DevTest Solutions

Agents

Version 8.0

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: DevTest Java Agent

The DevTest Java Agent is a piece of server-side technology that can be installed inside any Java process, including Java EE containers. The agent enables DevTest to control and monitor server-side activities.

The agent can do what most profilers do: monitor loaded classes/objects, CPU usage, memory usage, threads, track method calls, and so on. However, the agent works across multiple JVMs and is used with DevTest to bring unique features to testing.

In particular, the agent provides visibility into the actions that a test or test step causes the servers to do behind the scenes. This capability can help identify bugs and bottlenecks. This capability is similar to what CA Continuous Application Insight does. However, it works across all protocols that the Java applications use without the need to instrument any code or even any configuration files.

The agent also supports CA Service Virtualization. The agent enables record and replay of traffic and method calls across protocols. The agent gives CA Service Virtualization complete control of the areas of the target application that you want to virtualize. The agent provides a unified framework to accomplish this functionality regardless of the protocol.

This section contains the following topics:

Java Agent Architecture

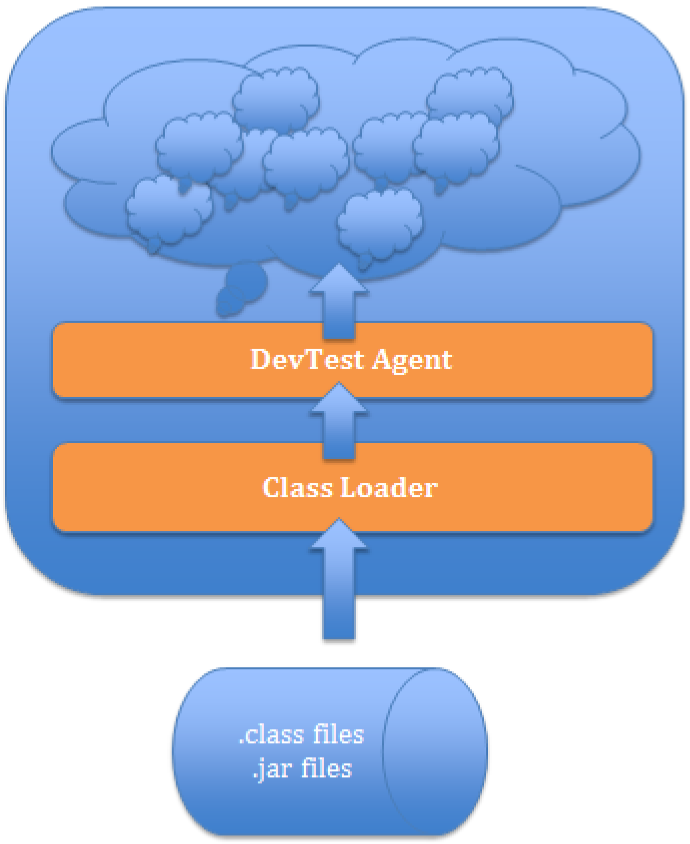
This section contains the following topics:

Java Agent Technology

The DevTest Java Agent is a type of Java agent.

Java agents are programs that are embedded in a Java virtual machine (JVM). These programs can be designed to support many functions, such as collecting information about a running application or virtualizing parts of an application.

In the following graphic, the large container represents the JVM. The JVM includes an agent and the class loader, which loads Java class files at run time. The clouds represent resources that the Java application loads.



An agent is packaged in a JAR file. The JVM must be configured with the path to the JAR file. As of Java 1.5, you use a string that starts with **-javaagent** to specify the path and any options. For example:

```
-javaagent:C:\myagent.jar=option1=true,option2=false
```

A single JVM can include multiple agents.

Java Agent Components

The DevTest Java Agent has the following components:

- The agent itself, which runs embedded in a Java process
- The broker
- The consoles (or clients)
- The database

The **broker** is a hub that dispatches Java Message Service (JMS) messages between agents, consoles, and an embedded client.

The embedded client tracks network wide/agent-spanning properties, such as the set of agents, their open queues, or current network connections. As such, the embedded client can assemble partial transaction data into full transactions for the benefit of consoles.

