
- Stage a test case

You can monitor the results in the Consolidated Baseline tab (see page 125).

Note: For information about how to stage a quick test or a test case, see Running Test Cases and Suites in *Using CA Application Test*.

Web HTTP Baseline Suites

The test cases in a Web HTTP baseline suite are located in the Baselines folder, rather than the Tests folder.

The suite document is located in the Suites folder.

The test cases in a Web HTTP baseline suite are similar to Web HTTP baseline test cases. However, they do not include a Large Data data set. The data is stored in the test step instead.

When you run the suite, the outcome of each test is shown in the Baseline Results panel (see page 127).

Note: For information about how to run a suite, see Run a Test Suite in *Using CA Application Test*.

HTTP Client Calls

When an application makes an HTTP client call using the Apache HttpClient API for POST, GET, PUT, and DELETE, CAI assigns the REST category to the transaction frame.

To force the POST and GET calls to be categorized as HTTP instead, add the **lisa.agent.rest.enabled** property to the **agent** element of the **rules.xml** file. Set the value to false.

```
<property key="lisa.agent.rest.enabled" value="false"/>
```

If the registry is running when you update the **rules.xml** file, restart the registry.

Forcing the PUT and DELETE calls to be categorized as HTTP is not supported.

Web Service Baselines

This section contains the following topics:

Generate a Web Service Baseline

Use the following procedure to generate a web service baseline from a set of transactions in the CAI database.

Web services with attachments (see page 106) are supported.

Signed SOAP messages are supported on all platforms except IBM WebSphere Application Server 8.5. However, CAI does not save SSL information to the baseline. If the SSL information is required, configure the following fields in the generated baseline:

- SSL Keystore File
- SSL Keystore Password
- SSL Key Alias
- SSL Key Password

These fields are located in the Web Service Execution (XML) step.

This feature supports transport-level security, which is a point-to-point security model. This feature does not support signed message-level security, which is an end-to-end security model.

Follow these steps:

1. Add one or more web service transaction frames to the shelf.
2. Open the shelf.
3. Click Create Baseline.
4. To change the default name, select the name and make your edits, then click ✓ to save.
5. Click the downward-facing arrow.
6. If the creation mode fields are available, select an option:

   Stateful

   Create a baseline that contains one or more test cases that apply to an entire conversation or session, instead of simply one of its transactions. This option is not available for merged transactions.

   Consolidated

   Create a baseline that contains a single test case and a data set.

   Expanded

   Create a baseline that contains a suite of tests (one for each transaction) and a suite document for running the tests.

7. Specify the type of step that is included in the test case:

Use Application Test Steps

The test case includes a step that corresponds to the transaction frame that you added to the shelf. For example, selecting a SOAP frame results in a test case that includes a Web Service Execution (XML) step.

Use Transaction Frame Step

The test case includes an Execute Transaction Frame step.

8. (Optional) Configure the baseline to use magic dates.

   Apply magic dates to test cases

   Converts the date strings in the baseline test case or suite to variable definition strings. For example, instead of a string that contains a specific date and time, the string can contain a function that specifies seven days from the current date and time. This behavior is equivalent to the **doDateDeltaFromCurrent** form of magic dates in CA Service Virtualization. If this option is unsupported, the option does not appear.

9. (Optional) Specify a test case to be used as a template for the baseline.
10. Click Create.
11. Select the project where the baseline will be created.
12. Click Create.

Web Services with Attachments

You can generate a baseline for web services that have the following attachment types:

- MIME
- DIME
- XOP
- MTOM

CAI assumes that if a web service has multiple attachments, the attachments are the same type.

The Use Transaction Frame Step option is not supported for web services with attachments.

Outbound web service calls with attachments are not supported.

When the baseline is created, the attachment content is saved in the frame. The type is **Base64 Encoded**.

If the frame is generated from an MTOM attachment transaction, the attachment type in the baseline is XOP.

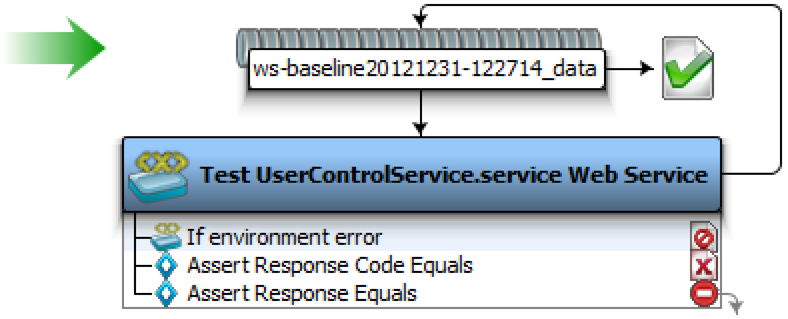
By default, the maximum size of an attachment is 10 MB. If the attachment is larger, the transaction frame is marked as truncated and the attachment content is not saved in the frame.

To change the default maximum size, go to the Agents window (see page 19). Select the agent or broker. Click the Settings tab and select the Transactions category. Configure the Max Attachment Size property. Set the value to the number of kilobytes, not megabytes. Then click Save.

Web Service Baseline Test Cases

Consolidated web service baseline test cases have a test step that reads from a Large Data data set. The test step is either a Web Service Execution (XML) step or an Execute Transaction Frame step.

The following graphic shows an example of a consolidated web service baseline test case. The test case has a Web Service Execution (XML) step.


ws-baseline20121231-122714_data
Test UserControlService.service Web Service
If environment error
Assert Response Code Equals
Assert Response Equals

If the test case has a Web Service Execution (XML) step, the data set contains information such as the URL, request body, expected response, and response code. The data set can also include data from web service attachments.

If the test case has an Execute Transaction Frame step, the data set contains the agent name, transaction frame, and expected response.

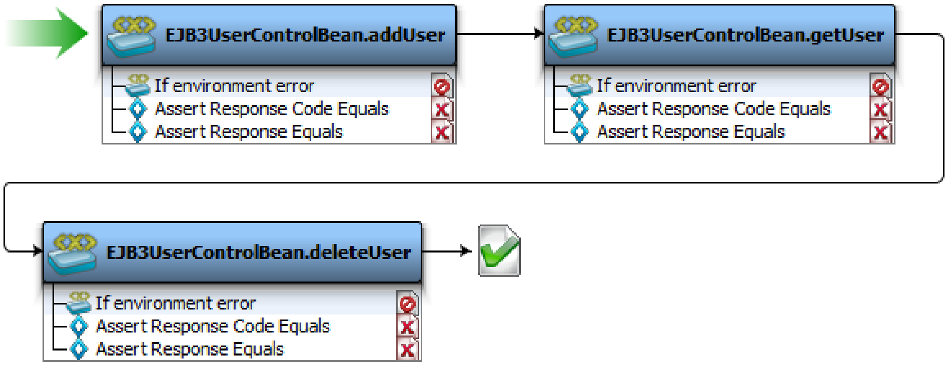
By default, the data set is local, instead of global.

The test step includes one or more of the following assertions:

- **Assert Response Code Equals.** This assertion verifies that the actual response code matches the expected response code.
- **Assert Response Equals.** This assertion verifies that the actual response matches the expected response. The type of assertion depends on whether the response format is XML, JSON, or raw text.

Stateful web service baseline test cases have one or more Web Service Execution (XML) steps. Each step includes the **Assert Response Code Equals** assertion and the **Assert Response Equals** assertion.

The following graphic shows an example of a stateful web service baseline test case.


EJB3UserControlBean.addUser
If environment error
Assert Response Code Equals
Assert Response Equals
EJB3UserControlBean.getUser
If environment error
Assert Response Code Equals
Assert Response Equals
EJB3UserControlBean.deleteUser
If environment error
Assert Response Code Equals
Assert Response Equals

To run the test case, do one of the following actions:

- Stage a quick test
- Stage a test case

You can monitor the results in the Consolidated Baseline tab (see page 125).

Note: For information about how to stage a quick test or a test case, see Running Test Cases and Suites in *Using CA Application Test*.

Web Service Baseline Suites

The test cases in a web service baseline suite are located in the Baselines folder, instead of the Tests folder.

The suite document is located in the Suites folder.

The test cases in a web service baseline suite are similar to web service baseline test cases. However, they do not include a Large Data data set. The data is stored in the test step instead.

When you run the suite, the outcome of each test is shown in the Baseline Results panel (see page 127).

Note: For information about how to run a suite, see Run a Test Suite in *Using CA Application Test*.

webMethods Baselines

This section contains the following topics:

Generate a webMethods Baseline

Use the following procedure to generate a webMethods Integration Server baseline from a set of transactions in the CAI database.

webMethods Integration Server includes the concepts of a flow service and a pipeline. A flow service lets you encapsulate a group of services and manage the flow of data among them. The pipeline is a data structure that contains the input and output values for a flow service. You can add steps to a flow service to perform such actions as invoking services and changing data in the pipeline.

The flow service is the unit that is baselined.

Follow these steps:

1. Add a webMethods Integration Server transaction frame to the shelf.
2. Open the shelf.
3. Click Create Baseline.
4. To change the default name, select the name and make your edits, then click ✓ to save.
5. Click the downward-facing arrow.
6. If the creation mode fields are available, select an option:

   Stateful

   Create a baseline that contains one or more test cases that apply to an entire conversation or session, instead of simply one of its transactions. This option is not available for merged transactions.

   Consolidated

   Create a baseline that contains a single test case and a data set.

   Expanded

   Create a baseline that contains a suite of tests (one for each transaction) and a suite document for running the tests.

7. Specify the type of step that is included in the test case:

   Use Application Test Steps

   The test case includes a step that corresponds to the transaction frame that you added to the shelf. For example, selecting a SOAP frame results in a test case that includes a Web Service Execution (XML) step.

   Use Transaction Frame Step

   The test case includes an Execute Transaction Frame step.

8. (Optional) Configure the baseline to use magic dates.

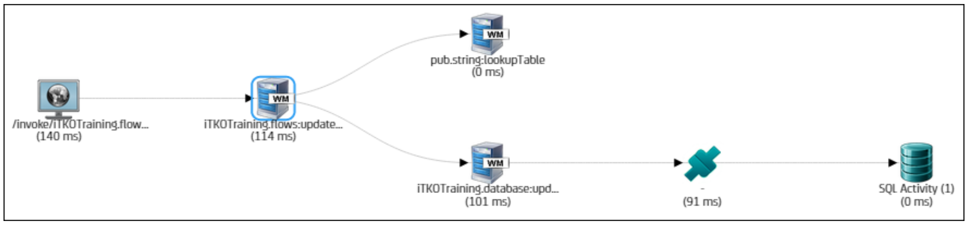
   Apply magic dates to test cases

Converts the date strings in the baseline test case or suite to variable definition strings. For example, instead of a string that contains a specific date and time, the string can contain a function that specifies seven days from the current date and time. This behavior is equivalent to the **doDateDeltaFromCurrent** form of magic dates in CA Service Virtualization. If this option is unsupported, the option does not appear.

9. (Optional) Specify a test case to be used as a template for the baseline.
10. Click Create.
11. Select the project where the baseline will be created.
12. Click Create.

Example: Update User

The following graphic shows a path graph that includes webMethods Integration Server frames. The flow service in this example is named **updateUser**. The purpose is to send updated information for a user to a database.



Note: If you select any of the webMethods Integration Server frames, you can view the pipeline data in the transaction detail dialog.

When a path graph has multiple webMethods Integration Server frame, you must select one frame for baselining. Typically, you select the entry point. In this example, you would select the leftmost webMethods Integration Server frame.

webMethods Baseline Test Cases

webMethods baseline test cases have a test step that reads from a Large Data data set. The test step is either a webMethods Integration Server Services step or an Execute Transaction Frame step.

If the test case has a webMethods Integration Server Services step, then the data set contains such information as the URL, request body, expected response, and response code.

If the test case has an Execute Transaction Frame step, then the data set contains the agent name, transaction frame, and expected response.

By default, the data set is local, instead of global.

The test step includes one or both of the following assertions:

- **Assert Response Code Equals**. When you run the baseline test case, this assertion verifies that the actual response code matches the expected response code. If the codes do not match, then the test generates an error.
- **Assert Response Equals**. When you run the baseline test case, this assertion verifies that the actual response value matches the expected response value. If the values do not match, then the test generates an error.

To run the test case, do one of the following actions:

- Stage a quick test
- Stage a test case

You can monitor the results in the Consolidated Baseline tab (see page 125).

Note: For information about how to stage a quick test or a test case, see Running Test Cases and Suites in *Using CA Application Test*.

webMethods Baseline Suites

The test cases in a webMethods baseline suite are located in the Baselines folder, instead of the Tests folder.

The suite document is located in the Suites folder.

The test cases in a webMethods baseline suite are similar to webMethods baseline test cases. However, they do not include a Large Data data set. The data is stored in the test step instead.

When you run the suite, the outcome of each test is shown in the Baseline Results panel (see page 127).

Note: For information about how to run a suite, see Run a Test Suite in *Using CA Application Test*.

WebSphere MQ Baselines

This section contains the following topics:

Generate a WebSphere MQ Baseline

Use the following procedure to generate a WebSphere MQ baseline from a set of transactions in the CAI database.

Prerequisites: Using DevTest with this application requires that you make one or more files available to DevTest. For more information, see Third-Party File Requirements in *Administering*.

Follow these steps:

1. Determine which WebSphere MQ scenario (see page 116) you want to use, and enable the DevTest Java Agent as appropriate.
2. (Optional) Configure the WebSphere MQ properties (see page 117) in the **rules.xml** file.
3. Add a WebSphere MQ transaction frame to the shelf. The path graph from which you select a transaction frame to shelve includes multiple WebSphere MQ transaction frames. Be sure to select the *first* WebSphere MQ transaction frame.
4. Open the shelf.
5. Click Create Baseline.
6. To change the default name, select the name and make your edits, then click ✓ to save.
7. Click the downward-facing arrow.
8. If the creation mode fields are available, select an option:

   Consolidated

   Create a baseline that contains a single test case and a data set.

   Expanded

   Create a baseline that contains a suite of tests (one for each transaction) and a suite document for running the tests.
9. Specify the type of step that is included in the test case:

   Use Application Test Steps

   The test case includes a step that corresponds to the transaction frame that you added to the shelf. For example, selecting a SOAP frame results in a test case that includes a Web Service Execution (XML) step.

   Use Transaction Frame Step

   The test case includes an Execute Transaction Frame step.
10. (Optional) Configure the baseline to use magic dates.

    Apply magic dates to test cases

Converts the date strings in the baseline test case or suite to variable definition strings. For example, instead of a string that contains a specific date and time, the string can contain a function that specifies seven days from the current date and time. This behavior is equivalent to the **doDateDeltaFromCurrent** form of magic dates in CA Service Virtualization. If this option is unsupported, the option does not appear.

11. (Optional) Specify a test case to be used as a template for the baseline.
12. Click Create.
13. Select the project where the baseline will be created.
14. Click Create.

WebSphere MQ Baseline Scenarios

When working with WebSphere MQ, you can enable the DevTest Java Agent on the client, the server, or both the client and the server. The contents of the path graph vary depending on the scenario.

Note: DevTest can behave as if it is an agent-enabled client. This behavior occurs when a test case uses an IBM WebSphere MQ step to invoke the service.

Scenario 1: Client Enabled, Service Not Enabled

The client is agent-enabled, and the service is not agent-enabled. This scenario simulates a Java-based client running against a Windows or mainframe-based service that cannot be agent-enabled.

When you view the transaction details in the DevTest Portal, the path graph contains two nodes.

The first node represents the request. To view the message body, select the node and then click the Request tab.

The second node represents the response. To view the message body, select the node and then click the Response tab.

Scenario 2: Client Not Enabled, Service Enabled

The client is not agent-enabled, and the service is agent-enabled. This scenario simulates a Windows or mainframe-based client that cannot be agent-enabled running against a Java-based service.

The first transaction in the DevTest Portal is a priming transaction. The agent on the service side discovers that it is running on the service side and initializes itself.

The subsequent transactions are the same as for scenario 1. The path graph contains a request node and a response node.

Scenario 3: Client Enabled, Service Enabled

Both the client and the service are agent-enabled. This scenario simulates a Java-based client running against a Java-based service.

The first transaction in the DevTest Portal is a priming transaction. The agent on the service side discovers that it is running on the service side and initializes itself.

For the subsequent transactions, the path graph contains four nodes:

- The first node represents the client PUT for the request.
- The second node represents the service GET for the request.
- The third node represents the service PUT for the response.
- The fourth node represents the client GET for the response.

WebSphere MQ Baseline Properties

The configuration file for the DevTest Java Agent is named **rules.xml**.

Note: For detailed information about this file, see *Agents.*

Before you create a WebSphere MQ baseline, you can add the **QUEUE_REQUEST_MATCHES** and **QUEUE_RESPONSE_MATCHES** properties to the **rules.xml** file. These properties indicate which queues are for requests and which queues are for responses.

The **QUEUE_REQUEST_MATCHES** and **QUEUE_RESPONSE_MATCHES** properties are optional. They are necessary only if the agent cannot determine on its own whether a message is a request or a response.

If you include these properties, the property names must be followed by a colon and the actual queue name. The value of each property is a period followed by an asterisk. For example:

```
property key=QUEUE_REQUEST_MATCHES:ORDERS.REQUEST value=.*
property key=QUEUE_RESPONSE_MATCHES:ORDERS.RESPONSE value=.*
```

WebSphere MQ Baseline Test Cases

WebSphere MQ baseline test cases have the following steps:

- A do-nothing step that reads from a Large Data data set
- A WebSphere MQ step that sends the request
- A WebSphere MQ step that receives the response

The data set contains information that changes for each transaction in the baseline. This information includes the request payload and the response payload.

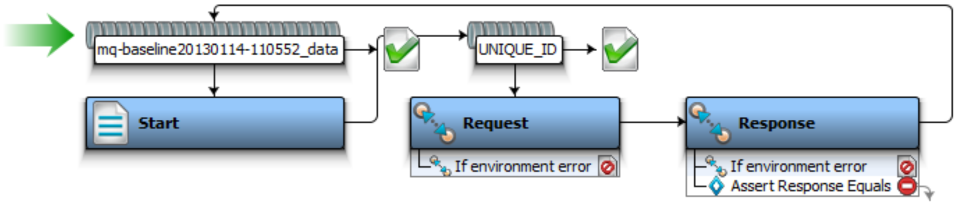
In the WebSphere MQ steps, the client mode is set to native client.

The first WebSphere MQ step sends the request using data from the data set.

The second WebSphere MQ step receives the response. The step includes a Graphical XML Side-by-Side Comparison assertion with the name **Assert Response Equals**. When you run the baseline test case, this assertion verifies that the actual response matches the expected response. If the responses do not match, then the test generates an error.

The test case can also include a Message/Correlation ID Generator data set, which generates a 24-byte code that is used as the correlation ID.

The following graphic shows an example test case.



Binary payloads affect the structure of a WebSphere MQ baseline test case. In this scenario, the test case has the following steps:

- A do-nothing step that reads from a Large Data data set
- A Java Script step that decodes the request payload from Base64 notation into an array of bytes
- A WebSphere MQ step that sends the request
- A WebSphere MQ step that receives the response
- A Java Script step that encodes the response payload from an array of bytes into Base64 notation

The Large Data data set includes two versions of the original request payload: a human-readable version and the Base64 representation.

The Large Data data set includes two versions of the original response payload: a human-readable version and the Base64 representation.

The second Java Script step includes an assertion with the name **Check Base64 Response**. When you run the baseline test case, this assertion verifies that the Base64 representation of the actual response matches the Base64 representation of the expected response. If the responses do not match, then the test generates an error.

To run the test case, do one of the following actions:

- Stage a quick test
- Stage a test case

You can monitor the results in the Consolidated Baseline tab (see page 125).

Note: For information about how to stage a quick test or a test case, see Running Test Cases and Suites in *Using CA Application Test*.

WebSphere MQ Baseline Suites

The test cases in a WebSphere MQ baseline suite are located in the Baselines folder, instead of the Tests folder.

The suite document is located in the Suites folder.

The test cases in a WebSphere MQ baseline suite are similar to WebSphere MQ baseline test cases. However, they do not include a Large Data data set. The data is stored in the steps instead.

For binary payloads, the test cases have the following steps:

- A Parse Text as Response step that contains a Base64 representation of the original request payload
- An Output Log Message step that writes a human-readable version of the original request payload to the log file
- A Java Script step that decodes the actual request payload from Base64 notation into an array of bytes
- A WebSphere MQ step that sends the request
- A WebSphere MQ step that receives the response
- A Parse Text as Response step that contains a Base 64 representation of the original response payload
- An Output Log Message step that writes a human-readable version of the original response payload to the log file
- A Java Script step that encodes the actual response payload from an array of bytes into Base64 notation

When you run the suite, the outcome of each test is shown in the Baseline Results panel (see page 127).

Note: For information about how to run a suite, see Run a Test Suite in *Using CA Application Test*.

Generic Baselines

The term *generic baseline* refers to baselines that are generated for protocols other than the following protocols:

- EJB
- JMS
- REST
- TIBCO ActiveMatrix BusinessWorks
- TIBCO Enterprise Message Service (EMS)
- Web HTTP
- Web service
- webMethods Integration Server
- WebSphere MQ

This section contains the following topics:

Generate a Generic Baseline

Use the following procedure to generate a generic baseline.