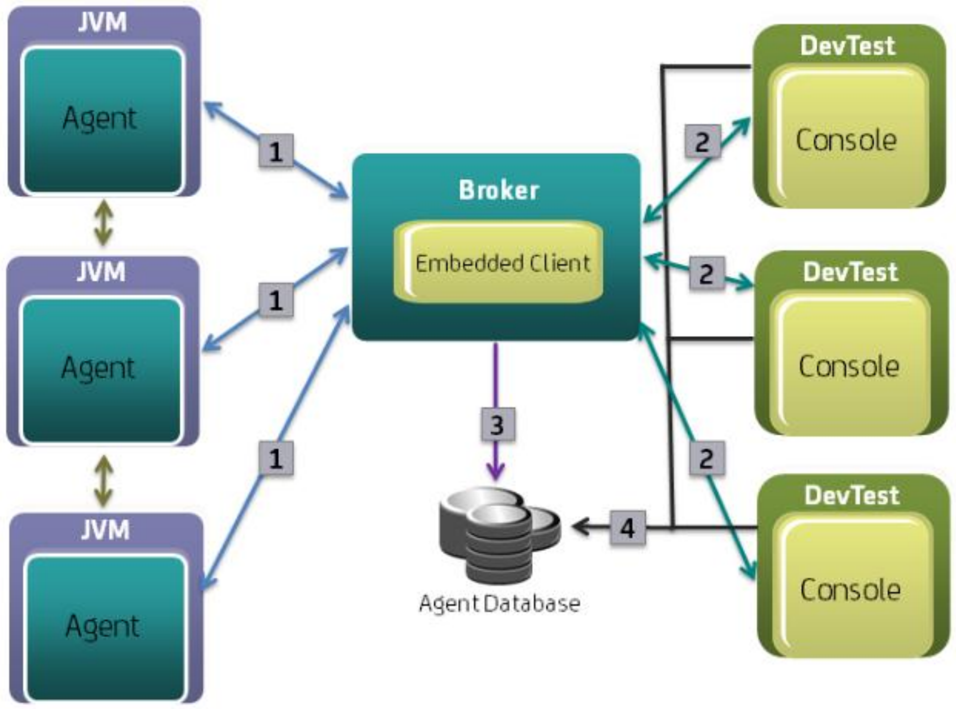
After the data is assembled, the data is sent to the consoles (short term) and persisted to the database for long-term storage.

This documentation ignores the distinction and refers to the embedded client as the broker.

The **consoles** get their finalized data from the broker (if recent) and the database (if older) for display and user interaction. A console does not need to be a GUI component. Consoles include DevTest Workstation, VSE, and simulators.

Java Agent Data Flow

The following graphic shows the Java agent components and how they interact.



The following JMS destinations specify the flow of data:

- The **lisa.agent.info** topic is carried over connections 1 and 2, produced by the agents and consumed by the broker and the consoles. This topic lets the broker and the consoles see which agents are currently online and what their basic properties are.
- The **lisa.agent.port** topic is carried over connection 1, produced by the agents and consumed by the broker. This topic lets the broker see the connections that are currently active between multiple agents.
- The **lisa.agent.api** topic is carried over connections 1 and 2, produced by the consoles and consumed (and replied to) by the agents. This topic allows the consoles to invoke agent APIs over JMS.
- The **lisa.broker.api** topic is carried over connection 2, produced by the consoles and consumed (and replied to) by the broker. This topic allows the consoles to invoke broker APIs over JMS.
- The **lisa.stats** topic is carried over connections 1 and 2, produced by the agents and consumed by the broker and the consoles. This topic gives the consoles an idea of what type of load the agents are currently under. This topic also lets the broker persist those to the database.

- The **lisa.vse** topic is carried over connections 1 and 2, produced by the agents and consumed by the consoles. When VSE is enabled, the consoles receive VSE frames (and reply to them in playback mode).
- The **lisa.tx.partial** queue is carried over connection 1, produced by the agents and consumed by the broker. When an agent captures a partial transaction (all the frames that happen in its JVM), the agent sends it to the broker for assembly.
- The **lisa.tx.full** topic is carried over connection 2, produced by the broker and consumed by the consoles. When the broker finishes assembling the partial transactions that are received over **lisa.tx.partial**, the broker sends the full transactions to the consoles.
- The **lisa.tx.incomplete** topic is carried over connection 2, produced by the broker and consumed by the consoles. This topic is similar to **lisa.tx.full**, but is used for transactions that are not fully completed within the allowed timeout.
- **JDBC** connection 3 is used when the broker saves **StatsFrame** objects or fully assembled **TransactionFrame** objects.
- The consoles use **JDBC** connection 4 to perform their queries for transactions or statistics that are no longer held in memory.

The startup order of agents, broker, and consoles does not matter because all communications are asynchronous and can reestablish themselves automatically. This concept is especially important for agents to avoid performance issues because the broker goes down, for example.

When an agent comes online, the agent starts pulsing its information over the **lisa.agent.info** topic at regular, short intervals.

If a broker is not available, the agent does not try to notify any listeners of anything else until a connection is established or reestablished.

If the broker is available, all interested parties are quickly notified of the online agents. The broker and consoles keep a running list of those agents. When they stop pulsing their information, they are expired and removed from the list after a while.

Java Agent Database Schema

The broker automatically creates the database schema for the DevTest Java Agent.

To create the schema manually, obtain the DDL statements by running the following command:

```
java -jar LisaAgent.jar -ddl <oracle|sqlserver|mysql|derby|db2>
```

You can also obtain the DDL statements from the following files in the **LISA_HOME\database** directory:

- **db2_pathfinder.ddl**
- **derby_pathfinder.ddl**
- **mysql_pathfinder.ddl**
- **oracle_pathfinder.ddl**
- **sqlserver_pathfinder.ddl**

Typically, you create the schema manually for security or process reasons, or possibly migration or documentation concerns.

Java Agent Data Categories

The DevTest Java Agent can capture the following categories of data:

- Client
- EJB
- GUI (AWT, Swing, SWT)
- JCA
- JDBC
- JMS
- Logging
- REST
- RMI
- SAP
- Thread: a synthetic frame that shows stitching across threads
- Throwable
- TIBCO ActiveMatrix BusinessWorks
- webMethods Integration Server
- WebSphere MQ
- Web (HTTP/S)
- Web service

Java Agent Installation

This section describes how to install the DevTest Java Agent.

The DevTest Java Agent has two versions:

- Pure Java agent
- Native agent

The pure Java agent has no platform-dependent code.

The native agent requires a platform-dependent library module.

Important! Install the pure Java agent unless you need to create TIBCO BusinessWorks baselines. This feature requires the native agent.

This section contains the following topics:

Install the Pure Java Agent

This topic describes how to install the pure Java version of the DevTest Java Agent.

Note: Investigate the target environment ahead of time and ensure that it has been tested, or test it yourself. Most Java agent issues are OS or JVM-dependent, instead of application-dependent. Be sure to follow the instructions in this documentation. If that does not help, review Java Agent Troubleshooting (see page 85) before contacting Support.

Follow these steps:

1. Obtain the following files from the **LISA_HOME\agent** directory:
   - **LisaAgent.jar**
   - **InsightAgent.jar** (for all platforms other than Oracle WebLogic Server)
   - **LisaAgent2.jar** (for Oracle WebLogic Server)
2. Place the files anywhere on your disk that has read permissions from the Java application and is not automatically loaded in the application classpath (such as **\WEB-INF\lib** directories).
3. Go to the platform-specific agent installation (see page 22) section and follow the instructions for the platform that you are using.

   For all platforms other than Oracle WebLogic Server, you will specify an agent parameters string in the following format:

   ```
   -javaagent:<path_to_InsightAgent.jar>[=url=<broker_url>][,name=<agent_name>]
   ```

   For Oracle WebLogic Server, you will specify an agent parameters string in the following format:

   ```
   -javaagent:<path_to_LisaAgent2.jar>[=url=<broker_url>][,name=<agent_name>]
   ```
4. Start the Java application.

Install the Native Agent

This topic describes how to install the native version of the DevTest Java Agent.

Note: Investigate the target environment ahead of time and ensure that it has been tested, or test it yourself. Most Java agent issues are OS or JVM-dependent, instead of application-dependent. Be sure to follow the instructions in this documentation. If that does not help, review Java Agent Troubleshooting (see page 85) before contacting Support.

If you want to create TIBCO BusinessWorks baselines, install the native agent. In addition, add the **heap** option to the agent parameters string and set the value to true.