Follow these steps:

1. Add one or more transaction frames to the shelf for a protocol other than the protocols listed in Generic Baselines (see page 121).
2. Open the shelf.
3. Click Create Baseline.
4. To change the default name, select the name and make your edits, then click ✓ to save.
5. Click the downward-facing arrow.
6. If the creation mode fields are available, select an option:

   Consolidated

   Create a baseline that contains a single test case and a data set.

   Expanded

   Create a baseline that contains a suite of tests (one for each transaction) and a suite document for running the tests.
7. Specify the type of step that is included in the test case:

   Use Application Test Steps

   The test case includes a step that corresponds to the transaction frame that you added to the shelf. For example, selecting a SOAP frame results in a test case that includes a Web Service Execution (XML) step.

   Use Transaction Frame Step

   The test case includes an Execute Transaction Frame step.
8. (Optional) Configure the baseline to use magic dates.

   Apply magic dates to test cases

   Converts the date strings in the baseline test case or suite to variable definition strings. For example, instead of a string that contains a specific date and time, the string can contain a function that specifies seven days from the current date and time. This behavior is equivalent to the **doDateDeltaFromCurrent** form of magic dates in CA Service Virtualization. If this option is unsupported, the option does not appear.
9. (Optional) Specify a test case to be used as a template for the baseline.
10. Click Create.
11. Select the project where the baseline will be created.
12. Click Create.

Generic Baseline Test Cases

Generic baseline test cases have a test step that reads from a Large Data data set. The test step is one of the following:

- A step that corresponds to the node that you selected in the path graph
- An Execute Transaction Frame step

If the test case has a step that corresponds to the node that you selected in the path graph, then the data set contains such information as the URL, request body, expected response, and response code.

If the test case has an Execute Transaction Frame step, then the data set contains the agent name, transaction frame, and the expected response.

By default, the data set is local, instead of global.

The test step includes an assertion with the name **Assert Response Value Equals**. When you run the baseline test case, this assertion verifies that the actual response value matches the expected response value. If the values do not match, then the test generates an error.

To run the test case, do one of the following actions:

- Stage a quick test
- Stage a test case

You can monitor the results in the Consolidated Baseline tab (see page 125).

Note: For information about how to stage a quick test or a test case, see Running Test Cases and Suites in *Using CA Application Test*.

Generic Baseline Suites

The test cases in a generic baseline suite are located in the Baselines folder, instead of the Tests folder.

The suite document is located in the Suites folder.

The test cases in a generic baseline suite are similar to generic baseline test cases. However, they do not include a Large Data data set. The data is stored in the test step instead.

When you run the suite, the outcome of each test is shown in the Baseline Results panel (see page 127).

Note: For information about how to run a suite, see Run a Test Suite in *Using CA Application Test*.

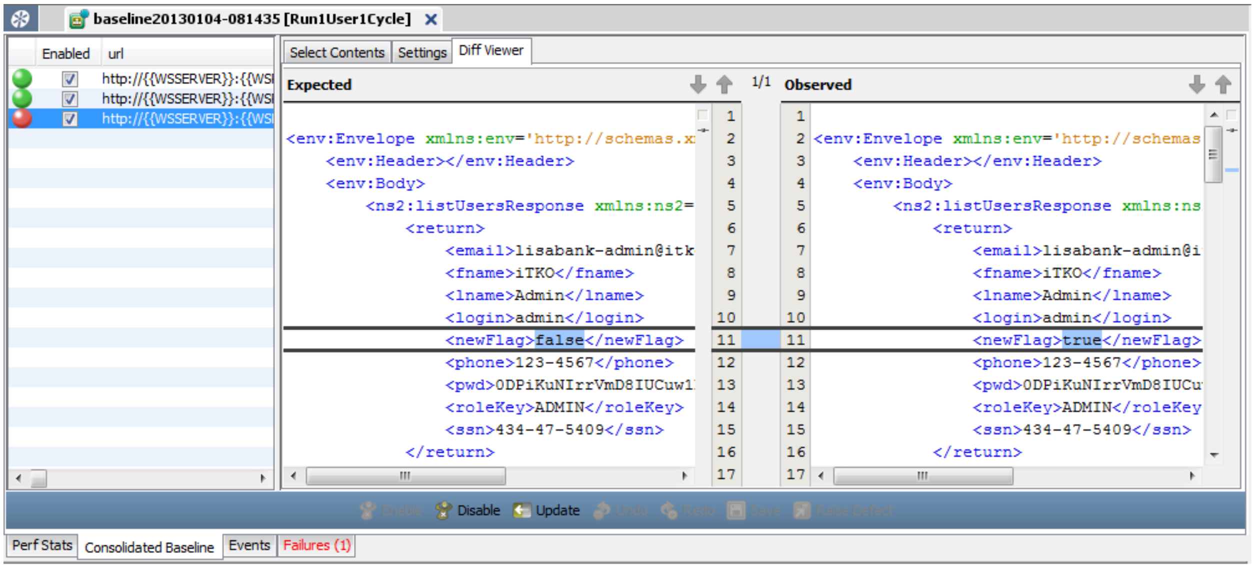
Consolidated Baseline Tab

The Consolidated Baseline tab in DevTest Workstation lets you monitor and update baseline test cases at run time.

When you do either of the following tasks, the Consolidated Baseline tab appears in the Test Monitor window:

- Perform a quick test on a baseline test case
- Stage and run a baseline test case

The following graphic shows the Consolidated Baseline tab for a baseline test case that contains three scenarios. In the test run, the last scenario failed. The diff viewer shows where the issue occurred in the last scenario.



baseline20130104-081435 [Run1User1Cycle]

	Enabled	url
🟢	☑	http://{{WSSERVER}}:{{WSI
🟢	☑	http://{{WSSERVER}}:{{WSI
🔴	☑	http://{{WSSERVER}}:{{WSI

Expected

```
<env:Envelope xmlns:env='http://schemas.x
    <env:Header></env:Header>
    <env:Body>
        <ns2:listUsersResponse xmlns:ns2=
            <return>
                <email>lisabank-admin@itk
                <fname>iTKO</fname>
                <lname>Admin</lname>
                <login>admin</login>
                <newFlag>false</newFlag>
                <phone>123-4567</phone>
                <pwd>0DPiKuNIrrVmD8IUCuw1
                <roleKey>ADMIN</roleKey>
                <ssn>434-47-5409</ssn>
            </return>
```

Observed

```
<env:Envelope xmlns:env='http://schemas
    <env:Header></env:Header>
    <env:Body>
        <ns2:listUsersResponse xmlns:ns
            <return>
                <email>lisabank-admin@i
                <fname>iTKO</fname>
                <lname>Admin</lname>
                <login>admin</login>
                <newFlag>true</newFlag>
                <phone>123-4567</phone>
                <pwd>0DPiKuNIrrVmD8IUCu
                <roleKey>ADMIN</roleKey
                <ssn>434-47-5409</ssn>
            </return>
```

The left portion of the tab contains the rows of the baseline test case's Large Data data set. Each row includes a status icon:

- The color gray indicates that the scenario has not yet been run.
- The color green plus a white arrow indicates that the scenario is running.
- The color green indicates that the scenario succeeded.
- The color red indicates that the scenario failed.
- The color orange indicates that the scenario stopped.
- The color yellow indicates that the expected response was updated with the actual response.

After the baseline test case is run, you can select a row to display the results. If the scenario succeeded or failed, the diff viewer shows the expected response and the actual response. Any differences are highlighted. If the scenario stopped, the cycle history appears instead.

You can use the Settings tab to change the comparison options.

Note: For information about the comparison options, see the documentation for the Graphical XML Side-by-Side Comparison assertion in *Using CA Application Test.*

The toolbar at the bottom contains the following buttons:

Enable

Includes the scenario in subsequent test runs. Be sure to click Save afterward.

Disable

Prevents the scenario from running in subsequent test runs. Be sure to click Save afterward.

Update

Updates the expected response with the actual response. You can perform this action when a scenario failed not because of a functional error but because the system under test has changed. Be sure to click Save afterward.

Undo

Reverses the most recent change.

Redo

Performs the most recent change again.

Save

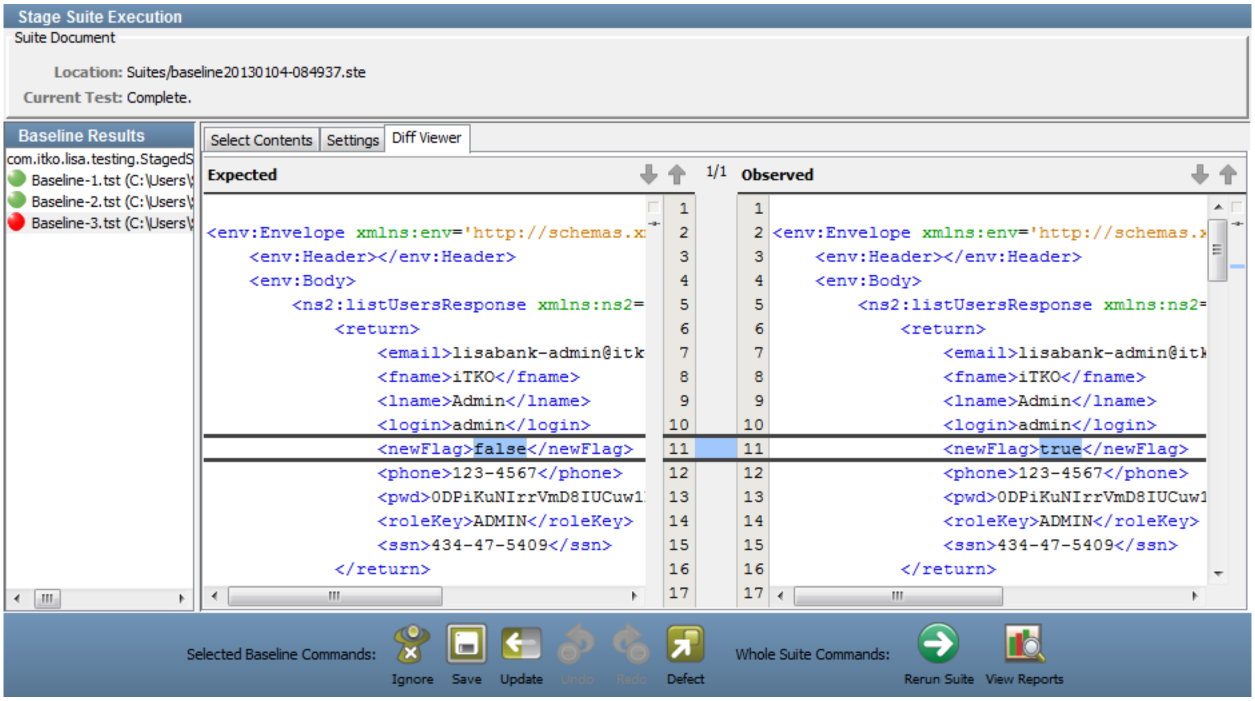
Saves any changes that were made.

Baseline Results Panel

The Baseline Results panel in DevTest Workstation lets you monitor and update baseline suites at run time.

When you run a baseline suite, the Baseline Results panel opens in the Stage Suite Execution tab.

The following graphic shows the Baseline Results panel for a baseline suite that contains three tests. In the suite run, the last test failed. The diff viewer shows where the issue occurred in the last test.



Location: Suites/baseline20130104-084937.ste

Current Test: Complete.

com.itko.lisa.testing.StagedS
- 🟢 Baseline-1.tst (C:\Users\
- 🟢 Baseline-2.tst (C:\Users\
- 🔴 Baseline-3.tst (C:\Users\

Expected

```

<env:Envelope xmlns:env='http://schemas.x
    <env:Header></env:Header>
    <env:Body>
        <ns2:listUsersResponse xmlns:ns2=
            <return>
                <email>lisabank-admin@itk
                <fname>iTKO</fname>
                <lname>Admin</lname>
                <login>admin</login>
                <newFlag>false</newFlag>
                <phone>123-4567</phone>
                <pwd>0DPiKuNIrrVmD8IUCuw1
                <roleKey>ADMIN</roleKey>
                <ssn>434-47-5409</ssn>
            </return>
```

Observed

```

<env:Envelope xmlns:env='http://schemas.x
    <env:Header></env:Header>
    <env:Body>
        <ns2:listUsersResponse xmlns:ns2=
            <return>
                <email>lisabank-admin@itk
                <fname>iTKO</fname>
                <lname>Admin</lname>
                <login>admin</login>
                <newFlag>true</newFlag>
                <phone>123-4567</phone>
                <pwd>0DPiKuNIrrVmD8IUCuw1
                <roleKey>ADMIN</roleKey>
                <ssn>434-47-5409</ssn>
            </return>
```

Each test includes a status icon:

- The color green indicates that the test succeeded.
- The color red indicates that the test failed (that is, an assertion fired that failed the test).
- The color orange indicates that the test stopped. For example, a test step threw an unexpected exception.

After the baseline suite is run, you can select a test to display the results. If the test succeeded or failed, the diff viewer shows the expected response and the actual response. Any differences are highlighted. If the test stopped, the cycle history appears instead.

If a test failed, you can perform one of the following actions:

- Select the test, right-click a row in the Diff Viewer tab, and click Ignore. In subsequent runs of the suite, the row is skipped for all tests in the suite.
- Select the test and click the Ignore icon at the bottom. In subsequent runs of the suite, the test does not run.
- Select the test and click the Update icon at the bottom. The expected response is updated with the actual response.

The toolbar at the bottom contains the following whole suite commands:

Rerun Suite

Runs the suite again.

View Reports

Displays the run statistics in the Reporting Console.

Chapter 8: Creating Virtual Services

CAI lets you create virtual services from transactions in the CAI database.

When you generate the transactions, ensure that the capture levels (see page 26) for the appropriate protocols are set to Full Data.

You cannot create virtual services for transaction frames that have a category of GUI.

Notes:

- This feature requires a separate license.
- For detailed information about service virtualization, see Using CA Service Virtualization.

This section contains the following topics:

Consolidation of Transactions When Creating Virtual Services

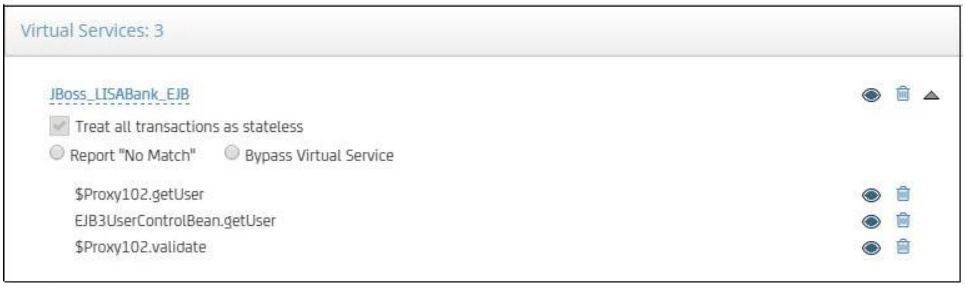
When creating virtual services for transactions with the same category, CAI consolidates the transactions into one virtual service per category. For example, if there are three EJB transactions, only one virtual service is created. You can view all the transactions that were consolidated into a virtual service in the shelf. Only single and merged transactions can be consolidated. All occurred transaction frames cannot be consolidated.

The consolidated virtual service name is the category name when all the transactions are consolidated into one virtual service.

When all the category frames for one agent are consolidated, the consolidated virtual service name is the category name and the agent name.

To display the transactions that were consolidated into the single virtual service, from the shelf click Create VS and click ▼.

The following graphic displays a consolidated EJB virtual service that is created from three EJB transactions in the shelf:



Virtual Services: 3

JBoss_LISABank_EJB

☑ Treat all transactions as stateless

○ Report "No Match" ○ Bypass Virtual Service

$Proxy102.getUser
EJB3UserControlBean.getUser
$Proxy102.validate

Notice that the transactions **$Proxy102.getUser**, **EJB3UserControlBean.getUser**, and **$Proxy102.validate** were consolidated into one virtual service and named **JBoss_LISABank_EJB**.

Note: You have the option of deleting any of the consolidated EJB transactions before creating the virtual service.

The following protocols are supported for consolidation when creating a virtual service:

- REST: one virtual service
- All SOAP: one virtual service
- All HTTP: one virtual service
- EJB: one virtual service per agent
- JDBC: one virtual service per agent
- JCA: one virtual service per agent
- TIBCO: one virtual service per agent

Note: JMS and SAP protocols are not supported.

See the Add All Occurred Transactions to the Shelf (see page 55) topic for more information about all occurred transactions. For more information about merged transactions, see the Merge Repeated Paths (see page 44) topic.

Create Virtual Services from EJB Transactions

Use the following procedure to create a virtual service from a set of EJB transactions in the CAI database.

All methods of the same EJB are virtualized.

A virtual service includes the responses that are sent for unknown conversational requests and unknown stateless requests. When you create a virtual service, you can configure the body of these responses. The following list describes the options:

Report "No Match"

Causes an exception to be raised in the virtualized application.

Bypass Virtual Service

Allows the original request to pass straight through, as if the class and method were not virtualized at all.

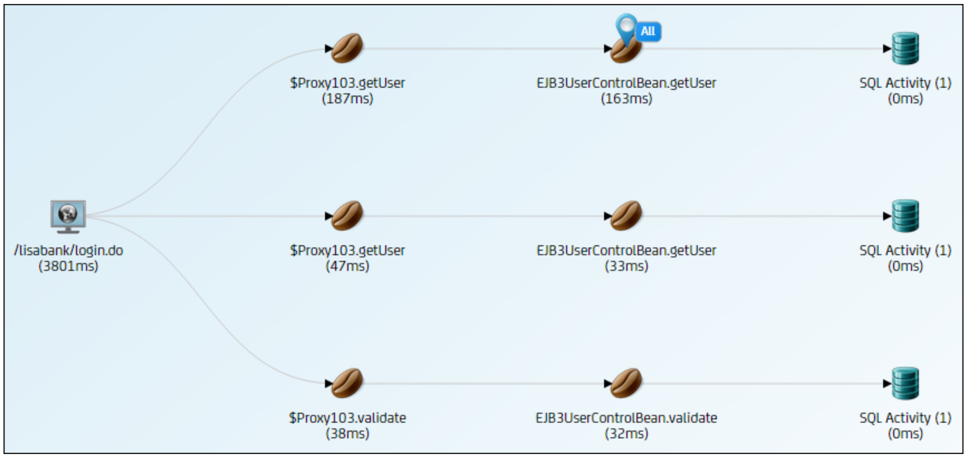
Follow these steps:

1. Add an EJB transaction frame to the shelf.
2. Open the shelf.
3. Click Create VS.
4. To change the default name, select the name and make your edits, then click ✓ to save.
5. If you want all transactions to be treated as stateless, ensure that the check box is selected.
6. Configure the response for unknown requests.
7. To view the consolidated transactions, click ▼.
8. To delete a consolidated transaction, click 🗑.
9. Click Create.
10. Select the project where the virtual service will be created.
11. Click Create.

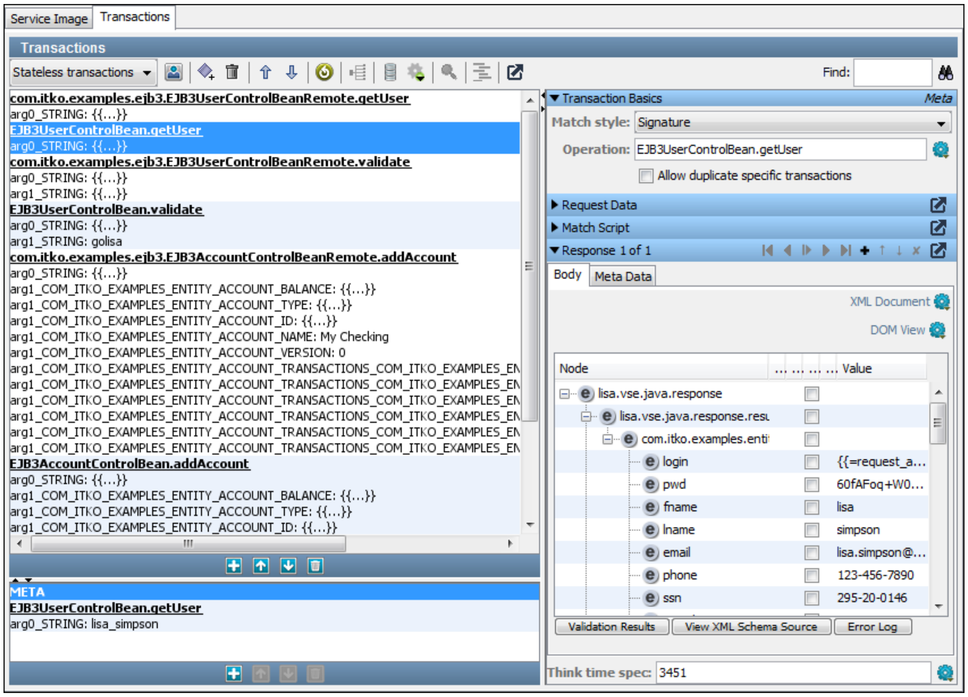
Note: For more information about consolidated transactions, see the Consolidation of Transactions When Creating Virtual Services (see page 130) topic.