The following table contains the file names of the platform-dependent library modules for the native agent:

Module	**File Name**
Windows (32-bit JVM)	**JavaBinder.dll**
Windows (64-bit JVM)	**JavaBinder.amd64.dll**
Mac OS X (x86/x64) 32/64-bit	**libJavaBinder.jnilib**
Linux (x86) 32-bit	**libJavaBinder.x86.so**
Linux (x64) 64-bit	**libJavaBinder.x64.so**
Solaris (x86) 32-bit	**libJavaBinder.solaris.x86.so**
Solaris (x64) 64-bit	**libJavaBinder.solaris.x64.so**
Solaris (Sparc) 32-bit	**libJavaBinder.solaris.sparc32.so**
Solaris (Sparc) 64-bit	**libJavaBinder.solaris.sparc64.so**
HP-UX (RISC) 32-bit	**libJavaBinder.hp-ux.sl**

Follow these steps:

1. Obtain the following files from the **LISA_HOME\agent** directory:
   - **LisaAgent.jar**
   - The platform-dependent library module for the target environment.
2. Place the files anywhere on your disk that has read permissions from the Java application and is not automatically loaded in the application classpath (such as \WEB-INF\lib directories).
3. Go to the platform-specific agent installation (see page 22) section and follow the instructions for the platform that you are using. You will specify an agent parameters string in the following format:

   ```
   -agentpath:<path_to_JavaBinder_file>=jar=file:<path_to_LisaAgent.jar>[,url=<b
   roker_url>][,name=<agent_name>]
   ```
4. Start the Java application.

Options for Agent Parameters String

When you install the DevTest Java Agent, you specify an agent parameters string. This topic describes the options that can be included in the string.

Important! The **name** option is required.

url

Defines the broker to which the agent connects. Use the following format:

```
url=tcp://broker-host:broker-port
```

Set the **broker-host** portion to the name or IP address of the computer where the broker is deployed.

Set the **broker-port** portion to the TCP port on which the broker is listening. The default value is 2009.

name

Defines the name of the agent. Use the following format:

```
name=agent-name
```

Be sure to provide a descriptive name.

Avoid using the same name for multiple agents.

token

Makes access to the agent secure. Use the following format:

```
token=admin-token:user-token
```

For more information, see Java Agent Security (see page 80).

heap

The **heap** option is available only with the native agent.

Specifies whether to enable the heap-walking APIs, which are required for TIBCO BusinessWorks baselines. The valid values are true and false. The default value is false.

jar

The **jar** option is available only with the native agent.

Defines the path to the **LisaAgent.jar** file.

Agent Install Assistant

The agent install assistant is a command-line utility that helps you determine the value for the agent parameters string.

The agent install assistant also verifies that the JVM runs properly with the agent installed.

Note: If you intend to create TIBCO BusinessWorks baselines, add the **heap** option to the agent parameters string and set the value to true.

Automatic Configuration

The agent install assistant provides an option to automatically configure the Java agent on platforms that are based on the user values in the **JavaAgentInstaller.xml** file. This XML file is a template that contains the platform and the version information that is supported by DevTest. The **JavaAgentInstaller.xml** file is located in the same directory as the **LisaAgent.jar** file.

The following prerequisite steps must be performed to use the option to configure the Java agent automatically:

Follow these steps:

1. Search for the platform that is based on the platform name and operating system type that is listed in the **JavaAgentInstaller.xml** file.
2. Define the environment variable, if applicable.

   The environment variable is embedded in ${}, for example ${JBOSS_HOME}.
3. Replace any customized value, if applicable.

   The customized value is embedded in {}, for example:

   ```
   <ConfigFileName>${ORACLE_HOME}\user_projects\domains\{custom domain
   name}\bin\startWebLogic.cmd<ConfigFileName>
   ```

   Replace it with your project name:

   ```
   <ConfigFileName>${ORACLE_HOME}\user_projects\domains\lisa_domain\bin\startWeb
   Logic.cmd</ConfigFileName>
   ```
4. Review all the predefined values and edit values that are based on your system.

Run the Agent Install Assistant

The following procedure has separate steps for entering the path of the **java** executable, the broker URL, and the agent name. However, you can specify any of these values in the second step instead.

Follow these steps:

1. Go to the computer where you are installing the agent.
2. Run the **LisaAgent.jar** file with the **-ia** option.
   - If you are installing the pure Java agent, add the **-java** option.
   - If you are installing the native agent, add the **-native** option.
3. When prompted, enter the path of the **java** executable. The default value is the path of the **java** executable that is executing.
4. When prompted, enter the broker URL. The default value is **tcp://localhost:2009**.
5. When prompted, enter the agent name. The default value is **{{x}}**, which means that a unique name is generated at runtime.
6. Review the output.
7. (Optional) To configure the agent automatically, enter **Yes** and follow the prompts.