Example: EJB User Control Bean

The following graphic shows a path graph that includes EJB components. The Shelve All Occurred Transactions action has been applied to one of the **getUser()** components.



The following graphic shows the service image that was generated. The service image contains a stateless transaction. Notice that multiple methods are virtualized.



```
com.itko.examples.ejb3.EJB3UserControlBeanRemote.getUser
arg0_STRING: {{...}}
EJB3UserControlBean.getUser
arg0_STRING: {{...}}
com.itko.examples.ejb3.EJB3UserControlBeanRemote.validate
arg0_STRING: {{...}}
arg1_STRING: {{...}}
EJB3UserControlBean.validate
arg0_STRING: {{...}}
arg1_STRING: golisa
com.itko.examples.ejb3.EJB3AccountControlBeanRemote.addAccount
arg0_STRING: {{...}}
arg1_COM_ITKO_EXAMPLES_ENTITY_ACCOUNT_BALANCE: {{...}}
arg1_COM_ITKO_EXAMPLES_ENTITY_ACCOUNT_TYPE: {{...}}
arg1_COM_ITKO_EXAMPLES_ENTITY_ACCOUNT_ID: {{...}}
arg1_COM_ITKO_EXAMPLES_ENTITY_ACCOUNT_NAME: My Checking
arg1_COM_ITKO_EXAMPLES_ENTITY_ACCOUNT_VERSION: 0
arg1_COM_ITKO_EXAMPLES_ENTITY_ACCOUNT_TRANSACTIONS_COM_ITKO_EXAMPLES_EN
arg1_COM_ITKO_EXAMPLES_ENTITY_ACCOUNT_TRANSACTIONS_COM_ITKO_EXAMPLES_EN
arg1_COM_ITKO_EXAMPLES_ENTITY_ACCOUNT_TRANSACTIONS_COM_ITKO_EXAMPLES_EN
arg1_COM_ITKO_EXAMPLES_ENTITY_ACCOUNT_TRANSACTIONS_COM_ITKO_EXAMPLES_EN
arg1_COM_ITKO_EXAMPLES_ENTITY_ACCOUNT_TRANSACTIONS_COM_ITKO_EXAMPLES_EN
arg1_COM_ITKO_EXAMPLES_ENTITY_ACCOUNT_TRANSACTIONS_COM_ITKO_EXAMPLES_EN
EJB3AccountControlBean.addAccount
arg0_STRING: {{...}}
arg1_COM_ITKO_EXAMPLES_ENTITY_ACCOUNT_BALANCE: {{...}}
arg1_COM_ITKO_EXAMPLES_ENTITY_ACCOUNT_TYPE: {{...}}
arg1_COM_ITKO_EXAMPLES_ENTITY_ACCOUNT_ID: {{...}}
```

```
META
EJB3UserControlBean.getUser
arg0_STRING: lisa_simpson
```

Match style: Signature

Operation: EJB3UserControlBean.getUser

☐ Allow duplicate specific transactions

Node	... ...	Value
lisa.vse.java.response	☐	
lisa.vse.java.response.resu	☐	
com.itko.examples.enti	☐	
login	☐	{{=request_a...
pwd	☐	60fAFoq+W0...
fname	☐	lisa
lname	☐	simpson
email	☐	lisa.simpson@...
phone	☐	123-456-7890
ssn	☐	295-20-0146

Think time spec: 3451

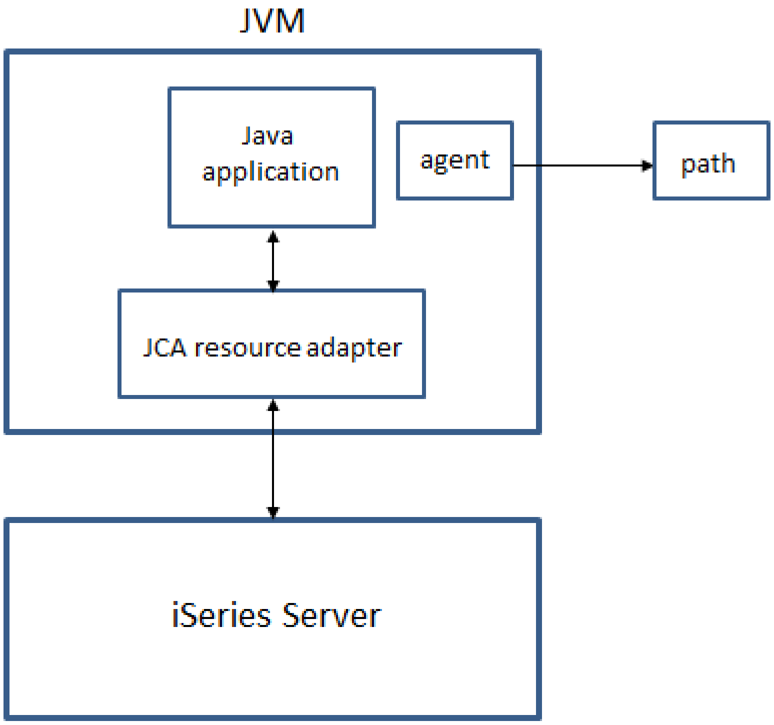
In the virtual service model, the Virtual Java Listener step contains the agent name and the EJB name.

Create Virtual Services from JCA iSeries Transactions

This feature works with the following IBM products:

- IDE: IBM Integration Designer 8.0
- Adapter: IBM WebSphere Adapter for IBM i, version 7.5.0.2

The following graphic shows the architecture of the recording phase. The DevTest Java Agent is configured for a Java application. The Java application uses a JCA resource adapter to issue a program call to an iSeries server. The agent observes the call and generates a corresponding path that can be viewed in the DevTest Portal.



After the recording phase, you generate the virtualization artifacts (see page 135) and deploy the virtual service.

Note: For information about installing and configuring the agent, see *Agents*. For information about deploying the virtual service, see *Using CA Service Virtualization*.

Generate the JCA iSeries Virtualization Artifacts

Use the following procedure to generate the JCA iSeries virtualization artifacts.

A virtual service includes the responses that are sent for unknown conversational requests and unknown stateless requests. When you create a virtual service, you can configure the body of these responses. The following list describes the options:

Report "No Match"

Causes an exception to be raised in the virtualized application.

Bypass Virtual Service

Allows the original request to pass straight through, as if the class and method were not virtualized at all.

Follow these steps:

1. Add a JCA transaction frame to the shelf.
2. Open the shelf.
3. Click Create VS.
4. To change the default name, select the name and make your edits, then click ✓ to save.
5. If you want all transactions to be treated as stateless, ensure that the check box is selected.
6. Configure the response for unknown requests.
7. To view the consolidated transactions, click ▼.
8. To delete a consolidated transaction, click 🗑.
9. Click Create.
10. Select the project where the virtual service will be created.
11. Click Create.

Note: For more information about consolidated transactions, see the Consolidation of Transactions When Creating Virtual Services (see page 130) topic.

Create Virtual Services from JMS Transactions

The result of the first procedure is a raw traffic file that can be imported into the Virtual Service Image Recorder. The benefit of this approach is that it eliminates the need to do proxy recording.

In addition to the request and response bodies, the raw traffic file contains all the connection and queue information in metadata and attributes.

In the second procedure, you use the raw traffic file to create a service image and a virtual service model.

Note: This feature is supported for configurations in which JNDI is used to obtain the JMS connection factory. This feature is also supported for configurations in which a direct API from IBM WebSphere MQ is used to obtain the JMS connection factory.

Note: For detailed information about the Virtual Service Image Recorder, data protocols, magic strings, and deploying virtual services, see *Using CA Service Virtualization*.

To create the raw traffic file from JMS transactions:

1. Add a JMS transaction frame to the shelf.
2. Open the shelf.
3. Click Create VS.
4. To change the default name, select the name and make your edits, then click ✓ to save.
5. Click Create.
6. Select the project where the raw traffic file will be added.
7. Click Create.

To create the service image and virtual service model:

1. From the main menu of DevTest Workstation, select File, New, VS Image, By recording.

   The Virtual Service Image Recorder appears.

2. Do the following steps:

   a. In the Write image to field, enter the fully qualified name of the service image to be created.

   b. In the Import traffic field, browse to and select the raw traffic file in the Data folder.

   c. In the Transport protocol field, select Standard JMS.

   d. In the Model file field, enter the fully qualified name of the virtual service model to be created.

   e. Click Next. The next step prompts you to select the message recording style.

3. If you want to review the request and response queue information, do the following steps:

   a. Select the Review the destinations and transaction tracking mode check box.

   b. If the correlation scheme in the Correlation drop-down list is incorrect, change the value.

   c. Click Next. The next step contains a Destination Info tab and a Connection Setup tab.

   d. The values in the Destination Info and Connection Setup tabs are automatically populated. The Proxy Destination field is set to **N/A** because you are not doing proxy recording. You should not need to update either tab, but you can do so if CAI does not set a correct value. Click Next. The next step contains response information.

   e. The values in this step are automatically populated. The Response Destinations area contains one or more response queues. You should not need to make changes, but you can do so if CAI does not set a correct value. Click Next. The data protocols step appears.

4. Do the following steps:

   a. In the Request Side Data Protocols area, click the plus sign.

   b. Click the left column of the newly added row and select the appropriate data protocol. For XML-based applications, Generic XML Payload Parser is a good generic choice.

   c. If the application responses are not in an XML or text format, a response-side data protocol might be required for VSE to perform magic string substitutions on the response.

   d. Click Next.

5. The next step or steps that appear (if any) depend on the data protocol that you selected. For example, if you selected the Generic XML Payload Parser data protocol, the next step prompts you to create XPaths to form VSE requests. See *Using CA Service Virtualization* as needed to complete the step or steps. The last step indicates that the recorder is performing post-processing on what has been recorded.

6. Click Finish.

   The transactions are saved to a service image, and the virtual service model is created.

To run the virtual service:

1. Shut down the original service.

2. Go to DevTest Workstation and deploy the virtual service model that you created.
3. Run a client application with the virtual service as the service.

Create Virtual Services from REST Transactions

Use the following procedure to create a virtual service from a set of REST transactions in the CAI database.

A virtual service includes the responses that are sent for unknown conversational requests and unknown stateless requests. When you create a virtual service, you can configure the body of these responses. The following list describes the options:

Report "No Match"

Causes an exception to be raised in the virtualized application.

Bypass Virtual Service

Allows the original request to pass straight through, as if the class and method were not virtualized at all.

Follow these steps:

1. Add a REST transaction frame to the shelf.
2. Open the shelf.
3. Click Create VS.
4. To change the default name, select the name and make your edits, then click ✓ to save.
5. Configure the response for unknown requests.
6. To view the consolidated transactions, click ▼.
7. To delete a consolidated transaction, click 🗑.
8. Click Create.
9. Select the project where the virtual service will be created.
10. Click Create.

Note: For more information about consolidated transactions, see the Consolidation of Transactions When Creating Virtual Services (see page 130) topic.

Create Virtual Services from SAP ERPConnect Transactions

ERPConnect is a library that lets you interact with SAP from .NET applications.

You can create a virtual service from a set of ERPConnect transactions that CAI captured.

The general steps are as follows:

1. Install and configure the agent files.
2. Exercise the .NET application.
3. Generate the virtualization artifacts.
4. Deploy the virtual service.

Notes:

- ERPConnect is not included with DevTest Solutions.
- For information about deploying the virtual service, see *Using CA Service Virtualization*.

Install and Configure the Agent Files for SAP ERPConnect

The following versions of the .NET Framework are supported:

- .NET 2.0 Framework, v2.0.50727
- .NET 3.0 Framework, v3.0.4506
- .NET 3.5 Framework, v3.5.21022

Before you start, install Visual C++ Redistributable for Visual Studio 2012 Update 4. As of this writing, you can obtain this component from the Microsoft web site.

The procedure varies depending on which of the following .NET applications you have:

- .NET application that is running in an Internet Information Services (IIS) server
- Stand-alone .NET application

To install and configure the agent files for SAP ERPConnect when you have a .NET application that is running in an IIS server:

1. Copy the following files from the **LISA_HOME\agent** directory to a directory on the computer where the IIS server is located:
   - **LisaAgent.dll**
   - **NativeAgent32.dll**
   - **NativeAgent64.dll**
   - **LisaAgentLauncher.exe**
2. Open a command prompt from the directory where you copied the files.
3. Run the following command:

   ```
   LisaAgentLauncher.exe /i
   ```

   The agent files are registered on the computer.
4. Run the following command:

   ```
   LisaAgentLauncher.exe /iis /name <agent-name> /url
   tcp://<broker-host>:<broker-port>
   ```

   To specify a domain, add the **/domain** option.
5. When the worker process for IIS starts, the agent is enabled.

To install and configure the agent files for SAP ERPConnect when you have a stand-alone .NET application:

1. Copy the following files from the **LISA_HOME\agent** directory to the directory that contains the executable for the .NET application:
   - **LisaAgent.dll**
   - **NativeAgent32.dll**

- **NativeAgent64.dll**
- **LisaAgentLauncher.exe**

2. Open a command prompt from the directory where you copied the files.
3. Run the following command:

   ```
   LisaAgentLauncher.exe /i
   ```

   The agent files are registered on the computer.
4. Configure the environment variables as follows:

   ```
   set COR_PROFILER={BEB45448-91FA-4A7C-BF9A-68AA889DC873}
   set COR_ENABLE_PROFILING=1
   set COR_LISA_AGENT=name=<agent-name>
   ```
5. Restart the .NET application.

Generate the SAP ERPConnect Virtualization Artifacts

Use the following procedure to create a virtual service from a set of ERPConnect transactions in the CAI database.

A virtual service includes the responses that are sent for unknown conversational requests and unknown stateless requests. When you create a virtual service, you can configure the body of these responses. The following list describes the options:

Report "No Match"

Causes an exception to be raised in the virtualized application.

Bypass Virtual Service

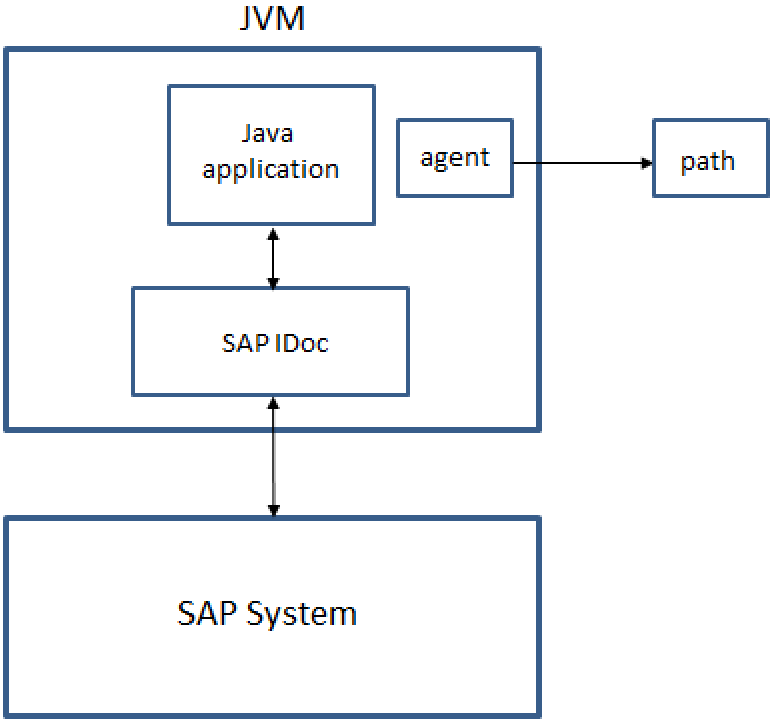
Allows the original request to pass straight through, as if the class and method were not virtualized at all.

Follow these steps:

1. Add an SAP transaction frame to the shelf.
2. Open the shelf.
3. Click Create VS.
4. To change the default name, select the name and make your edits, then click ✓ to save.
5. If you want all transactions to be treated as stateless, ensure that the check box is selected.
6. Configure the response for unknown requests.
7. Click Create.
8. Select the project where the virtual service will be created.
9. Click Create.

Create Virtual Services from SAP IDoc Transactions

The following graphic shows the architecture of the recording phase. The DevTest Java Agent is configured for a Java application. The Java application sends an IDoc message to an SAP system. The agent observes the method calls and generates a corresponding path that can be viewed in the DevTest Portal.



Stateful conversations are not supported.

You do not need to install and configure the agent on the SAP system.

After the recording phase, you generate the virtualization artifacts (see page 144) and deploy the virtual service.

Note: For information about installing and configuring the agent, see *Agents*. For information about deploying the virtual service, see *Using CA Service Virtualization*.

Generate the SAP IDoc Virtualization Artifacts

Use the following procedure to generate the SAP IDoc virtualization artifacts.

A virtual service includes the responses that are sent for unknown conversational requests and unknown stateless requests. When you create a virtual service, you can configure the body of these responses. The following list describes the options:

Report "No Match"

Causes an exception to be raised in the virtualized application.

Bypass Virtual Service

Allows the original request to pass straight through, as if the class and method were not virtualized at all.

Follow these steps:

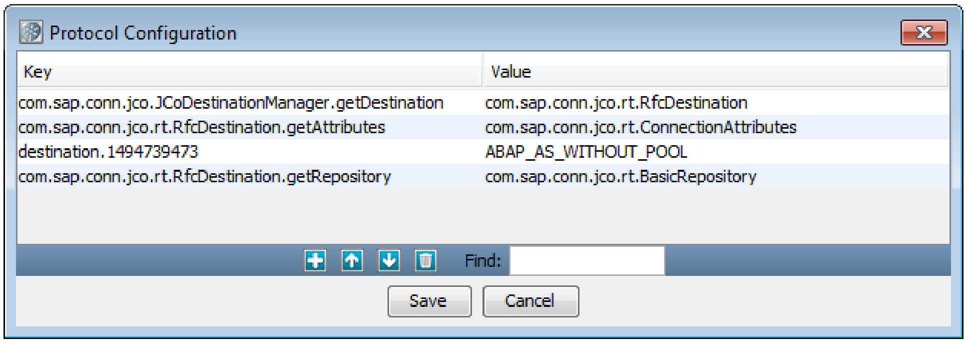
1. Add an SAP transaction frame to the shelf.
2. Open the shelf.
3. Click Create VS.
4. To change the default name, select the name and make your edits, then click ✓ to save.
5. If you want all transactions to be treated as stateless, ensure that the check box is selected.
6. Configure the response for unknown requests.
7. Click Create.
8. Select the project where the virtual service will be created.
9. Click Create.

SAP IDoc Destination Filtering

You can change which SAP IDoc destinations are virtualized.

In the virtual service model that is generated, open the Virtual Java Listener step and double-click **SAP** protocol in the lower right list. The Protocol Configuration dialog opens. This dialog contains a set of key/value pairs.

The following graphic shows the Protocol Configuration dialog.



Key	Value
com.sap.conn.jco.JCoDestinationManager.getDestination	com.sap.conn.jco.rt.RfcDestination
com.sap.conn.jco.rt.RfcDestination.getAttributes	com.sap.conn.jco.rt.ConnectionAttributes
destination.1494739473	ABAP_AS_WITHOUT_POOL
com.sap.conn.jco.rt.RfcDestination.getRepository	com.sap.conn.jco.rt.BasicRepository

The key that begins with **destination.** and ends with a number specifies which destination to virtualize.

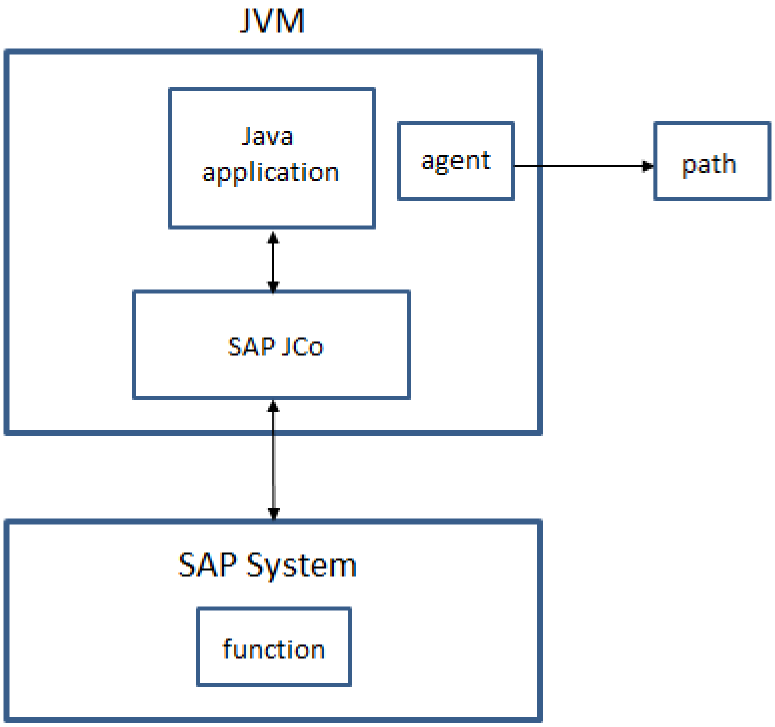
To virtualize multiple destinations, add a destination key for each one. You can use any number, as long as the number is unique in the Protocol Configuration dialog.

To virtualize all destinations, remove the destination key.

You can ignore the keys that resemble Java class names.

Create Virtual Services from SAP JCo Transactions

The following graphic shows the architecture of the recording phase. The DevTest Java Agent is configured for a Java application. The Java application uses SAP JCo to make an RFC call to a function in an SAP system. The agent observes the call and generates a corresponding path that can be viewed in the DevTest Portal.



This feature supports SAP JCo 3.0 only.

This feature supports RFC calls with input and output parameters and table data, and stateful calls.

You do not need to install and configure the agent on the SAP system.

After the recording phase, you generate the virtualization artifacts (see page 147) and deploy the virtual service.

Note: For information about installing and configuring the agent, see *Agents*. For information about deploying the virtual service, see *Using CA Service Virtualization*.

Generate the SAP JCo Virtualization Artifacts

Use the following procedure to generate the SAP JCo virtualization artifacts.

A virtual service includes the responses that are sent for unknown conversational requests and unknown stateless requests. When you create a virtual service, you can configure the body of these responses. The following list describes the options:

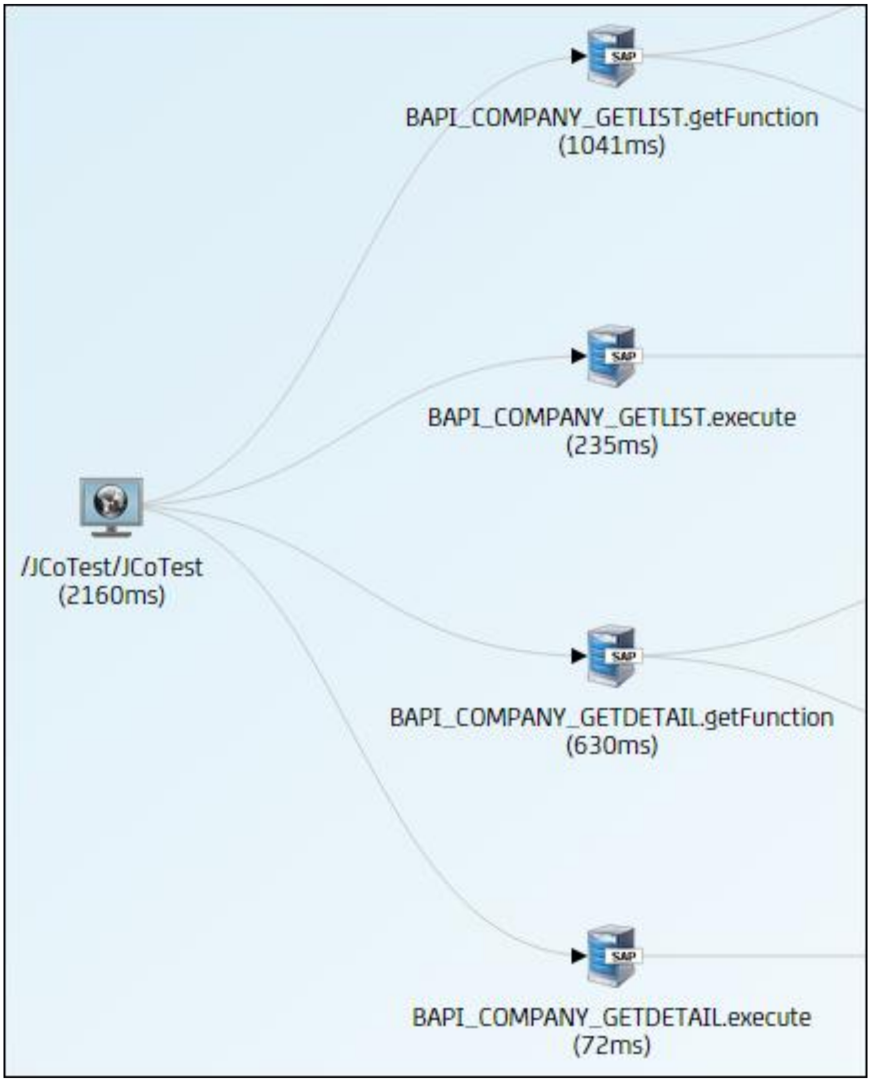
Report "No Match"

Causes an exception to be raised in the virtualized application.

Bypass Virtual Service

Allows the original request to pass straight through, as if the class and method were not virtualized at all.

The following graphic shows a path graph that includes SAP frames. This path graph illustrates the standard SAP JCo pattern of getting a function and then executing the function.


BAPI_COMPANY_GETLIST.getFunction
(1041ms)
BAPI_COMPANY_GETLIST.execute
(235ms)
/JCoTest/JCoTest
(2160ms)
BAPI_COMPANY_GETDETAIL.getFunction
(630ms)
BAPI_COMPANY_GETDETAIL.execute
(72ms)

When the virtualization artifacts are generated, CAI checks the destination name of the selected SAP component. CAI then searches all of the visible paths for SAP components that have this destination. These SAP components are used to create the virtual service.

To view the destination name of the selected SAP component, click the XML tab in the transaction details dialog.

Follow these steps:

1. Add an SAP transaction frame to the shelf.
2. Open the shelf.
3. Click Create VS.

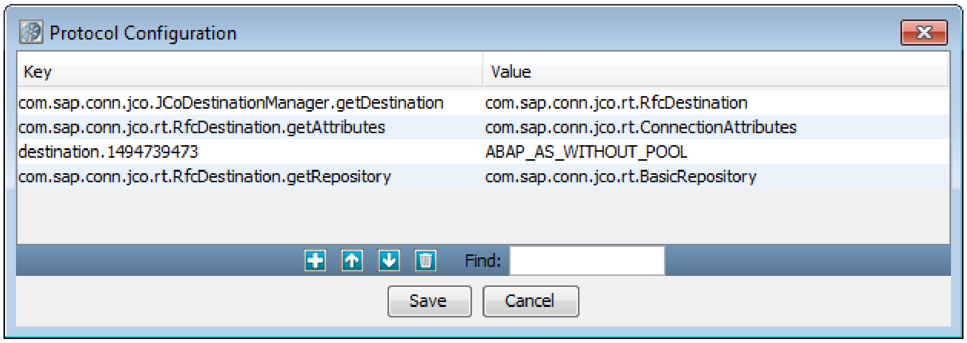
4. To change the default name, select the name and make your edits, then click ✓ to save.
5. If you want all transactions to be treated as stateless, ensure that the check box is selected.
6. Configure the response for unknown requests.
7. Click Create.
8. Select the project where the virtual service will be created.
9. Click Create.

SAP JCo Destination Filtering

You can change which SAP JCo destinations are virtualized.

In the virtual service model that is generated, open the Virtual Java Listener step and double-click **SAP** protocol in the lower right list. The Protocol Configuration dialog opens. This dialog contains a set of key/value pairs.

The following graphic shows the Protocol Configuration dialog.



Protocol Configuration

Key	Value
com.sap.conn.jco.JCoDestinationManager.getDestination	com.sap.conn.jco.rt.RfcDestination
com.sap.conn.jco.rt.RfcDestination.getAttributes	com.sap.conn.jco.rt.ConnectionAttributes
destination.1494739473	ABAP_AS_WITHOUT_POOL
com.sap.conn.jco.rt.RfcDestination.getRepository	com.sap.conn.jco.rt.BasicRepository

The key that begins with **destination.** and ends with a number specifies which destination to virtualize.

To virtualize multiple destinations, add a destination key for each one. You can use any number, as long as the number is unique in the Protocol Configuration dialog.

To virtualize all destinations, remove the destination key.

You can ignore the keys that resemble Java class names.

Create Virtual Services from TIBCO ActiveMatrix BusinessWorks Transactions

You can use CA Continuous Application Insight to virtualize a process in TIBCO ActiveMatrix BusinessWorks. When the virtual service is deployed, it substitutes the behavior of process activities with the responses collected in the service image.

The following approaches are supported:

- Virtualize from DevTest Workstation. You use the Virtual Service Image Recorder to capture traffic and generate the virtual service. You do not need to enable CAI.
- Virtualize from the DevTest Portal. You generate transactions that appear in the DevTest Portal. At any time afterward, you can use the transactions to create the virtual service. CAI must be enabled. This approach provides more flexibility, but has greater overhead than the first approach.

You can generate virtual services that are stateful or stateless. Stateless is recommended. You can control this setting from the Virtual Service Image Recorder and the DevTest Portal.

A virtual service includes the responses that are sent for unknown conversational requests and unknown stateless requests. When you create a virtual service, you can configure the body of these responses. The following list describes the options:

Report "No Match"

Causes an exception to be raised in the virtualized application.

Bypass Virtual Service

Allows the original request to pass straight through, as if the class and method were not virtualized at all.

These procedures assume that the DevTest Java Agent was installed and configured on TIBCO ActiveMatrix BusinessWorks.

Note: For detailed information about installing and configuring the DevTest Java Agent, see *Agents.* For detailed information about using the Virtual Service Image Recorder and deploying virtual services, see *Using CA Service Virtualization.*

To virtualize TIBCO ActiveMatrix BusinessWorks from DevTest Workstation:

1. Go to DevTest Workstation and start the Virtual Service Image Recorder.

   You are prompted to provide basic information.

2. Specify the names of the service image and virtual service model.
3. Set the transport protocol to Java.
4. Click Next.

   You are prompted to select the Java classes to virtualize.

5. Select the agent and move it into the Connected Agents list.
6. Expand the Protocols arrow and move the TIBCO BW protocol into the right pane.
7. Click Next. If necessary, trigger the execution of the TIBCO processes that you want to record. As the processes run, the agent records the activities.
8. When the recording has completed, click Next and perform the remaining steps in the Virtual Service Image Recorder. You do not need to select a data protocol.

   A service image and virtual service model are created.
9. Deploy and start the virtual service model.

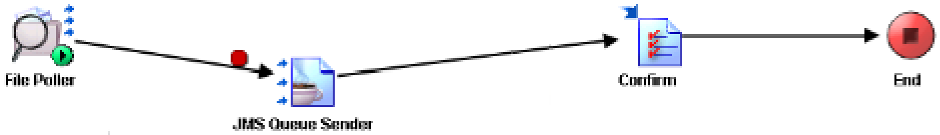
To virtualize TIBCO ActiveMatrix BusinessWorks from the DevTest Portal:

1. Trigger the execution of the TIBCO processes.
2. Add a TIBCO transaction frame to the shelf. You can virtualize the entire conversation by selecting the leftmost frame. You can virtualize a specific activity by selecting the frame that includes the activity name.
3. Open the shelf.
4. Click Create VS.
5. To change the default name, select the name and make your edits, then click ✓ to save.
6. If you want all transactions to be treated as stateless, ensure that the check box is selected.
7. Configure the response for unknown requests.
8. To view the consolidated transactions, click ▼.
9. To delete a consolidated transaction, click 🗑.
10. Click Create.
11. Select the project where the virtual service will be created.
12. Click Create.

Note: For more information about consolidated transactions, see the Consolidation of Transactions When Creating Virtual Services (see page 130) topic.

Example: Virtualize an Add User Process

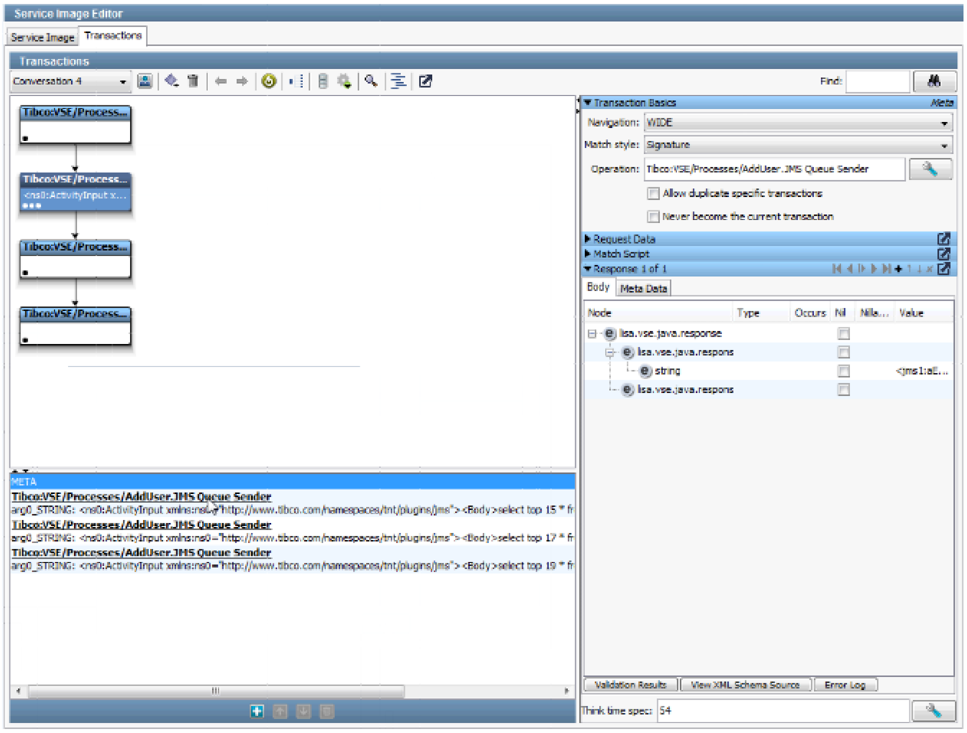
The following graphic shows a process definition in TIBCO Designer. The process contains the following activities: File Poller, JMS Queue Sender, Confirm, and End.



The File Poller activity monitors a text file. When the file is changed, the activity starts the process.

The JMS Queue Sender activity sends a message to the specified queue.

The following graphic shows a service image that the Virtual Service Image Recorder generated. A conversation with four nodes appears in the Transactions tab. The four nodes correspond to the four activities in the process definition.



Conversation 4

Navigation: WIDE

Match style: Signature

Operation: Tibco:VSE/Processes/AddUser.JMS Queue Sender

□ Allow duplicate specific transactions

□ Never become the current transaction

Node	Type	Occurs	Nil	Nilla...	Value
lisa.vse.java.response			□		
lisa.vse.java.respons			□		
string			□		<jms1:aE...
lisa.vse.java.respons			□		

Think time spec: 54

```
Tibco:VSE/Processes/AddUser.JMS Queue Sender
arg0_STRING: <ns0:ActivityInput xmlns:ns0="http://www.tibco.com/namespaces/tnt/plugins/jms"><Body>select top 15 * fr
Tibco:VSE/Processes/AddUser.JMS Queue Sender
arg0_STRING: <ns0:ActivityInput xmlns:ns0="http://www.tibco.com/namespaces/tnt/plugins/jms"><Body>select top 17 * fr
Tibco:VSE/Processes/AddUser.JMS Queue Sender
arg0_STRING: <ns0:ActivityInput xmlns:ns0="http://www.tibco.com/namespaces/tnt/plugins/jms"><Body>select top 19 * fr
```

Each node in the conversation includes an operation field. The value of this field consists of the fully qualified process name and the activity name. In the preceding graphic, the value for the selected node is **Tibco:VSE/Processes/AddUser.JMS Queue Sender**.

The virtual service model is the default for the Java transport protocol.

Create Virtual Services from Web HTTP Transactions

Use the following procedure to create a virtual service from a set of Web HTTP transactions in the CAI database.

A virtual service includes the responses that are sent for unknown conversational requests and unknown stateless requests. When you create a virtual service, you can configure the body of these responses. The following list describes the options:

Report "No Match"

Causes an exception to be raised in the virtualized application.

Bypass Virtual Service

Allows the original request to pass straight through, as if the class and method were not virtualized at all.

Follow these steps:

1. Add a Web HTTP transaction to the shelf.
2. Open the shelf.
3. Click Create VS.
4. To change the default name, select the name and make your edits, then click ✓ to save.
5. Configure the response for unknown requests.
6. To view the consolidated transactions, click ▼.
7. To delete a consolidated transaction, click 🗑.
8. Click Create.
9. Select the project where the virtual service will be created.
10. Click Create.

Note: For more information about consolidated transactions, see the Consolidation of Transactions When Creating Virtual Services (see page 130) topic.

Create Virtual Services from Web Service Transactions

Use the following procedure to create a virtual service from a set of web service transactions in the CAI database.

Web services with attachments are not supported.

A virtual service includes the responses that are sent for unknown conversational requests and unknown stateless requests. When you create a virtual service, you can configure the body of these responses. The following list describes the options:

Report "No Match"

Causes an exception to be raised in the virtualized application.

Bypass Virtual Service

Allows the original request to pass straight through, as if the class and method were not virtualized at all.

Follow these steps:

1. Add a SOAP transaction frame to the shelf.
2. Open the shelf.
3. Click Create VS.
4. To change the default name, select the name and make your edits, then click ✓ to save.
5. Configure the response for unknown requests.
6. To view the consolidated transactions, click ▼.
7. To delete a consolidated transaction, click 🗑.
8. Click Create.
9. Select the project where the virtual service will be created.
10. Click Create.

Note: For more information about consolidated transactions, see the Consolidation of Transactions When Creating Virtual Services (see page 130) topic.

Create Virtual Services from webMethods Transactions

Use the following procedure to create a virtual service from a set of webMethods Integration Server transactions in the CAI database.

webMethods Integration Server includes the concepts of a flow service and a pipeline. A flow service lets you encapsulate a group of services and manage the flow of data among them. The pipeline is a data structure that contains the input and output values for a flow service. You can add steps to a flow service to perform such actions as invoking services and changing data in the pipeline.

The flow service is the unit that is virtualized.

A virtual service includes the responses that are sent for unknown conversational requests and unknown stateless requests. When you create a virtual service, you can configure the body of these responses. The following list describes the options:

Report "No Match"

Causes an exception to be raised in the virtualized application.

Bypass Virtual Service

Allows the original request to pass straight through, as if the class and method were not virtualized at all.

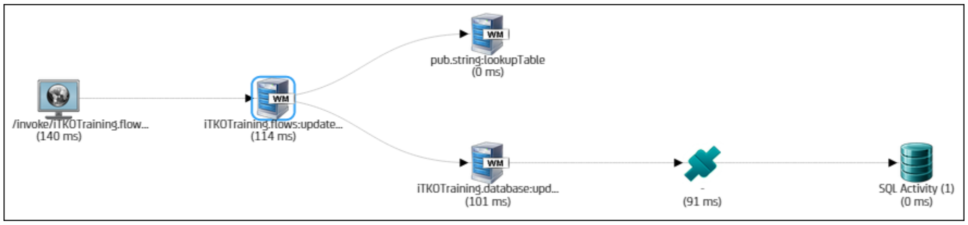
Follow these steps:

1. Add a webMethods Integration Server transaction frame to the shelf.
2. Open the shelf.
3. Click Create VS.
4. To change the default name, select the name and make your edits, then click ✓ to save.
5. If you want all transactions to be treated as stateless, ensure that the check box is selected.
6. Configure the response for unknown requests.
7. To view the consolidated transactions, click ▼.
8. To delete a consolidated transaction, click 🗑.
9. Click Create.
10. Select the project where the virtual service will be created.
11. Click Create.

Note: For more information about consolidated transactions, see the Consolidation of Transactions When Creating Virtual Services (see page 130) topic.

Example: Update User

The following graphic shows a path graph that includes webMethods Integration Server frames. The flow service in this example is named **updateUser**. The purpose is to send updated information for a user to a database.



Note: If you select any of the webMethods Integration Server frames, you can view the pipeline data in the transaction detail dialog.

When a path graph has multiple webMethods Integration Server frame, you must select one frame for virtualization. The decision of which frame to select depends on the system under test. This example could result in the following decisions:

- To ensure that your software interfaces with a webMethods process, select the leftmost webMethods Integration Server frame.
- To remove the database, select the webMethods Integration Server frame that directly calls the database.

Create Virtual Services from WebSphere MQ Transactions

The result of the first procedure is a raw traffic file that can be imported into the Virtual Service Image Recorder. The benefit of this approach is that it eliminates the need to do proxy recording.

In addition to the request and response bodies, the raw traffic file contains all the connection and queue information in metadata and attributes.

If you are working with binary payloads, use an extension data protocol handler to convert the binary requests into an XML form that VSE can handle. You select this data protocol handler in the Virtual Service Image Recorder.

If the response is also binary, you have two options:

- Use an extension data protocol handler to parse the binary responses into a text format for the service image. At runtime, the original binary format is reconstructed for each response.
- Do not use an extension data protocol handler for the response. At runtime, the VSE engine returns the original binary response and is not able to perform magic string substitutions.

In the second procedure, you use the raw traffic file to create a service image and a virtual service model.

Note: For detailed information about the Virtual Service Image Recorder, data protocols, magic strings, and deploying virtual services, see *Using CA Service Virtualization*.

To create the raw traffic file from WebSphere MQ transactions:

1. Add a WebSphere MQ transaction frame to the shelf.
2. Open the shelf.
3. Click Create VS.
4. To change the default name, select the name and make your edits, then click ✓ to save.
5. To view the consolidated transactions, click ▼.
6. To delete a consolidated transaction, click 🗑.
7. Click Create.
8. Select the project where the raw traffic file will be added.
9. Click Create.

Note: For more information about consolidated transactions, see the Consolidation of Transactions When Creating Virtual Services (see page 130) topic.

To create the service image and virtual service model:

1. From the main menu of DevTest Workstation, select File, New, VS Image, By recording.

   The Virtual Service Image Recorder appears.