For more information, see the following examples:

- Example 1: Pure Java Agent (see page 20)
- Example 2: Pure Java Agent Using Automatic Configuration (see page 21)
- Example 3: Native Agent (see page 22)

Example 1 - Pure Java Agent

This example is based on the pure Java agent. The automatic configuration option is not used.

```
java -jar LisaAgent.jar -ia -java
Enter the path of the Java executable [C:\Program Files\Java\jre7\bin\java]:


Enter the DevTest broker URL [tcp://localhost:2009]:

NOTE: If the DevTest agent name contains the character sequence

{{x}}

it will be replaced with a unique identifier.

Enter the DevTest agent name [{{x}}]:


=====

System verification complete.  You can manually add these options to the java
command-line to start the DevTest agent:


-javaagent:C:\DevTest\agent\InsightAgent.jar=url=tcp://localhost:2009,name={{x}}

Do you need assistance to automatically configure the DevTest agent? Please enter
[Yes/Y], if no, enter any other letter, then enter
n


Please manually add these options to the Java command-line to start the DevTest agent
```

Example 2 - Pure Java Agent Using Automatic Configuration

This example is based on the configuration of JBoss 4.2 pure Java agent on Windows using the automatic configuration option.

```
java -jar LisaAgent.jar -ia -java -broker tcp://localhost:2009 -name TestAgent66
Enter the path of the Java executable [C:\Program
Files\Java\jdk1.7.0_40\jre\bin\java]:
=====
System verification complete. You can manually add these options to the java
command-line to start the DevTest agent:

-javaagent:C:\Project\DevTest\main\remote\dist\agent\InsightAgent.jar=url=tcp://l
ocalhost:2009,name=TestAgent66
Do you need assistance to automatically configure the DevTest agent? Please enter
[Yes/Y], if no, enter any other letter, then enter
y
Please enter the index in [] to choose the platform:
[1] JBoss
[2] IBM WebSphere App Server
[3] Oracle WebLogic Server
[4] TIBCO BusinessWorks
[5] WebMethods Integration Server Windows Service
[X] Exit
1
Detected version 4.2 installed on your system.
Enter path for Java agent configuration file [C:\DevTest\jboss\bin\run.bat]:
Automatic configuration of the Java Agent is complete. Please start your service.
The configuration info has been updated. The previous configuration was:
SET
JAVA_TOOL_OPTIONS=-javaagent:C:\Project\DevTest\main\remote\dist\agent\InsightAge
nt.jar=url=tcp://localhost:2009,name=TestAgent7 %JAVA_TOOL_OPTIONS%
The previous configuration has been saved to file
'C:\DevTest\jboss\bin\run.bat.2014-54-25_09-54-22'
The current agent configuration is:
SET
JAVA_TOOL_OPTIONS=-javaagent:C:\Project\DevTest\main\remote\dist\agent\InsightAge
nt.jar=url=tcp://localhost:2009,name=TestAgent66 %JAVA_TOOL_OPTIONS%
```

Example 3 - Native Agent

This example is based on the Solaris (Sparc) 32-bit native agent.

```
$ java -jar dist/agent/LisaAgent.jar -ia -native tcp://localhost:2009 MyAgent
Enter the path of the 'java' executable [...]: /usr/jdk/instances/jdk1.6.0/bin/java

=====

System verification complete.  Add these options to the java command-line to start
the DevTest agent:


-agentpath:/tmp/user/dist/agent/libJavaBinder.solaris.sparc32.so=jar=file:/tmp/us
er/dist/agent/LisaAgent.jar,url=tcp://localhost:2009,name=MyAgent

Do you need assistance to automatically configure the DevTest agent? Please enter
[Yes/Y], if no, enter any other letter, then enter

n

Please manually add these options to the Java command-line to start the DevTest agent
```

Platform-Specific Agent Installation

This section contains platform-specific information for installing the DevTest Java Agent.

This section includes the following topics:

IBM WebSphere App Server

This topic applies to IBM WebSphere Application Server 7.0 and 8.5.

To configure IBM WebSphere Application Server to use the DevTest Java Agent, use any of the following approaches:

- Administrative console
- server.xml file
- wsadmin tool

For each approach, you specify the agent parameters string as a generic JVM argument. You can use the agent install assistant (see page 18) to determine the required value. The following example is based on the pure Java agent:

```
-javaagent:/home/itko/agent/InsightAgent.jar=url=tcp://172.24.255.255:2009,name=w
as70_linux32
```

Administrative Console

In this procedure, you use the web-based Administrative console to specify the agent parameters string.

Follow these steps:

1. Go to the Administrative console.
2. Navigate to your application server.
3. Expand Java and Process Management and click the Process definition link on the Configuration tab.
4. Click Java Virtual Machine under Additional Properties.
5. Specify the agent parameters string in the Generic JVM arguments field.

server.xml File

In this procedure, you use the **server.xml** file to specify the agent parameters string.

Follow these steps:

1. Go to the **WAS_HOME/AppServer/profiles/AppSrv01/config/cells/<cell_name>/nodes/<node_name>/servers/server1** directory.
2. Open the **server.xml** file.
3. At the bottom of the file, change the **genericJvmArguments** entry.

```
    <jvmEntries ... genericJvmArguments="<agent_parameters_string>" .../>
```

wsadmin Tool

You can use the **wsadmin** tool.

In the following example, the agent parameters string is specified with the modify command of the **AdminConfig** object. The Jacl scripting language is used.

```
C:\IBM\WebSphere70\AppServer\bin>hostname
cam-aa74651f617

C:\IBM\WebSphere70\AppServer\bin>wsadmin
WASX7209I: Connected to process "server1" on node cam-aa74651f617Node01 using SOAP
connector;  The type of process is: UnManagedProcess
WASX7029I: For help, enter: "$Help help"

wsadmin>set server1 [$AdminConfig getid
/Cell:cam-aa74651f617Node01Cell/Node:cam-aa74651f617Node01/Server:server1/ ]
server1(cells/cam-aa74651f617Node01Cell/nodes/cam-aa74651f617Node01/servers/serve
r1|server.xml#Server_1255494205517)

wsadmin>set jvm [$AdminConfig list JavaVirtualMachine $server1]
(cells/cam-aa74651f617Node01Cell/nodes/cam-aa74651f617Node01/servers/server1|serv
er.xml#JavaVirtualMachine_1255494205517)

wsadmin>$AdminConfig modify $jvm genericJvmArguments "<agent_parameters_string>"

wsadmin>$AdminConfig save

wsadmin>quit
```

JBoss

This topic applies to JBoss 4.2 and 7.

To configure JBoss to use the DevTest Java Agent, you edit the JBoss startup script.

You can use the agent install assistant (see page 18) to determine the value of the agent parameters string.

Follow these steps:

1. Open the JBoss **run.bat** or **run.sh** file in a text editor.
2. Define the following property before calling the Java executable:

   ```
   set JAVA_OPTS=<agent_parameters_string>
   ```

3. Save the file.

Jetty

This topic applies to Jetty 8 and 9.*x*.

To configure Jetty to use the DevTest Java Agent, specify the agent parameters string in the **JAVA_TOOL_OPTIONS** environment variable.

You can use the agent install assistant (see page 18) to determine the value of the agent parameters string.

```
set JAVA_TOOL_OPTIONS=<agent_parameters_string>
"%JAVA_HOME%/bin/java.exe" -DSTOP.PORT=28282 -DSTOP.KEY=secret -jar
start.jar etc/jetty-logging.xml
```

Oracle WebLogic Server

This topic applies to Oracle WebLogic Server 10.3 and 12.1.1.

To configure Oracle WebLogic Server to use the DevTest Java Agent, use any of the following approaches:

- Administration console
- config.xml file
- Startup script

For each approach, you specify the agent parameters string as a server startup argument. You can use the agent install assistant (see page 18) to determine the required value. The following example is based on the pure Java agent:

```
-javaagent:/export/home/wls/agent/LisaAgent2.jar=url=tcp://172.24.255.255:2009,na
me=wls
```

Administration Console

To follow the officially supported way to add arguments to the JVM, specify the agent parameters string from the WebLogic administration console.

Follow these steps:

1. Open the WebLogic administration console.
2. In the Domain Structure panel, expand the Environments node and click the Servers node.
3. Click the Configuration tab and the Server Start subtab.
4. In the Arguments field, add the agent parameters string.
5. Save your changes.

config.xml File

Another approach is to edit the central configuration file for the domain.

Follow these steps:

1. Go to the **WEBLOGIC_HOME\user_projects\domains\base_domain\config** directory.
2. Open the **config.xml** file.
3. Add the agent parameters string to the **<server>/<server-start>/<arguments>** node of the target server.

Startup Script

You can also edit the WebLogic startup script.

In the following example, the **JAVA_TOOL_OPTIONS** environment variable is used to set the agent parameters string. These lines are located in the **startWebLogic.sh** file after the Java version check and before the actual WebLogic invocation.

```
JAVA_TOOL_OPTIONS=<agent_parameters_string>
export JAVA_TOOL_OPTIONS
```

If the startup script has been customized, the **JAVA_TOOL_OPTIONS** environment variable is not used.