2. Do the following steps:

   a. In the Write image to field, enter the fully qualified name of the service image to be created.

   b. In the Import traffic field, browse to and select the raw traffic file in the Data folder.

   c. In the Transport protocol field, select IBM MQ Series.

   d. In the Model file field, enter the fully qualified name of the virtual service model to be created.

   e. Click Next. The next step prompts you to select the message recording style.

3. If you want to review the request and response queue information, do the following steps:

   a. Select the Review the queues and transaction tracking mode check box.

   b. If the correlation scheme in the Correlation drop-down list is incorrect, change the value.

   c. Click Next. The next step contains a Destination Info tab and a Connection Setup tab.

   d. The values in the Destination Info and Connection Setup tabs are automatically populated. The Proxy Queue field is set to N/A because you are not doing proxy recording. You should not need to update either tab, but you can do so if CAI does not set a correct value. Click Next. The next step contains response information.

   e. The values in this step are automatically populated. The Response Destinations area contains one or more response queues. You should not need to make changes, but you can do so if CAI does not set a correct value. Click Next. The data protocols step appears.

4. Do the following steps:

   a. In the Request Side Data Protocols area, click the plus sign.

   b. Click the left column of the newly added row and select the appropriate data protocol. For XML-based applications, Generic XML Payload Parser is a good generic choice.

   c. If the application responses are not in an XML or text format, a response-side data protocol might be required for the VSE engine to perform magic string substitutions on the response.

   d. Click Next.

5. The next step or steps that appear (if any) depend on the data protocol that you selected. For example, if you selected the Generic XML Payload Parser data protocol, the next step prompts you to create XPaths to form VSE requests. See *Using CA Service Virtualization* as needed to complete the step or steps. The last step indicates that the recorder is performing post-processing on what has been recorded.

6. Click Finish.

   The transactions are saved to a service image, and the virtual service model is created.

To run the virtual service:

1. Shut down the original service.
2. Go to DevTest Workstation and deploy the virtual service model that you created.
3. Run a client application with the virtual service as the service.

Virtualize Databases with the Web-Based Recorder

You can use the DevTest Java Agent with a web-based recorder to virtualize JDBC-based database traffic. This feature is also known as *agent-based JDBC virtualization*.

Note: JDBC is a chatty protocol. For example, a few clicks in a user interface that retrieves data from a database can result in a huge number of JDBC API calls. This behavior can make it difficult to model properly. If you want to capture and virtualize JDBC database traffic, consider doing so at a higher layer.

The following JDBC drivers and databases are supported:

- Apache Derby
  - Apache Derby 4.2.3 JDBC driver to Apache Derby database
- Oracle
  - Oracle 11*g* JDBC driver to Oracle 11*g* database
  - Oracle 11*g* JDBC driver to Oracle 12*c* database
- IBM DB2:
  - IBM DB2 9.5 JDBC driver to IBM DB2 9.5 database
  - IBM DB2 9.5 JDBC driver to IBM DB2 9.8 database
  - IBM DB2 9.5 JDBC driver to IBM DB2 10.5 database
- Microsoft SQL Server
  - Microsoft JDBC Driver 4.0 to Microsoft SQL Server 2008 R2 database
  - Microsoft JDBC Driver 4.0 to Microsoft SQL Server 2012 database
  - jTDS 1.3.1 JDBC driver to Microsoft SQL Server 2008 R2 database
  - jTDS 1.3.1 JDBC driver to Microsoft SQL Server 2012 database

Before you capture the JDBC traffic, install and configure the agent on the computer where the application makes JDBC calls to the database. The database itself can reside on a separate computer. When you configure the agent, be sure to perform the following actions:

- Set the capture level (see page 26) for the JDBC protocol to Full Data.
- Set the capture level for any protocol above the JDBC protocol to Counts and Paths or to Full Data.

Important! During the playback phase, start the virtual service *before* you start the system under test.

Note: For information about installing the agent, see *Agents*. For information about deploying the virtual service, see *Using CA Service Virtualization*.

Capture the JDBC Traffic

The Virtualize JDBC window in the DevTest Portal includes the following views:

- Entity View
- Conversation View

This procedure assumes that the Entity View is initially displayed.

Note: If you stop a recording and then start a new recording, any queries from the earlier recording are discarded.

Follow these steps:

1. Select Create, Virtualize JDBC in the left navigation menu

   The Virtualize JDBC window opens.

2. Click the Start recording button.
3. Exercise the parts of the application that make the queries that you want to virtualize.

   The Results pane displays each unique query as it is captured.

4. (Optional) Examine the captured queries (see page 161).
5. (Optional) Preview the conversations (see page 162).
6. When you have all the queries and conversations that you are interested in, click the Stop recording button.
7. Click Create VS Artifact.

   The virtual service is created.

Examine the Captured Queries

During the capture of JDBC traffic, you can examine details about the queries that have been captured.

You can also use the filter capability to narrow the scope of the displayed queries.

Note: Filtering does not affect which queries are virtualized.

Follow these steps:

1. To view an instance of a query, select the query in the Results pane.

   The SQL Query pane and the Transactions pane are populated.

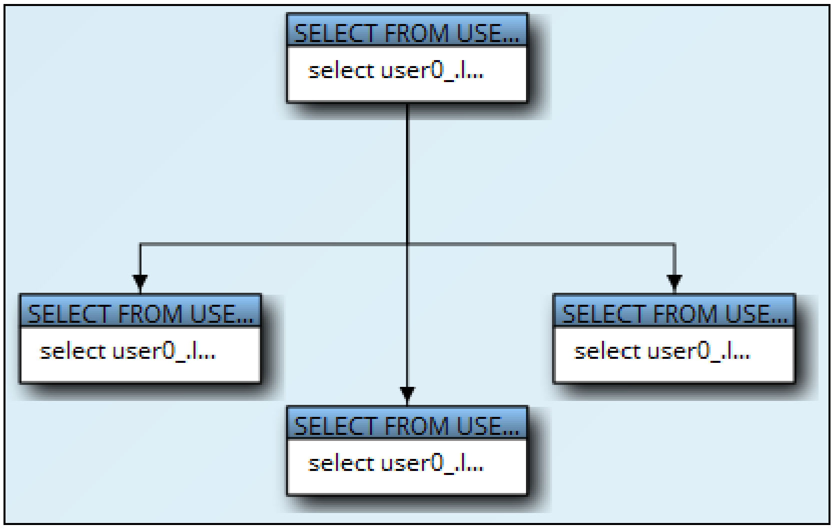
2. To filter the queries, go to the Databases section and select the check box for any database that you want to include.

Preview the Conversations

Virtual services can include both conversations and stateless transactions.

During the capture of JDBC traffic, you can preview the conversations in the virtual service that will be generated.

The following graphic shows an example of a conversation.



You can display a miniature view of the overall structure in the upper right corner. This capability is especially helpful for large conversations.

Follow these steps:

1. In the drop-down list, select Conversation View.
2. To display a miniature view of the overall structure, click Show Outline.
3. To display detailed information about a query in the conversation, click the node.

JDBC Service Image

In agent-based JDBC virtualization, the virtual service that you generate includes a service image.

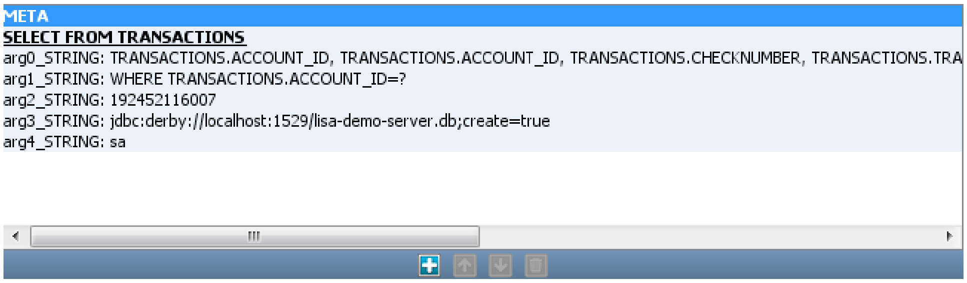
Each transaction in the service image has an operation and a series of arguments.

The operation conveys the main purpose of the SQL statement. For example:

```
SELECT FROM TRANSACTIONS
```

The arguments can include the list of columns, each major clause of the SQL statement, and parameter values that are bound to the SQL statement. The final two arguments are always the connection URL and the database user.

The following graphic shows a transaction. The operation is SELECT FROM TRANSACTIONS. The first argument contains the list of columns. The SQL statement has one parameter value: the account ID.



META

SELECT FROM TRANSACTIONS

```
arg0_STRING: TRANSACTIONS.ACCOUNT_ID, TRANSACTIONS.ACCOUNT_ID, TRANSACTIONS.CHECKNUMBER, TRANSACTIONS.TRA
arg1_STRING: WHERE TRANSACTIONS.ACCOUNT_ID=?
arg2_STRING: 192452116007
arg3_STRING: jdbc:derby://localhost:1529/lisa-demo-server.db;create=true
arg4_STRING: sa
```

Each transaction in the service image also has an XML document that contains the results of executing the SQL statement. The XML document is located in the Response panel of the Service Image Editor.

The results depend on the type of SQL statement:

- If the SQL statement retrieved data, the results contain one or more data sets.
- If the SQL statement updated data, the results contain update counts.

Multiple Data Set Editor

When a transaction is selected in a service image, the Response panel lets you view the results of executing the SQL statement.

You can also edit the data that appears. For example, you might want to change the value in a result set to test how the calling code deals with the data returned from the database.

The default editor in the Response panel is named MultiDataSet Document. This editor consists of the following components:

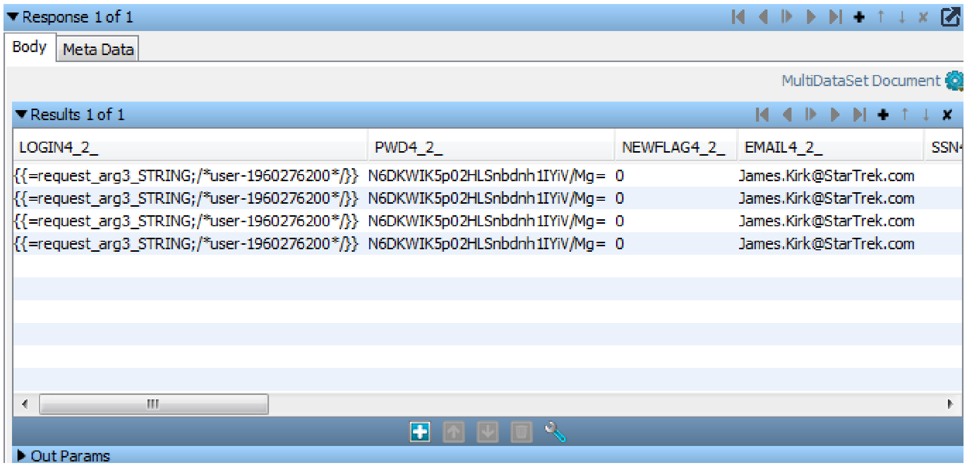
- Results panel
- Out Params panel

Results Panel

The Results panel can contain any number of result sets and any number of update counts. For example, the following contents are all valid:

- One result set
- One update count
- One result set and one update count
- Three result sets

The following graphic shows an example of a result set.



LOGIN4_2_	PWD4_2_	NEWFLAG4_2_	EMAIL4_2_	SSN
{{=request_arg3_STRING;/*user-1960276200*/}}	N6DKWIK5p02HLSnbdnh1IYiV/Mg=	0	James.Kirk@StarTrek.com	
{{=request_arg3_STRING;/*user-1960276200*/}}	N6DKWIK5p02HLSnbdnh1IYiV/Mg=	0	James.Kirk@StarTrek.com	
{{=request_arg3_STRING;/*user-1960276200*/}}	N6DKWIK5p02HLSnbdnh1IYiV/Mg=	0	James.Kirk@StarTrek.com	
{{=request_arg3_STRING;/*user-1960276200*/}}	N6DKWIK5p02HLSnbdnh1IYiV/Mg=	0	James.Kirk@StarTrek.com	

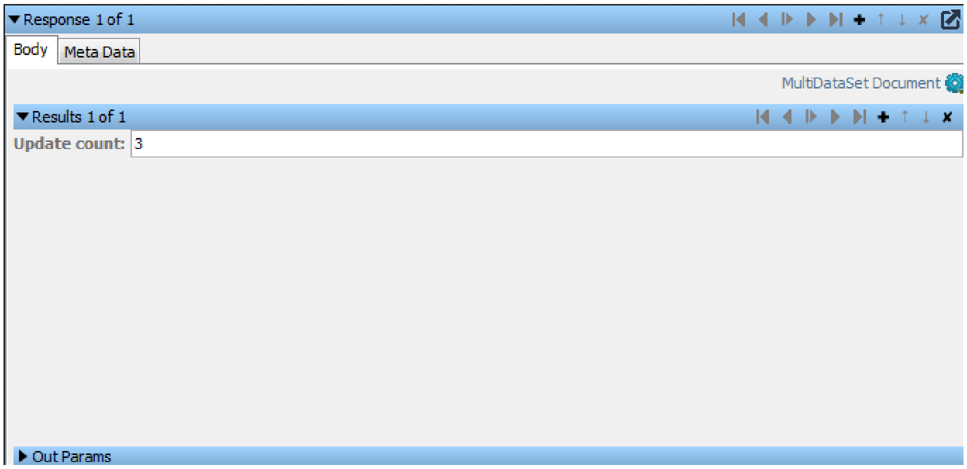
You can add and delete result sets.

In a result set, you can perform the following actions:

- Change a value

- Add a row
- Move a row up or down
- Delete a row
- View and edit column metadata, such as the label and data type

The following graphic shows an example of an update count.



Update count: 3

You can add and delete update counts.

In an update count, you can change the value. The value must be an integer or a property expression.

Out Params Panel

The Out Params panel is used when a stored procedure call has output parameters.

You can add two types of output parameters:

- Simple
- Result set

A simple output parameter has a key and a value.

A result set output parameter has a key and a result set.

The system under test accesses the parameter by an index number or by a string name. You specify the index number or string name in the key.

JDBC Virtual Service Model

In agent-based JDBC virtualization, the virtual service that you generate includes a virtual service model.

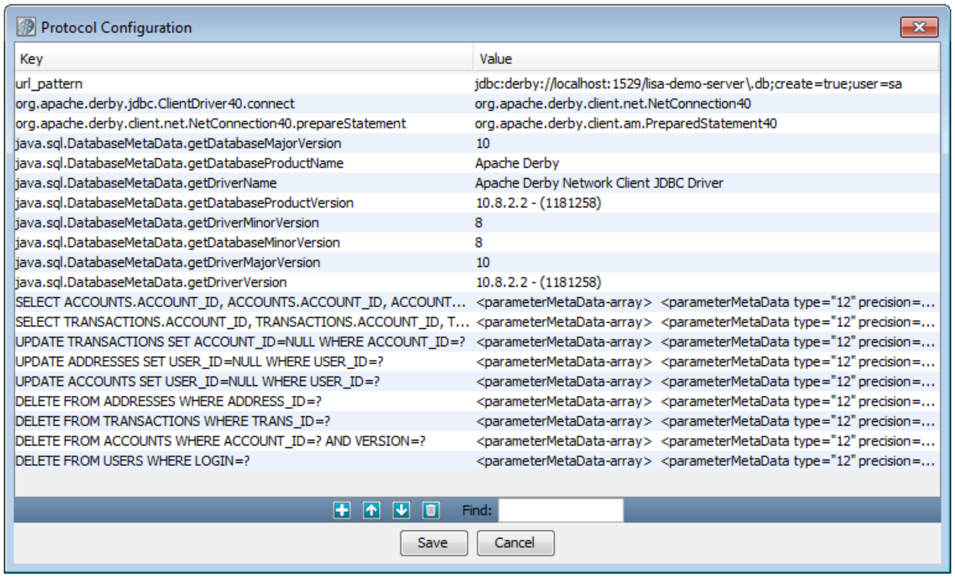
The virtual service model has the same listener and responder steps as in Java VSE:

- Virtual Java Listener
- Virtual Java Responder

To view the protocol configuration information, open the listener step and double-click JDBC protocol in the lower right list. The Protocol Configuration dialog opens. This dialog contains a set of key/value pairs.

Important! Do not delete any of the rows. If you delete a row, the system under test will not work correctly.

The following graphic shows the Protocol Configuration dialog. This example is based on the Apache Derby database.



Key	Value
url_pattern	jdbc:derby://localhost:1529/lisa-demo-server\.db;create=true;user=sa
org.apache.derby.jdbc.ClientDriver40.connect	org.apache.derby.client.net.NetConnection40
org.apache.derby.client.net.NetConnection40.prepareStatement	org.apache.derby.client.am.PreparedStatement40
java.sql.DatabaseMetaData.getDatabaseMajorVersion	10
java.sql.DatabaseMetaData.getDatabaseProductName	Apache Derby
java.sql.DatabaseMetaData.getDriverName	Apache Derby Network Client JDBC Driver
java.sql.DatabaseMetaData.getDatabaseProductVersion	10.8.2.2 - (1181258)
java.sql.DatabaseMetaData.getDriverMinorVersion	8
java.sql.DatabaseMetaData.getDatabaseMinorVersion	8
java.sql.DatabaseMetaData.getDriverMajorVersion	10
java.sql.DatabaseMetaData.getDriverVersion	10.8.2.2 - (1181258)
SELECT ACCOUNTS.ACCOUNT_ID, ACCOUNTS.ACCOUNT_ID, ACCOUNT...	<parameterMetaData-array> <parameterMetaData type="12" precision=...
SELECT TRANSACTIONS.ACCOUNT_ID, TRANSACTIONS.ACCOUNT_ID, T...	<parameterMetaData-array> <parameterMetaData type="12" precision=...
UPDATE TRANSACTIONS SET ACCOUNT_ID=NULL WHERE ACCOUNT_ID=?	<parameterMetaData-array> <parameterMetaData type="12" precision=...
UPDATE ADDRESSES SET USER_ID=NULL WHERE USER_ID=?	<parameterMetaData-array> <parameterMetaData type="12" precision=...
UPDATE ACCOUNTS SET USER_ID=NULL WHERE USER_ID=?	<parameterMetaData-array> <parameterMetaData type="12" precision=...
DELETE FROM ADDRESSES WHERE ADDRESS_ID=?	<parameterMetaData-array> <parameterMetaData type="12" precision=...
DELETE FROM TRANSACTIONS WHERE TRANS_ID=?	<parameterMetaData-array> <parameterMetaData type="12" precision=...
DELETE FROM ACCOUNTS WHERE ACCOUNT_ID=? AND VERSION=?	<parameterMetaData-array> <parameterMetaData type="12" precision=...
DELETE FROM USERS WHERE LOGIN=?	<parameterMetaData-array> <parameterMetaData type="12" precision=...

The **url_pattern** key specifies the connection URL that triggers virtualization at playback. If the value ends with a semicolon followed by the text **user=** and a database user, the virtualization is restricted to that database user. You can add key/value pairs for additional database users. The key for each user must begin with **url_pattern_**.

The second key is composed of the driver class name and the **connect** method. The value is the name of the connection class.

The third key is composed of the connection class name and the **prepareStatement** method. The value is the name of the prepared statement class.

Notice how the second and third keys create a "chain" of driver to connection to statement.

The keys that begin with **java.sql.DatabaseMetaData** contain name and version information for:

- The database driver
- The database that it was talking to when the SQL traffic was captured

The keys at the bottom contain a mapping of a specific SQL statement and the parameter metadata that the statement represents.

Match Exceptions

The **Throw exceptions on no match** property lets you control what happens when the virtual service model does not have an answer for a SQL statement. By default, the property is enabled.

- If the property is enabled, the agent throws a SQL exception noting the SQL statement that failed to be virtualized. The exception also suggests that you capture more JDBC frames and add them to the service image. This action involves generating new virtualization artifacts and using the service image combine function in either DevTest Workstation or the **ServiceImageManager** command-line tool.
- If the property is disabled, the agent provides appropriate empty results with a best guess as to column definitions.

You can configure this property from the Agents window (see page 19) of the DevTest Portal. The property appears in the Settings tab.

Database Virtualization Troubleshooting

Symptom:

I am trying to virtualize an Oracle database. When I exercise the JDBC transaction in playback mode, the transaction count does not increase in the VSE Console.

Solution:

Open the log file for the agent. Search for the following exception:

```
java.lang.RuntimeException: com.itko.javassist.NotFoundException: 
oracle.xdb.XMLType
   at 
com.itko.javassist.CtClassType.getClassFile2(CtClassType.java:202)
   at 
com.itko.javassist.CtClassType.getSuperclass(CtClassType.java:746)
```

If you find the exception, do the following steps:

1. Stop the application and the virtual service.
2. Start the virtual service. Make sure that it is in Running state.
3. Start the test application.
4. Exercise the JDBC transaction again.

   The transaction count should increase.

Alternately, you can download the **xdb.jar** file and can place it in the classpath of the application.

Chapter 9: Documenting Transactions for Testing

Documenting business transactions generates visibility into the system architecture and the flow of data. This visibility exposes what is happening in the backend code and database.

You document a test by recording test transactions and analyzing the generated backend calls. You can compare one or more tests. Transactions are marked with a blue pin and display in a graphical view to identify the exceptions that have occurred during the recording. The name of the recorded test case displays in the Points of Interest list in the Home page and in the Explore Defects window.

You document transactions for testing from the Document Transactions window. You can perform the following tasks:

- Record test transactions (see page 170)
- Search the CAI database for saved transaction recordings (see page 171)
- View and analyze recordings (see page 171) in list or graphical view
- Create documentation from recorded transactions (see page 172)

This section contains the following topics:

How to Document a Transaction for a Manual Test

This scenario takes you through the steps of documenting a transaction for a manual test.

1. Record Your Test Transactions (see page 170)
2. Search for Recorded Test Transactions (see page 171)
3. View and Analyze Recorded Test Transactions (see page 171)
4. Create Documentation from Recorded Transactions (see page 172)

Record Your Test Transactions

You document the exceptions that occur in business transactions for a specific IP address using the recording feature.

Follow these steps:

1. Select Application Insight, Document Transactions from the left navigation menu.

   The Document Transactions window opens.

2. Click Create a Recording to start a new case recording.

   The Record Case dialog opens. By default, the IP address displays for the system that is using the browser to connect to the DevTest Server.

   You can change the IP address when you are running the browser that connects to the agent application from a remote system.

3. (Optional) Select Auto Refresh to display the captured transactions automatically.
4. Enter a name for the test case.

   Including the test identifier from your ALM in the name is recommended. This name appears in the list of the Points of Interest portlet from the Home page.

5. Click Start.

   The Start button changes to an animated Recording... icon to indicate that the recording is in progress.

6. Run the agent application in another browser window, either on the same or on a remote system.
7. Perform the required work on the agent system that is being recorded.

   The frames that are recorded display in the Captured Frames pane.

   A purple pin is added to the frame of the IP address that is being recorded. The captured frames display automatically in the Captured Frames pane when Auto Refresh is selected. Otherwise, click the refresh button.

8. (Optional) To view the recorded transactions in a graphical view, click 🖧.
9. (Optional) To view the recorded transactions in a list view, click ☰.
10. (Optional) Click Show Outline and move the map to view portions of large transactions at a time.
11. Click Stop.
12. To save the recording, click Save.

    The recordings that are saved are listed in the Recording pane of the Document Transactions window.

13. To create documentation for your recorded transactions, click Create Document.

Search for Recorded Test Transactions

You narrow the list of recordings to analyze using the search capability in the Document Transactions window.

Follow these steps:

1. Select Application Insight, Document Transactions from the left navigation menu.
2. Enter the text search criteria in the Enter Search Text field and click Search.

View and Analyze Recorded Test Transactions

The Recording pane of the Document Transactions window displays a list of all the recorded test transactions that were saved. You can view and analyze a recorded test in graphical or list view.

Follow these steps:

1. Select Application Insight, Document Transactions from the left navigation menu.
2. Select a saved test recording from the list in the Recording pane.

   The test transaction opens in the lower pane in graphical view by default.
3. View the tooltip to display the transaction details.
4. To view the transaction details in list view, click ☰.
5. To delete a recording, click 🗑.

Create Documentation from Recorded Transactions

You can create a report of the transactions that were recorded during a test from the Document Transactions window. The report contains information about each frame that was recorded. You can save and print the report.

Follow these steps:

1. From the Document Transaction window, record and run your test.
2. Click Create Document.
3. Enter the title of the report and click OK.

   The report is generated and displays in a browser.

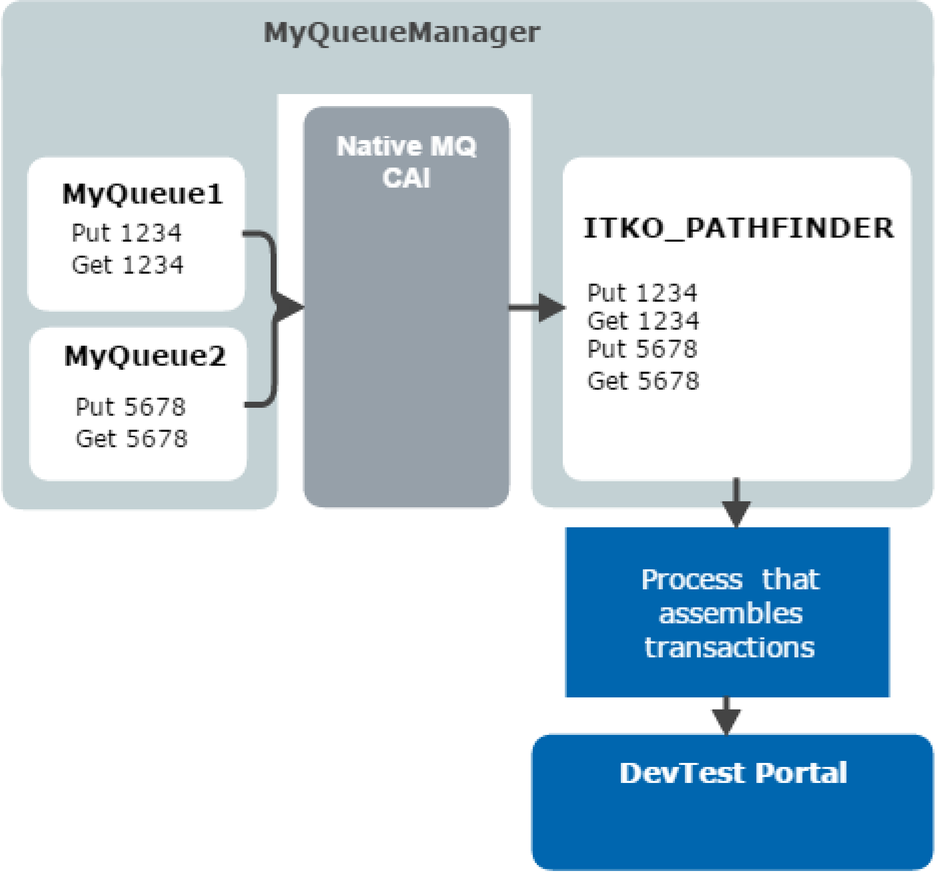
   Note: It may take a few minutes to generate the report depending on the number of transactions.
4. To save the report, move the mouse to the bottom, right of the browser to display the task bar, click the Save button, and enter the fields.
5. To print, click the Print button and enter the fields.

Chapter 10: Native MQ CAI

Native MQ CAI is a WebSphere MQ API exit that monitors the get and put calls for the queues of a queue manager. Native MQ CAI copies the get and put calls to a special queue named ITKO_PATHFINDER. A separate DevTest component uses the data in the ITKO_PATHFINDER queue to assemble transactions that can be viewed in the DevTest Portal. You can then use the transactions to create baselines, virtual services, and so on.

This feature is supported on WebSphere MQ 6.0 and 7.0.

The following graphic shows an example of how Native MQ CAI works. The queue manager has three queues: MyQueue1, MyQueue2, and ITKO_PATHFINDER. The ITKO_PATHFINDER queue contains copies of the get and put calls for MyQueue1 and MyQueue2. The data is converted into transactions and sent to the DevTest Portal.



Native MQ CAI ignores certain types of queues that are not relevant. For example, any queue that has a name beginning with **SYSTEM.** is ignored.

MQPUT and MQPUT1 calls are both supported.

This section contains the following topics:

Create the ITKO_PATHFINDER Queue

Native MQ CAI places copies of get and put calls in a queue that is named ITKO_PATHFINDER.

If the queue fills up, the oldest message is deleted.

Follow these steps:

1. Go to WebSphere MQ Explorer.
2. Create a queue with the name ITKO_PATHFINDER. This queue must be owned by the same queue manager as the queues that you want to monitor. The persistence setting, maximum queue depth, and so on, are up to you.

Install the Native MQ CAI API Exit

The LISA_HOME\agent\native_mq directory contains the API exit files for various operating systems.

Follow these steps:

1. Shut down WebSphere MQ.
2. Navigate to the LISA_HOME\agent\native_mq directory.
3. Copy the exit to the appropriate location, depending on operating system.

   Windows

   Copy the **iTKO_MQ_Pathfinder.dll** file to the MQ_HOME\exits directory.

   Linux

   Copy the **iTKO_MQ_Pathfinder_linux** and **iTKO_MQ_Pathfinder_linux_r** files to the /var/mqm/exits directory. The mqm user must own the files. The files must have the read and execute bits set (chmod a+rx).

   Linux 64 bit

   Copy the **iTKO_MQ_Pathfinder_linux_x64** and **iTKO_MQ_Pathfinder_linux_x64_r** files to the /var/mqm/exits64 directory. The mqm user must own the files. The files must have the read and execute bits set (chmod a+rx).

   SUSE Linux 64 bit

   Copy the **iTKO_MQ_Pathfinder_sles10.3_x64** and **iTKO_MQ_Pathfinder_sles10.3_x64_r** files to the /var/mqm/exits64 directory. The mqm user must own the files. The files must have the read and execute bits set (chmod a+rx).

   Solaris

   Copy the **iTKO_MQ_Pathfinder_sol_sparc** file to the /var/mqm/exits directory. The mqm user must own the file. The file must have the read and execute bits set (chmod a+rx).

   Solaris 64 bit

   Copy the **iTKO_MQ_Pathfinder_sol_sparc.64** file to the /var/mqm/exits64 directory and remove the .64 extension. The mqm user must own the file. The file must have the read and execute bits set (chmod a+rx).

   AIX

   Copy the **iTKO_MQ_Pathfinder_aix** and **iTKO_MQ_Pathfinder_aix_r** files to the /var/mqm/exits directory. The mqm user must own the files. The files must have the read and execute bits set (chmod a+rx).

   AIX 64 bit

Copy the **iTKO_MQ_Pathfinder_aix.64** and **iTKO_MQ_Pathfinder_aix_r.64** files to the /var/mqm/exits64 directory and remove the .64 extension. The mqm user must own the files. The files must have the read and execute bits set (chmod a+rx).

4. Restart WebSphere MQ.

Configure the Native MQ CAI API Exit

After you install the API exit, provide information such as the module and the entry point.

Follow these steps:

1. Go to WebSphere MQ Explorer or open the **qm.ini** configuration file.
2. Configure the queue manager to run an API exit.
   - Set the Name attribute to any descriptive name.
   - Set the Function attribute to **EntryPoint**. This value is case-sensitive.
   - Set the Module attribute to the API exit file that you installed (for example, **/var/mqm/exits/iTKO_MQ_Pathfinder_linux**).
   - If you want Native MQ CAI to generate log files, set the Data attribute to the directory where the log files are written.
3. Restart the queue manager.
4. Send a test message to a queue.
5. Browse the ITKO_PATHFINDER queue and verify that the test message was mirrored.

Example: qm.ini API Exit Configuration on AIX

Add the following lines to **/var/mqm/qmgrs/[QueueManager]/qm.ini**, where **[QueueManager]** is the name of the queue manager in your environment:

```
ApiExitLocal:
   Module=/var/mqm/exits64/iTKO_MQ_Pathfinder_aix
   Name=Pathfinder
   Sequence=100
   Function=EntryPoint
```

This example applies to an AIX system that has 64-bit kernel packages installed. If the system is a pure 32-bit, change the Module attribute to **/var/mqm/exits/iTKO_MQ_Pathfinder_aix**.

You could copy **iTKO_MQ_Pathfinder_aix** and **iTKO_MQ_Pathfinder_aix_r** into both **/var/mqm/exits/** (32-bit libraries) and **/var/mqm/exits64/** (64-bit libraries). Configure the Module attribute with a relative path such as **Module=iTKO_MQ_Pathfinder_aix**. This action enables WebSphere MQ to pick up the correct version of the libraries based on the default ClientExitPath attribute path that is defined in the global WebSphere MQ configuration file (**/var/mqm/mqs.ini**). The **/var/mqm/mqs.ini** file looks like the following example:

```
   DefaultPrefix=/var/mqm
ClientExitPath:
```

```
ExitsDefaultPath=/var/mqm/exits
ExitsDefaultPath64=/var/mqm/exits64
```

Configure the Agent Properties for Native MQ CAI

The DevTest Java Agent includes a set of Native MQ CAI properties that you must configure.

By default, Native MQ CAI monitors all the queues of a queue manager. You can use the Include queues and Exclude queues properties to override this behavior. The property values are regular expressions. The Exclude queues property takes precedence over the Include queues property. If you want to specify a queue name that contains a dot character, place a backslash immediately before the dot character. If you want to use a dot character as a regular expression construct, you do not need to include a backslash.

If the system under test uses an uncommon pattern, Native MQ CAI might need help determining which queue/message is a request and which queue/message is a response. In this scenario, try adding the optional **QUEUE_REQUEST_MATCHES** and **QUEUE_RESPONSE_MATCHES** properties to the **rules.xml** file. Set the values to regular expressions. The regular expressions are evaluated against the message payload. For example:

```
<property key="QUEUE_REQUEST_MATCHES:SOME_REQUEST_QUEUE" value=".*"/>
<property key="QUEUE_RESPONSE_MATCHES:SOME_RESPONSE_QUEUE"
value=".*<response>.*"/>
```

Follow these steps:

1. Open the Agents window.
2. In the left portion, select the agent.
3. Click the Settings tab.
4. Select the MQMirror category.
5. Set the Queue name property to **ITKO_PATHFINDER**. This value is the only valid value.
6. Set the Queue manager name property to the queue manager that owns the queues that you want to monitor.
7. Set the Host property to the IP address or host name of the server where WebSphere MQ is running.
8. Set the Port property to the port number on which WebSphere MQ is listening for connection requests.
9. Set the Channel name property to the WebSphere MQ channel for connecting to the queue manager.
10. (Optional) Set the Connect interval property.
11. If you need a user and password to access WebSphere MQ, set the User and Password properties.
12. If you want to specify which queues to include, set the Include queues property.

13. If you want to specify which queues to exclude, set the Exclude queues property.
14. Click Save.

Generate Transactions from the ITKO_PATHFINDER Queue

The **LisaAgent.jar** file includes an option for creating transactions that are based on the messages that have been mirrored to the ITKO_PATHFINDER queue.

If the broker is running on another computer, you must also specify the **-u** option with the URL to the broker.

Follow these steps:

1. Copy the WebSphere MQ JAR files to the directory where the **LisaAgent.jar** file is located. For the list of JAR files, see the WebSphere MQ section of Third-Party File Requirements in *Administering*.
2. Run the following command:

   ```
   java -jar LisaAgent.jar -m
   ```

   The messages are retrieved from the ITKO_PATHFINDER queue and assembled into transactions. You can view the transactions in the DevTest Portal.

Chapter 11: CAI Command-Line Tool

The **PFCmdLineTool** command lets you perform various CA Continuous Application Insight tasks from the command line.

The main options are **--count, --roots, --paths, --export, --import, --baseline,** and **--virtualize**.

This command has the following format:

```
PFCmdLineTool
[--count|--roots|--paths|--export|--import|--baseline|--virtualize|--help|--versi
on] [task-specific options] [search-criteria]
```

Search Criteria

For all the main options except **--import**, the set of transaction frames that are acted on is defined by those matching the search criteria specified. Use any of the following options to specify the search criteria: **--from, --to, --localIP, --remoteIP, --category, --class, --method, --session, --transaction, –agent, --min-time**, **--tag**, and **--max-frames**. These options are used to narrow the set of root transaction frames that are acted on.

Alternately, you can use the **--frame** option to limit the search results to a single transaction frame.

--from=start-time

Specifies the start time of the window of transaction frames you want. Use either of the following formats: **yyyy-mm-dd** or **yyyy-mm-ddThh:mm:ss**. If this option and the **--frame** option are not specified, the start of the current day is used.

--to=end-time

Specifies the end time of the window of transaction frames you want. Use either of the following formats: **yyyy-mm-dd** or **yyyy-mm-ddThh:mm:ss**. If this option and the **--frame** option are not specified, the end of the current day is used.

--localIP=IP address

Specifies the client-side IP address that CAI records.

--remoteIP=IP address

Specifies the server-side IP address that CAI records.

--category=category

Specifies the type of transaction frame. You can search for more than one category at a time by repeating this option.

Values: amx, client, dn_default, dn_remoting, dn_sql, ejb, gui, jca, jdbc, jms, logging, mq, rest_http, rmi, rmi_http, rmi_ssl, thread, throwable, tibco, web_http, web_https, wm, wps, ws_http, ws_https

--class=class-name

Specifies the class name. The value can be either the full class name or a string ending with '%' to perform a "starts with" type of match.

--method=method-name

Specifies the method name. The value can be either the full method name or a string ending with '%' to perform a "starts with" type of match.

--session=session-id

Specifies the session identifier.

--transaction=transaction-id

Specifies the transaction identifier.

--agent=agent-name

Specifies the name of the agent that is the source for the transaction frames you want.

--min-time=min-time

Specifies the minimum execution time (in milliseconds) of transaction frames you want to see. Frames that are executed faster than this value are filtered out.

--tag=name, --tag=name=value

Specifies a frame tag that is associated with a transaction frame. You can specify the name only, or both the name and value. You can search for more than one tag at a time by repeating this option.

--max-frames=max-frames

Specifies the maximum number of root transaction frames that are retrieved. If this option is not specified, it defaults to 10000. There can be circumstances when more frames are processed than the maximum.

--frame=frame-id

Specifies the identifier of the particular transaction frame desired.

Querying and Display

The following options let you query and display transactions.

-c, --count

Displays the number of root transaction frames that match the specified search criteria.

-r, --roots

Displays the root transaction frames that match the specified search criteria.

-p, --paths

Displays the transaction frame hierarchy for a set of root transaction frames.

Export and Import

The following options let you export and import paths.

-e output-zip-file-name, --export=output-zip-file-name

Exports the selected transaction frames from the database.

-i input-zip-file-name, --import=input-zip-file-name

Imports transactions into the database. Any search criteria that you specify is ignored.

--no-persist

Specifies that the import operation does not persist the data, causing the import file to be validated only.

Note: When DevTest is connected to a database other than the default Derby, you need to specify the database to the broker element of the **rules.xml** file.

```
 <database driver="yourdatabasedriver" url="yourdatabaseurl"
user="yourdatabaseuser" password="yourdatabasepassword"/>
```

This setting is used for the lifetime of the agent.

Baselines

The following options let you create baseline test cases and suites. The **--to-dir** option or the **--to-project** option is required.

-b name, --baseline=name

Generates baseline tests for the selected transaction frames.

--refer=frame-id

Specifies the identifier of the particular transaction frame that is designated as the key transaction frame. If this option is not specified, the first transaction frame in the query result is designated as the key transaction frame.

--consolidated

Specifies that a baseline test case is generated. If this option is not specified, a baseline suite is generated.

--stateful

Specifies that a stateful baseline is generated.

--force-tf-steps

Specifies that baseline tests are generated using the Execute Transaction Frame step only. If this option is not specified, any available CA Application Test test steps are used whenever possible.

Note: For more information about the Execute Transaction Frame step, see *Using CA Application Test*.

--magic-dates

Specifies that baseline tests have magic date processing applied.

--jndi-factory=factory-class-name

Specifies the JNDI factory class to use when generating EJB baseline tests.

--jndi-url=url

Specifies the JNDI URL to use when generating EJB baseline tests.

--jndi-user=user-id

Specifies the JNDI user to use when generating EJB baseline tests.

--jndi-user-cred=user-credentials

Specifies the JNDI user credentials to use when generating EJB baseline tests.

--to-dir=output-pfi-file-name

Specifies the name of the directory where the generated PFI file is written.

--to-project=lisa-project-dir

Specifies the root directory of a project into which the generated artifacts are written, without the requirement of an intermediate import file. You can specify this option instead of, or in addition to, the **--to-dir** option.

Virtualization Artifacts

The following options let you generate virtualization artifacts. The **--to-dir** option or the **--to-project** option is required.

-v name, --virtualize=name

Generates virtualization artifacts for the selected transaction frames.

--refer=frame-id

Specifies the identifier of the particular transaction frame that is designated as the key transaction frame. If this option is not specified, the first transaction frame in the query result is designated as the key transaction frame.

--force-stateless

Specifies that VSE conversational processing is applied when generating a service image.

--unknown-request-action=no_match|no_hijack

Specifies the action that is taken for unknown requests. This option applies to generated virtualization artifacts based around Java VSE. The valid values are **no_match** (return a "no match") and **no_hijack** (bypass the virtual service).

--to-dir=output-pfi-file-name

Specifies the name of the directory where the generated PFI file is written.

--to-project=lisa-project-dir

Specifies the root directory of a project into which the generated artifacts are written, without the requirement of an intermediate import file. You can specify this option instead of, or in addition to, the **--to-dir** option.

Agent Condition

The following options let you check for a specified condition in an agent. The only supported condition is whether the agent is currently dispatching. The term *dispatching* refers to the action of capturing transactions.

--check

Checks for a specified condition in an agent.

--agent=agent-name

Specifies the name of the agent.

--condition=condition

Specifies the condition to check for.

Values: Dispatching

--check-timeout=number-of-seconds

Specifies the number of seconds to wait for the condition before timing out.

Capture Levels

The following options let you modify the capture level for each protocol that the Java agent can capture.

Note: Setting different capture levels is not supported for queue-based client and server communication, for example, WebSphere MQ and JMS.

--set-weights

Modifies the capture level for one or more protocols.

--agent=agent-name

Specifies the name of the agent.

--protocols="protocol[,protocol]"

Specifies the protocol names. If you include more than one value, separate the values by commas.