TIBCO BusinessWorks

This topic applies to TIBCO BusinessWorks 5.*x*.

To configure the agent for all applications, you edit the following file:

```
...\tibco\bw\5.x\bin\bwengine.tra
```

Editing this file does not affect applications that have already been created.

To configure the agent for an existing application, edit the following file:

```
...\tibco\tra\domain\<domain-name>\application\<application-name>\<applicatio
n-name>-<service-name>.tra
```

Where:

- <domain-name> is the name of your TIBCO domain.
- <application-name> is the name of the deployed TIBCO application.
- <service-name> is the name of the deployed service.

For example:

```
D:\tibco\tra\domain\itko-tibcobw\application\MyBWProjectXml\MyBWProjectXml-it
koUserCRUD.tra
```

In either file, add the following line:

```
java.extended.properties=<agent-parameters-string>
```

You can use the agent install assistant (see page 18) to determine the string. If you want to create TIBCO BusinessWorks baselines, be sure to add the **heap** option and set the value to **true**.

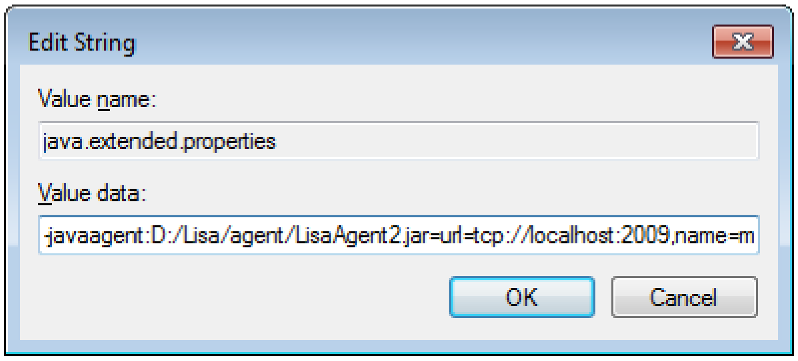
Configuration Difference for Windows Service

If the TIBCO process is running as a Windows service, you do not add the **java.extended.properties** line to the .tra file. Instead, you specify the required information in the Windows registry.

In the Windows registry editor, locate the registry key for the TIBCO BusinessWorks service for the application. The usual name is **domain-name.application-name**.

Under that registry key, locate the Parameters key. Under the Parameters key, create or edit a string value with the name **java.extended.properties**. Set the value data to the agent parameters string.

The following graphic shows an example of setting the value data.



Value name:
java.extended.properties
Value data:
-javaagent:D:/Lisa/agent/LisaAgent2.jar=url=tcp://localhost:2009,name=m

webMethods Integration Server

This topic applies to webMethods Integration Server 9.0 and 9.5.

Specify the agent parameters string in the **server.bat** or **server.sh** file. You can use the agent install assistant (see page 18) to determine the required value. The following example is based on the pure Java agent:

```
set
JAVA_TOOL_OPTIONS=-javaagent:E:/agent/InsightAgent.jar=url=tcp://172.24.255.255:2
009,name=wm
```

Follow these steps:

1. Locate the **bin** directory in the webMethods Integration Server installation.
2. Open the **server.bat** or **server.sh** file.
3. Add the agent parameters string before the line that calls the **java** executable.
4. Save the file.

Windows Service

If webMethods Integration Server is defined as a Windows service, you can edit the **server.bat** file by inserting the agent parameters string in the **/jvmargs** value that is defined in the **if "1%1"=="1-service"** switch. For example:

```
"%IS_DIR%\bin\SaveSvcParams.exe" /svcname %2 /jvm "%JAVA_DIR%\.." /binpath "%PATH%"
/classpath %CLASSPATH% /jvmargs
"-javaagent:c:/DevTest/agent/InsightAgent.jar=name=MyIS %JAVA2_MEMSET%" /progargs
"%IS_DIR%\bin\ini.cnf"#"-service
%2"#%PREPENDCLASSES_SWITCH%#%PREPENDCLASSES%#%APPENDCLASSES_SWITCH%#%APPENDCLASSE
S%#%ENV_CLASSPATH_SWITCH%#%ENV_CLASSPATH%#%3#%4#%5#%6#%7#%8#%9
```

Wily Introscope Agent

The DevTest Java Agent can coexist with the Wily Introscope Agent.

The Wily Introscope Agent is a pure Java agent and is typically configured with the **-javaagent** syntax.