Values: ALL, HTTP Client, HTTP Server, Logging, Category, Exception, GUI, EJB, JMS, MQ, JCA, RCP, RMI, SAP, Tibco, WPS, WebMethods, JDBC

--weights="weight[,weight]"

Specifies the protocol weights. If you include more than one value, separate the values by commas.

Values: 0, 4, 8. The value 0 corresponds to the Counts level. The value 4 corresponds to the Counts and Paths level. The value 8 corresponds to the Full Data level.

Miscellaneous

The following options are also available.

--broker=broker-url

Specifies the broker connection string. The default value is **tcp://localhost:2009**.

--search-frame

Searches for the transaction frame that matches the specified search criteria and frame name.

--frame-name=frame-name

Specifies the name of a transaction frame to search for.

--help -h

Displays help text.

--version

Prints the VSE version number.

Example: Display Root Transaction Frames

This example shows how to display the root transaction frames for a specified agent and start time.

```
PFCmdLineTool --roots --agent=JBoss_LISABank --from=2014-03-14T12:28:00

Frame ID                          Time                   Category  Local IP      Remote IP     Name
Exec Time
------------------------------------ ---------------------- -------- ------------- -------------
-------  ---------
c3a9c830-abae-11e3-b937-0024d6ab5ce2 2014-03-14 12:28:04 660 web_http 10.132.92.143 10.132.92.143
Unknown  984 ms
```

Example: Display Transaction Frame Hierarchy

This example shows how to display the transaction frame hierarchy for a set of root transaction frames.

```
PFCmdLineTool --paths --agent=JBoss_LISABank --from=2014-03-14T12:28:00

984 ms /lisabank/buttonclick.do (c3a9c830-abae-11e3-b937-0024d6ab5ce2)
   50 ms $Proxy93.getUser (c3c73b40-abae-11e3-b937-0024d6ab5ce2)
      37 ms EJB3UserControlBean.getUser (c3c8c1e0-abae-11e3-b937-0024d6ab5ce2)
         3 ms SQL Activity (1) (c3c8c1e0-abae-11e3-b937-0024d6ab5ce2-SQL)
```

Example: Generate a Baseline Suite

This example shows how to generate a baseline suite.

```
PFCmdLineTool --baseline=BaselineName --refer=c3c8c1e0-abae-11e3-b937-0024d6ab5ce2
--to-dir=C:\DevTest
```

Example: Generate a Stateful Baseline

This example shows how to generate a stateful baseline.

```
PFCmdLineTool --baseline=BaselineName --stateful --from=2014-03-14T12:28:00
--refer=c3c8c1e0-abae-11e3-b937-0024d6ab5ce2 --to-dir=C:\DevTest
```

Example: Generate Virtualization Artifacts

This example shows how to generate virtualization artifacts.

```
PFCmdLineTool --virtualize=VSEServiceName --from=2014-03-14T12:28:00
--refer=c3c8c1e0-abae-11e3-b937-0024d6ab5ce2 --category=ejb --to-dir=C:\DevTest
```

Example: Check the Agent Condition

This example shows how to check whether an agent is capturing transactions. The output indicates that the agent is not capturing transactions.

```
PFCmdLineTool --check --agent=JBoss_LISABank --condition=Dispatching --check-timeout=10
```

```
Agent has not yet started dispatching.
```

Example: Configure the Capture Level

This example shows how to modify the capture level for one protocol.

```
PFCmdLineTool --set-weights --agent=JBoss_LISABank --protocols=EJB --weights=8
```

This example shows how to modify the capture level for multiple protocols. Notice the use of quotation marks.

```
PFCmdLineTool --set-weights --agent=JBoss_LISABank --protocols="EJB,JMS" --weights="8,4"
```

Example: Search Transaction Frame by Name

This example shows how to verify whether CAI captured a particular transaction frame by the frame name. If the search finds more than one matching frame, the newest frame is displayed. The output consists of the name, duration, and unique identifier.

```
PFCmdLineTool --search-frame --frame-name=EJB3AccountControlBean.addAccount
--from=2014-03-20T10:20:00
```

```
EJB3AccountControlBean.addAccount, 141 ms, 27ca1bd0-b03c-11e3-a8f1-005056ba138a
```

Chapter 12: VS Traffic to CA Application Insight Companion

The VS Traffic to CA Application Insight Companion pushes data to the CAI database to generate virtual services and visibility into the application. When you add this companion to a virtual service model, it performs the following actions:

- Captures the requests that the model processes and their corresponding responses.
- Saves the requests in the CAI database as transaction frames.

This companion supports HTTP, JMS, and WebSphere MQ models.

Prerequisites

- Messages from an application are captured by the VSE Recorder.
- A virtual service model file was created. See *Using CA Service Virtualization* for more information about creating a virtual service model.

Follow these steps:

1. Open the virtual service model.
2. In the right panel, select Companions and click + Add.

   The Companions menu opens.
3. Select Virtual Service Environment, VS Traffic to CA Application Insight Companion.
4. Save the virtual service model.
5. Right-click the virtual service and select Deploy/Redeploy to VSE@default. See *Using CA Service Virtualization* for more information about deploying a virtual service.

   A message indicates that the virtual service was deployed to the VSE@default VSE server.
6. Invoke the virtual service.

   The transaction frame is created.

   Note: The transaction frames that are created by this companion display in the DevTest Portal with a virtualized label at the end of the name. An example of a component name for an HTTP transaction is **POST/itkoExamples/EJB3AccountControlBean(virtualized).** See the Path Graph (see page 38) topic for more information about component names.
7. Go to the DevTest Portal and create the baseline test. See the Creating Baselines (see page 81) topic for more information about creating baselines for HTTP, JMS, and WebSphere protocols.
8. Create a DevTest test from the baseline.
9. Verify the test in DevTest Workstation.

Chapter 13: CA Continuous Application Insight Agent Light

The CAI Agent Light is a lightweight agent that sits on any external system and pushes data into CAI. The data is integrated directly into an application and captures transactions using log file format without needing to inject an agent. The light agent is a Java executable JAR file that is named **LisaAgentLight.jar**. The JAR file is executed from the command line. See the Command-Line (see page 199) topic for more information about using the command line with the agent lights.

The data that is captured by the log file is added to the CAI database and displays in the DevTest Console.

CAI has the following agent lights:

- Agent Light for webMethods HTTP (see page 200)
- Agent Light for JMS (see page 203)
- Agent Light for WebSphere MQ (see page 208)

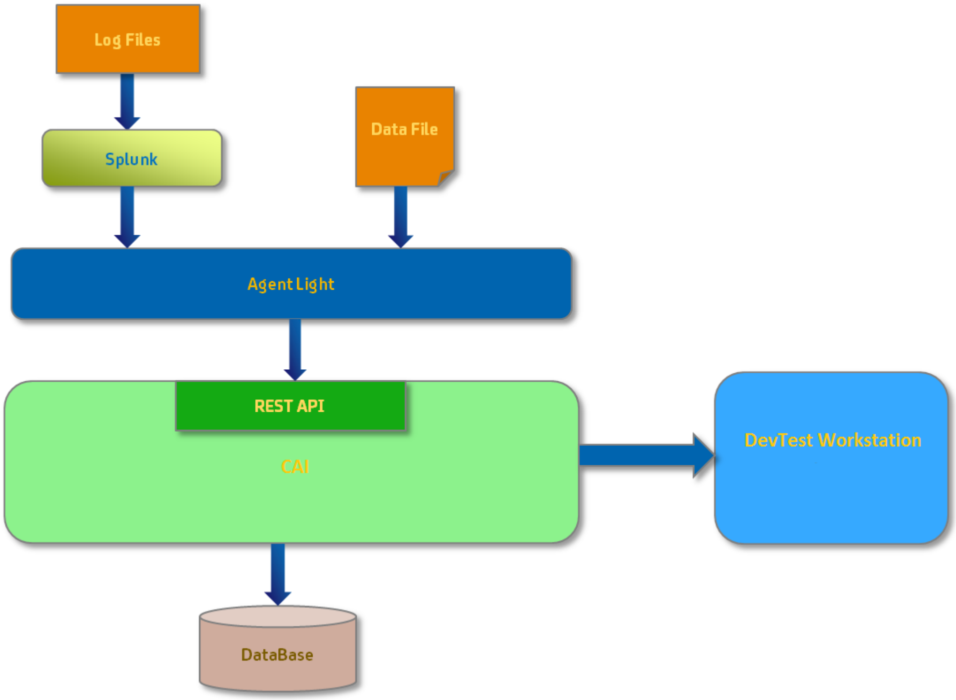
This section contains the following topics:

Agent Light Architecture

The CAI Agent Light architecture has the following components:

- Data from a third-party log file or a data transaction
- Agent Light
- CAI REST API
- DevTest Workstation
- Database

The following graphic shows the components and how they interact:



The agent light receives the data from data transaction files or from third-party log files using a Splunk transaction query.

The agent light sends the data to CAI to create the path through a REST API call.

Agent Light Prerequisites

The following list is requirements to use the agent light:

- Java 1.7 or later
- LisaAgentLight.jar

Install the Agent Light

The **LisaAgentLight.jar** file is installed in the same location as the other CAI agent modules in the **LISA_HOME\agent** directory. The **LisaAgentLight.jar** file can be copied anywhere on any system.

REST API

The CAI REST API takes the transaction data and passes it to the CAI to create the transaction path.

URL

Defines the URL, http://LisaRegistryServer:1505/api/Pathfinder/convertTransaction?Protocol={protocolvalue}.

Method

Defines the POST method.

Parameters

Defines the protocol. HTTP, HTTPS, JMS, and MQ are supported.

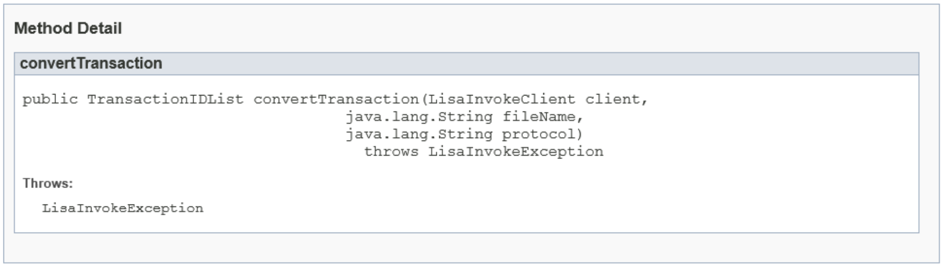
Request Body

(Required) Defines the entire content of the transaction data in Multipart format.

Return

Defines the transaction IDs.

The CAI REST API uses the Java SDK to wrap the REST API, **lisa-invoke2-client.jar**. In the Java SDK, the class **com.itko.lisa.invoke.client.PathfinderServer** wraps the CAI REST APIs. The following graphic displays the method within this class that wraps the REST call:



Method Detail

convertTransaction

```
public TransactionIDList convertTransaction(LisaInvokeClient client,
                                            java.lang.String fileName,
                                            java.lang.String protocol)
                                              throws LisaInvokeException
```

Throws:

```
  LisaInvokeException
```

Command-Line

You use the command line to import the transaction data into CA Continuous Application Insight for the agent lights. Information is parsed and inserted into the CAI database. The command-line provides a method of inserting nodes into CAI without needing to customize or implement a CAI agent. This method includes the ability to specify a source for data (file base and other data) that can read and parse the data. The data is then formatted and inserted into the CAI database.

From the command line, run the **LisaAgentLight.jar** file and use the **-f** and **-t** parameters to specify the location and type of data file. The paths are created and displayed in the CAI Console.

-f, -file

Defines the transaction data file to be imported into CAI.

-t, -type

Defines the data protocol type. HTTP, JMS, and MQ are supported.

Default: HTTP

Agent Light for webMethods HTTP

The Agent Light for webMethods HTTP is an agent that communicates with CA Continuous Application Insight using a REST API. The purpose of the light agent is to take the input from a webMethods HTTP request/response data file and send the data to CAI through the REST API call.

The agent light can receive the webMethods HTTP transactions from the following sources:

- HTTP transaction data file

  The light agent uses the **-f** parameter to take a data file as an input. The data in this file must be in the standard HTTP request/response format. See the Input Transaction Data Format for HTTP (see page 202) topic for more information about the standard data format.

  The transaction parameters from the data file are:

  -f, -file

  Defines the transaction data file to be imported into CAI.

  -t, -type

  Indicates the data protocol type.
  Default: http

  Create a path from data file using the following example code:

  ```
  Java –jar lisaagentlight.jar –url http://<LisaServer>:1505 –f webmethods.log –t
  http
  ```

- webMethods log file using a Splunk query

  When the webMethods integration server is set to **trace**, the web service call details for the HTTP request/response are logged to the log file. Splunk indexes the log file and produces the HTTP transaction data. The agent light queries Splunk to retrieve the webMethods integration server HTTP transactions and creates the CAI paths for them.

  The transaction parameters from Splunk are:

  -splunk

  Indicates transactions acquired from Splunk.

  -h, -hostname

  Defines the Splunk server hostname or IP address.

  -port

  Defines the Splunk port number.

  -u, -username

  Defines the Splunk server user name.

-p, -password

Defines the Splunk server password.

-s, -search

(Optional) Defines the Splunk transaction search statement.

-t, -type

Defines the data protocol type. HTTP, JMS, and MQ are supported.

Default: HTTP

-m, -maxtrans

Defines the maximum transactions to be imported.

Default: 500

Create a path from Splunk using the following example code:

```
Java –jar lisaagentlight.jar –url http://<LisaServer>:1505 –splunk –hostname
10.130.151.105 –port 8089 –username admin –password admin
```

When you query the Splunk server for webMethods HTTP transactions, a Splunk query statement must be provided. The agent has an integrated default query statement. If you want a different query, pass in the new query statement using the **-search** parameter.

The following example displays a default query statement:

```
"search index=main ((CASE(GET) OR CASE(POST) OR CASE(PUT) OR CASE(DELETE)) /*) OR
((Accept OR User-Agent OR Accept-Encoding OR \"-- Host\" OR Authorization OR Cookie
OR Accept-Language OR \"-- Connection\" OR lisaFrameRoot OR lisaFrameRemoteIP OR
lisaFrameID OR Authorization OR Set-Cookie OR SOAPAction OR Content-Type OR
Content-Length): *) OR (\" SOAP Request:\") OR (\" SOAP Response:\") OR (HTTP/1.*)
| transaction startsWith=(CASE(GET) OR CASE(POST) OR CASE(PUT) OR CASE(DELETE))
endsWith=(\"--> Content-Length: \")  | where !searchmatch(\"-- User-Agent:
webMethods\")"
```

Note: When the Agent Light for webMethods HTTP is executed from a remote system, the DevTest Server REST URL must be specified or the agent assumes the DevTest Server is local. The URL format is http://<lisa-server>:1505.

The agent light has the following arguments:

-url

Defines the CAI REST base URL.

Default: http://localhost:1505

-wsuser

(Optional) Defines the CAI REST username.

-wspassword

(Optional) Defines the CAI REST password.

Input Transaction Data Format for webMethods HTTP

The input transaction data for webMethods HTTP must be in the standard HTTP request/response data format. The agent light takes the input data and converts it to the standard HTTP form.

```
HTTP request method URL {version} [line break]
HTTP request header [line break]
….
HTTP request header [line break]
[line break – a blank line]
{HTTP request body – optional}[line break]
HTTP response status line[line break]
HTTP response header[line break]
…
HTTP response header[line break]
[line break – a blank line]
{HTTP response body – optional}[line break]
```

Agent Light for webMethods HTTP Baseline Support

The paths that are created by the agent light support only the DevTest test steps. The use transaction frame step option is disabled in the web interface. If the web service has the authorization mechanism through the HTTP header, the HTTP authorization header must be modified for the generated baseline. See *Generate a Web HTTP Baseline* (see page 100) in for information about generating a Web HTTP baseline.

Agent Light for webMethods HTTP Virtualization Support

The paths that are created by the agent light can be virtualized. The playback goes through the VSE and not the CAI VSE. See *Create Virtual Services from Web HTTP Transactions* (see page 153) for information about creating virtual services from web HTTP transactions.

Agent Light for JMS

The Agent Light for JMS is an agent that communicates with CA Continuous Application Insight using a REST API. The purpose of the light agent is to take the input from a JMS transaction data XML file and send the data to CAI through the REST API call. The agent light can receive the JMS transactions from the JMS transaction data file.

The light agent uses the **-f** parameter to take a data file as an input. The data in this file must be in the XML format that is defined by CAI. See the Input Transaction Data Format for JMS (see page 204) topic for more information about the standard data format.

The transaction parameters from the data file are:

-f, -file

Defines the transaction data file to be imported into CAI.

-t, -type

Indicates the data protocol type.
Default: http

Create a path from a data file using the following example code:

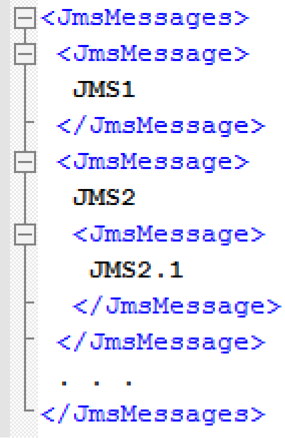
```
Java –jar lisaagentlight.jar –url http://<LisaServer>:1505 –f sampleJmsDataFile.xml
–t jms
```

Input Transaction Data Format for JMS

The input transaction data for JMS must be in the XML format specified by CA Continuous Application Insight.

JmsMessages Tag

JMS message begins with a <JmsMessage> tag and ends with a matching </JmsMessage> tag. JMS messages can contain child JMS messages.

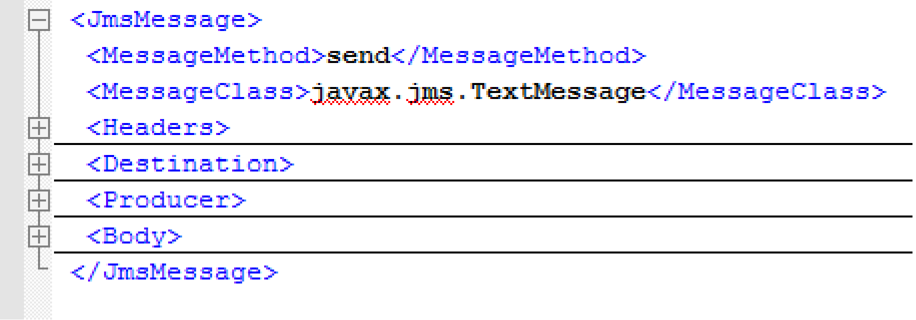


```
<JmsMessages>
 <JmsMessage>
  JMS1
 </JmsMessage>
 <JmsMessage>
  JMS2
  <JmsMessage>
   JMS2.1
  </JmsMessage>
 </JmsMessage>
 . . .
</JmsMessages>
```

MessageMethod and MessageClass Tags

The <MessageMethod> tag is used to specify the message method. The valid message methods are send, receive, and onMessage. When the tag is missing, the default value is send.

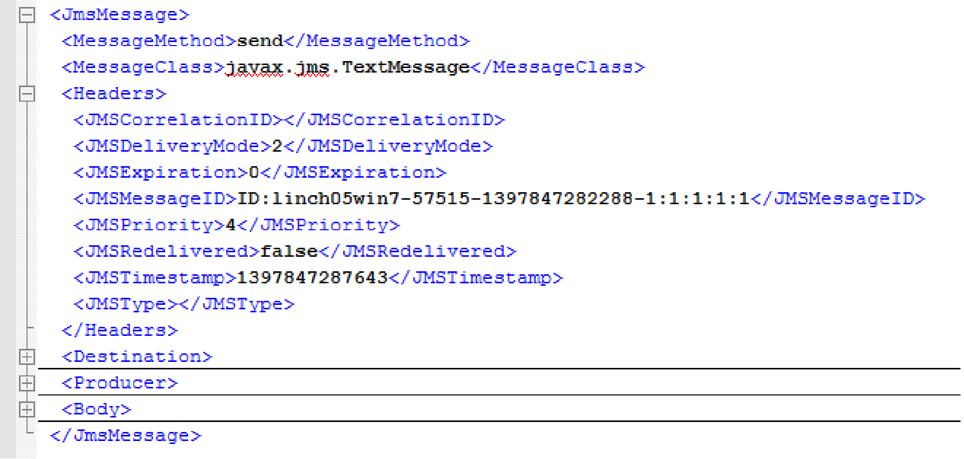
The <MessageClass> tag specifies the message class. The only valid message class is javax.jms.TextMessage. When the tag is missing, the default value is javax.jms.TextMessage.



```
<JmsMessage>
 <MessageMethod>send</MessageMethod>
 <MessageClass>javax.jms.TextMessage</MessageClass>
 <Headers>
 <Destination>
 <Producer>
 <Body>
</JmsMessage>
```

Headers Tags

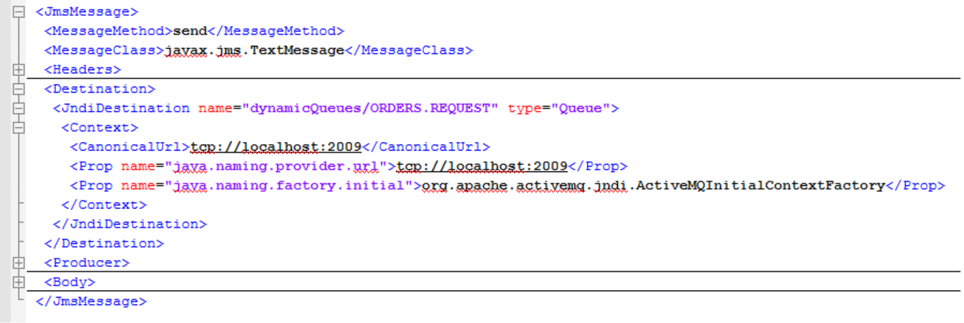
The <Headers> tag specifies the message headers. The valid headers are JMSCorrelationID, JMSDeliveryMode, JMSExpiration, JMSMessageID, JMSPriority, JMSRedelivered, JMSTimestamp, and JMSType.



```
<JmsMessage>
 <MessageMethod>send</MessageMethod>
 <MessageClass>javax.jms.TextMessage</MessageClass>
 <Headers>
  <JMSCorrelationID></JMSCorrelationID>
  <JMSDeliveryMode>2</JMSDeliveryMode>
  <JMSExpiration>0</JMSExpiration>
  <JMSMessageID>ID:linch05win7-57515-1397847282288-1:1:1:1:1</JMSMessageID>
  <JMSPriority>4</JMSPriority>
  <JMSRedelivered>false</JMSRedelivered>
  <JMSTimestamp>1397847287643</JMSTimestamp>
  <JMSType></JMSType>
 </Headers>
 <Destination>
 <Producer>
 <Body>
</JmsMessage>
```

Destination Tags

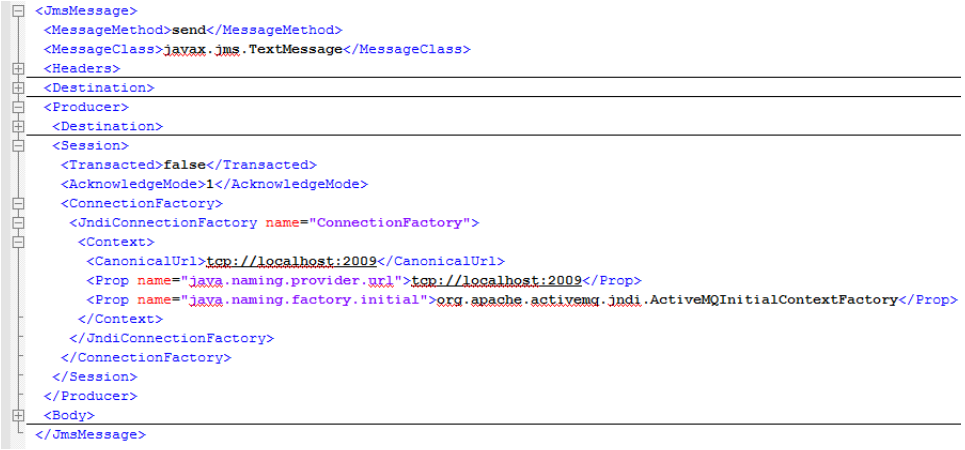
The <Destination> tag specifies the message destination. The valid destination is a JNDI destination (a JNDI-registered Queue or Topic). JMS message with the send method will specify the message destination.



```
<JmsMessage>
 <MessageMethod>send</MessageMethod>
 <MessageClass>javax.jms.TextMessage</MessageClass>
 <Headers>
 <Destination>
  <JndiDestination name="dynamicQueues/ORDERS.REQUEST" type="Queue">
   <Context>
    <CanonicalUrl>tcp://localhost:2009</CanonicalUrl>
    <Prop name="java.naming.provider.url">tcp://localhost:2009</Prop>
    <Prop name="java.naming.factory.initial">org.apache.activemq.jndi.ActiveMQInitialContextFactory</Prop>
   </Context>
  </JndiDestination>
 </Destination>
 <Producer>
 <Body>
</JmsMessage>
```

Producer Tags

The <Producer> tag is used to specify the message producer. The Producer can contain a Destination (JNDI) and a Session. The session contains the session information (Transacted, AcknowledgeMode) and a Connection Factory (JNDI). JMS message with the **send** method will specify the message producer.



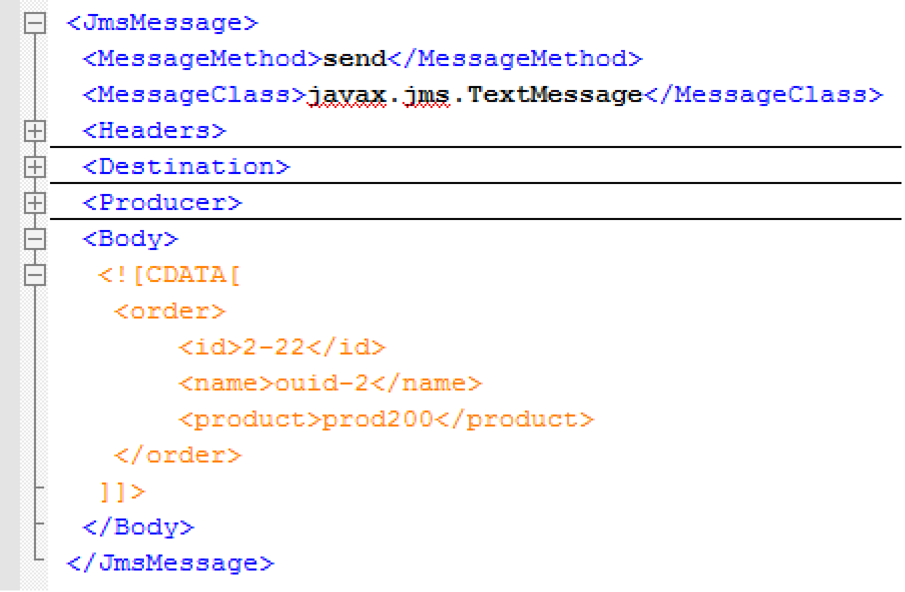
```
<JmsMessage>
 <MessageMethod>send</MessageMethod>
 <MessageClass>javax.jms.TextMessage</MessageClass>
 <Headers>
 <Destination>
 <Producer>
  <Destination>
  <Session>
   <Transacted>false</Transacted>
   <AcknowledgeMode>1</AcknowledgeMode>
   <ConnectionFactory>
    <JndiConnectionFactory name="ConnectionFactory">
     <Context>
      <CanonicalUrl>tcp://localhost:2009</CanonicalUrl>
      <Prop name="java.naming.provider.url">tcp://localhost:2009</Prop>
      <Prop name="java.naming.factory.initial">org.apache.activemq.jndi.ActiveMQInitialContextFactory</Prop>
     </Context>
    </JndiConnectionFactory>
   </ConnectionFactory>
  </Session>
 </Producer>
 <Body>
</JmsMessage>
```

Consumer Tags

The <Consumer> tag is used to specify the message consumer. The Consumer has the same structures as the Producer. JMS message with the receive/onMessage method will specify the message consumer.

Body Tags

The <Body> tag is used to specify the message payload (text). If the text payload itself contains the XML tags, the text payload must be inside a CDATA section. The payload is ignored by the parser.



```
<JmsMessage>
 <MessageMethod>send</MessageMethod>
 <MessageClass>javax.jms.TextMessage</MessageClass>
 <Headers>
 <Destination>
 <Producer>
 <Body>
  <![CDATA[
   <order>
       <id>2-22</id>
       <name>ouid-2</name>
       <product>prod200</product>
   </order>
  ]]>
 </Body>
</JmsMessage>
```

Agent Light for JMS Baseline Support

The paths that are created by the agent light support only the DevTest test steps. Consolidated and expanded baselines are only supported. See *Generate a JMS Baseline* (see page 89) for information about generating a JMS baseline.

Agent Light for JMS Virtualization Support

The paths that are created by the agent light can be virtualized. See the *Create Virtual Services from JMS Transactions* (see page 136) for more information about creating a virtual service for MQ transactions.

Agent Light for WebSphere MQ

The Agent Light for WebSphere MQ is an agent that communicates with CA Continuous Application Insight using a REST API. The purpose of the light agent is to take the input from an MQ request/response data file and send the data to CAI through the REST API call.

The agent light can receive the MQ transactions from the following sources:

- MQ transaction data file

  The light agent uses the **-f** parameter to take a data file as an input. The transaction parameters from the data file are:

  -f, -file

  Defines the transaction data file to be imported into CAI.

  -t, -type

  Defines the data protocol type. HTTP, JMS, and MQ are supported.

  Default: HTTP

  Create a path from data file using the following example code:

  ```
  Java –jar lisaagentlight.jar –url http://<LisaServer>:1505 –f mqSample.xml –t mq
  ```

- Third-party log file from a Splunk query

  Third-party log files are supported using a Splunk query. Splunk indexes the log file and produces the MQ transaction data. The agent light queries Splunk to get the MQ transactions and creates a CAI path for them. The third-party log file may not contain all the information required by the MQ transaction. An extended XML data file is defined to allow users to input the missing data. The data in the extended file must be in the XML format defined by DevTest Solutions.

  The transaction parameters from Splunk are:

  -splunk

  Indicates transactions acquired from Splunk.

  -h, -hostname

  Defines the Splunk server hostname or IP address.

  -port

  Defines the Splunk port number.

  -u, -username

  Defines the Splunk server user name.

  -p, -password

  Defines the Splunk server password.

  -s, -search

(Optional) Defines the Splunk transaction search statement.

-t, -type

Defines the data protocol type. HTTP, JMS, and MQ are supported.

Default: HTTP

-m, -maxtrans

Defines the maximum transactions to be imported.

Default: 500

-source

Defines the Splunk data source, either webMethods, swamq or JMS.

-extf, -extfile

Defines the extended transaction data file.

Create a path from Splunk using the following example code:

```
Java –jar lisaagentlight.jar –url http://<LisaServer>:1505 –splunk –hostname
10.130.151.105 –port 8089 –username admin –password admin -source swamq -type mq
-extfile extendedDataFile.xml
```

When you query Splunk for third-party transactions, a Splunk query statement must be provided to query for the transactions. The agent has a default query statement built-in. If a different query is needed, you can pass a new query statement using the **-search** parameter.

The following statement is the built-in default query statement:

```
"search index=main CASE(\"- REQUEST: <?xml\") OR CASE(\"- RESPONSE:
<?xml\") | transaction startsWith=CASE(REQUEST:)
endsWith=CASE(RESPONSE:)";
```

If the Agent Light for WebSphere MQ is executed from a remote system, the DevTest Server REST URL must be specified. Otherwise the agent assumes the DevTest Server is local. The URL format is **http://<lisa-server>:1505**.

The agent light has the following optional parameters:

-url

Defines the CAI REST base URL.

Default: **http://localhost:1505**

-wsuser

(Optional) Defines the CAI REST username.

-wspassword

(Optional) Defines the CAI REST password.

Note: The data in the transaction data file and in the extended XML data file must be in the XML format that is defined by DevTest Solutions. See the Input Transaction Data Format for MQ (see page 211) topic for more information about the standard data format.

Input Transaction Data Format for MQ

The input transaction data for MQ must be in the format specified by DevTest Solutions. Every MQ transaction must look like the following example:

```
<MQTransactions>
         <MQTransaction>
                  <MQMessage>
                           ....
                           <QueueConnection>
                           ...
                           </QueueConnection>
                           <request>request content</request>
                  </MQMessage>
                  <MQMessage>
                  ...
                  </MQMessage>
                  <MQMessage>
                  ...
                  </MQMessage>
                  ....
     </MQTransaction>
     ...
     <MQTransaction/>
</MQTransactions>
```

The data in the extended data file needs to be in the XML format defined by DevTest Solutions.

```
<Platforms>
         <SWA>
                  <!-- The log is from mq client or not, default value is true -->
                  <IsClient>true</IsClient>
                  <QueueConnection>
                           <!-- required -->
                           <RequestQueue>ORDERS.REQUEST</RequestQueue>
                           <ResponseQueue>ORDERS.RESPONSE</ResponseQueue>
                           <QueueManager>QueueManager</QueueManager>
                           <Host>HostName</Host>
                           CA Portal1414</Port>
                           <Channel>SERVERCON</Channel>
                           <!-- often appear -->
                           <Username>administrator</Username>
                           <Password>dcam09@CTC</Password>
                  </QueueConnection>
         </SWA>
</Platforms>
```

Agent Light for WebSphere MQ Baseline Support

The path created by the agent light supports only the DevTest test steps. Consolidated and expanded baselines are only supported. See *Generate a WebSphere MQ Baseline* (see page 114) for information about generating a WebSphere MQ baseline.

Agent Light for WebSphere MQ Virtualization Support

The paths created by the agent light can be virtualized. See *Create Virtual Service from WebSphere MQ Transactions* (see page 157) for more information about creating a virtual service for MQ transactions.

Glossary

assertion

An *assertion* is an element that runs after a step and all its filters have run. An assertion verifies that the results from running the step match the expectations. An assertion is typically used to change the flow of a test case or virtual service model. Global assertions apply to each step in a test case or virtual service model. For more information, see Assertions in *Using CA Application Test.*

asset

An *asset* is a set of configuration properties that are grouped into a logical unit. For more information, see Assets in *Using CA Application Test*.

audit document

An *audit document* lets you set success criteria for a test, or for a set of tests in a suite. For more information, see Building Audit Documents in *Using CA Application Test.*

companion

A *companion* is an element that runs before and after every test case execution. Companions can be understood as filters that apply to the entire test case instead of to single test steps. Companions are used to configure global (to the test case) behavior in the test case. For more information, see Companions in *Using CA Application Test.*

configuration

A *configuration* is a named collection of properties that usually specify environment-specific values for the system under test. Removing hard-coded environment data enables you to run a test case or virtual service model in different environments simply by changing configurations. The default configuration in a project is named project.config. A project can have many configurations, but only one configuration is active at a time. For more information, see Configurations in *Using CA Application Test.*

Continuous Service Validation (CVS) Dashboard

The *Continuous Validation Service (CVS) Dashboard* lets you schedule test cases and test suites to run regularly, over an extended time period. For more information, see Continuous Validation Service (CVS) in *Using CA Application Test.*

conversation tree

A *conversation tree* is a set of linked nodes that represent conversation paths for the stateful transactions in a virtual service image. Each node is labeled with an operation name, such as withdrawMoney. An example of a conversation path for a banking system is getNewToken, getAccount, withdrawMoney, deleteToken. For more information, see *Using CA Service Virtualization.*

coordinator

A *coordinator* receives the test run information as documents, and coordinates the tests that are run on one or more simulator servers. For more information, see Coordinator Server in *Using CA Application Test.*

data protocol

A *data protocol* is also known as a data handler. In CA Service Virtualization, it is responsible for handling the parsing of requests. Some transport protocols allow (or require) a data protocol to which the job of creating requests is delegated. As a result, the protocol has to know the request payload. For more information, see Using Data Protocols in *Using CA Service Virtualization.*

data set

A *data set* is a collection of values that can be used to set properties in a test case or virtual service model at run time. Data sets provide a mechanism to introduce external test data into a test case or virtual service model. Data sets can be created internal to DevTest, or externally (for example, in a file or a database table). For more information, see Data Sets in *Using CA Application Test.*

desensitize

Desensitizing is used to convert sensitive data to user-defined substitutes. Credit card numbers and Social Security numbers are examples of sensitive data. For more information, see Desensitizing Data in *Using CA Service Virtualization.*

event

An *event* is a message about an action that has occurred. You can configure events at the test case or virtual service model level. For more information, see Understanding Events in *Using CA Application Test.*

filter

A *filter* is an element that runs before and after a step. A filter gives you the opportunity to process the data in the result, or store values in properties. Global filters apply to each step in a test case or virtual service model. For more information, see Filters in *Using CA Application Test.*

group

A *group*, or a *virtual service group,* is a collection of virtual services that have been tagged with the same group tag so they can be monitored together in the VSE Console.

Interactive Test Run (ITR)

The *Interactive Test Run (ITR)* utility lets you run a test case or virtual service model step by step. You can change the test case or virtual service model at run time and rerun to verify the results. For more information, see Using the Interactive Test Run (ITR) Utility in *Using CA Application Test.*

lab

A *lab* is a logical container for one or more lab members. For more information, see Labs and Lab Members in *Using CA Application Test.*

magic date

During a recording, a date parser scans requests and responses. A value matching a wide definition of date formats is translated to a *magic date*. Magic dates are used to verify that the virtual service model provides meaningful date values in responses. An example of a magic date is {{=doDateDeltaFromCurrent("yyyy-MM-dd","10");/*2012-08-14*/}. For more information, see Magic Strings and Dates in *Using CA Service Virtualization.*

magic string

A *magic string* is a string that is generated during the creation of a service image. A magic string is used to verify that the virtual service model provides meaningful string values in the responses. An example of a magic string is {{=request_fname;/chris/}}. For more information, see Magic Strings and Dates in *Using CA Service Virtualization.*

match tolerance

Match tolerance is a setting that controls how CA Service Virtualization compares an incoming request with the requests in a service image. The options are EXACT, SIGNATURE, and OPERATION. For more information, see Match Tolerance in *Using CA Service Virtualization.*

metrics

Metrics let you apply quantitative methods and measurements to the performance and functional aspects of your tests, and the system under test. For more information, see Generating Metrics in *Using CA Application Test.*

Model Archive (MAR)

A *Model Archive (MAR)* is the main deployment artifact in DevTest Solutions. MAR files contain a primary asset, all secondary files that are required to run the primary asset, an info file, and an audit file. For more information, see Working with Model Archives (MARs) in *Using CA Application Test.*

Model Archive (MAR) Info

A *Model Archive (MAR) Info* file is a file that contains information that is required to create a MAR. For more information, see Working with Model Archives (MARs) in *Using CA Application Test.*

navigation tolerance

Navigation tolerance is a setting that controls how CA Service Virtualization searches a conversation tree for the next transaction. The options are CLOSE, WIDE, and LOOSE. For more information, see Navigation Tolerance in *Using CA Service Virtualization.*

network graph

The network graph is an area of the Server Console that displays a graphical representation of the DevTest Cloud Manager and the associated labs. For more information, see Start a Lab in *Using CA Application Test*.

node

Internal to DevTest, a test step can also be referred to as a *node*, explaining why some events have node in the EventID.

path

A *path* contains information about a transaction that the Java Agent captured. For more information, see *Using CA Continuous Application Insight.*

path graph

A *path graph* contains a graphical representation of a path and its frames. For more information, see Path Graph (see page 38) in *Using CA Continuous Application Insight*.

project

A *project* is a collection of related DevTest files. The files can include test cases, suites, virtual service models, service images, configurations, audit documents, staging documents, data sets, monitors, and MAR info files. For more information, see Project Panel in *Using CA Application Test.*

property

A *property* is a key/value pair that can be used as a run-time variable. Properties can store many different types of data. Some common properties include LISA_HOME, LISA_PROJ_ROOT, and LISA_PROJ_NAME. A configuration is a named collection of properties. For more information, see Properties in *Using CA Application Test.*

quick test

The *quick test* feature lets you run a test case with minimal setup. For more information, see Stage a Quick Test in *Using CA Application Test.*

registry

The *registry* provides a central location for the registration of all DevTest Server and DevTest Workstation components. For more information, see Registry in *Using CA Application Test.*

service image (SI)

A *service image* is a normalized version of transactions that have been recorded in CA Service Virtualization. Each transaction can be stateful (conversational) or stateless. One way to create a service image is by using the Virtual Service Image Recorder. Service images are stored in a project. A service image is also referred to as a *virtual service image* (VSI). For more information, see Service Images in *Using CA Service Virtualization.*

simulator

A *simulator* runs the tests under the supervision of the coordinator server. For more information, see Simulator Server in *Using CA Application Test.*

staging document

A *staging document* contains information about how to run a test case. For more information, see Building Staging Documents in *Using CA Application Test.*

subprocess

A *subprocess* is a test case that another test case calls. For more information, see Building Subprocesses in *Using CA Application Test.*

test case

A *test case* is a specification of how to test a business component in the system under test. Each test case contains one or more test steps. For more information, see Building Test Cases in *Using CA Application Test.*

test step

A *test step* is an element in the test case workflow that represents a single test action to be performed. Examples of test steps include Web Services, JavaBeans, JDBC, and JMS Messaging. A test step can have DevTest elements, such as filters, assertions, and data sets, attached to it. For more information, see Building Test Steps in *Using CA Application Test.*

test suite

A *test suite* is a group of test cases, other test suites, or both that are scheduled to execute one after other. A suite document specifies the contents of the suite, the reports to generate, and the metrics to collect. For more information, see Building Test Suites in *Using CA Application Test.*

think time

Think time is how long a test case waits before executing a test step. For more information, see Add a Test Step (example) and Staging Document Editor - Base Tab in *Using CA Application Test.*

transaction frame

A *transaction frame* encapsulates data about a method call that the DevTest Java Agent or a CAI Agent Light intercepted. For more information, see Business Transactions and Transaction Frames (see page 15) in *Using CA Continuous Application Insight.*

Virtual Service Environment (VSE)

The *Virtual Service Environment (VSE)* is a DevTest Server application that you use to deploy and run virtual service models. VSE is also known as CA Service Virtualization. For more information, see *Using CA Service Virtualization.*

virtual service model (VSM)

A *virtual service model* receives service requests and responds to them in the absence of the actual service provider. For more information, see Virtual Service Model (VSM) in *Using CA Service Virtualization.*