If you are using the **-javaagent** syntax for both agents, specify the DevTest Java Agent *before* the Wily Introscope Agent.

For a typical application server or container, the following example is appropriate:

```
set
JAVA_OPTS=-javaagent:E:\DevTest\agent\InsightAgent.jar=url=tcp://localhost:2009,n
ame=DevTestAgent -javaagent:e:/wily/Agent.jar %JAVA_OPTS%
```

This example is also appropriate:

```
set JAVA_OPTS=-javaagent:e:/wily/Agent.jar %JAVA_OPTS%
set
JAVA_TOOL_OPTIONS=-javaagent:E:\DevTest\agent\InsightAgent.jar=url=tcp://localhos
t:2009,name=DevTestAgent
```

However, this example is incorrect:

```
set JAVA_OPTS=-javaagent:e:/wily/Agent.jar
-javaagent:E:\DevTest\agent\InsightAgent.jar=url=tcp://localhost:2009,name=DevTes
tAgent %JAVA_OPTS%
```

Remember to perform the following somewhere along the line:

```
-Dcom.wily.introscope.agentProfile=e:/wily/core/config/IntroscopeAgent.profile
```

Java Agent Usage

This section contains the following topics:

Start the Broker

If the broker is not installed as a Windows service, start the broker manually.

To view agents in the Enterprise Dashboard and the DevTest Portal, the broker must be running.

Follow these steps:

1. Ensure that the registry is running.
2. Do one of the following actions:
   - Open a command prompt, navigate to the **LISA_HOME\bin** directory, and run the **Broker** executable.
   - If your installation has a Start menu folder, click Start menu, All Programs, DevTest Solutions, Broker.

rules.xml File

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

The **rules.xml** file is located in the **LISA_HOME** directory on the computer where the broker is running. The **rules.xml** file is generated when the broker is started for the first time. All agents that are connected to the broker use the settings.

By default, the **rules.xml** file includes only a sample set of configuration properties. All the properties and their default values are maintained internally.

You can view these properties in a file named **rules.xml.sample**. The **rules.xml.sample** file is generated when the broker is started for the first time. The **rules.xml.sample** file is located in the same directory as the **rules.xml** file.

Each property includes a comment, the name, and the value. For example:

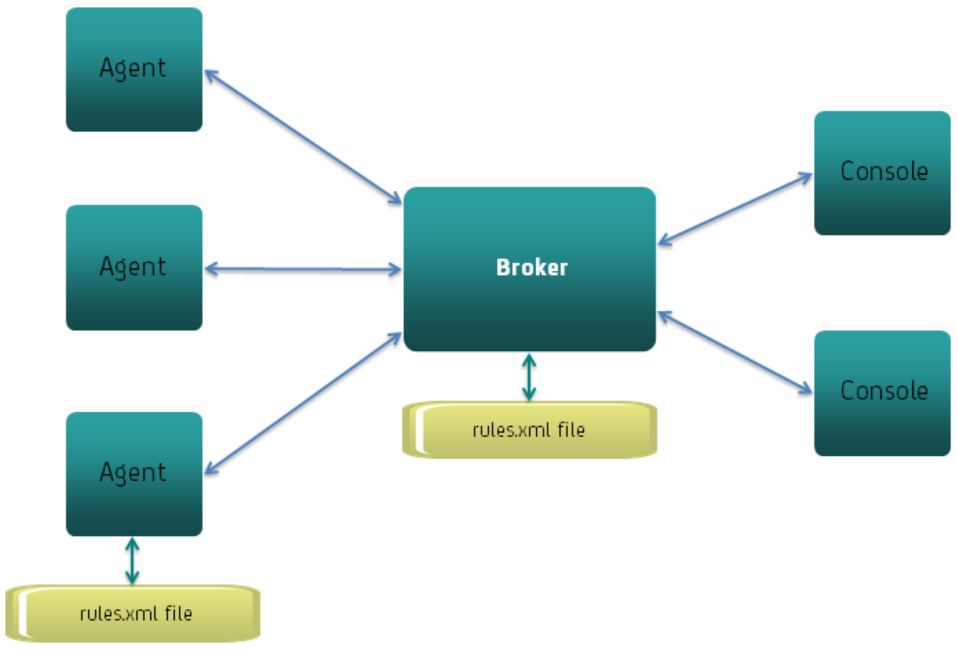
```
<property comment="Keep track of file descriptors"
key="lisa.agent.tracking.disabled" value="false"/>
```

The following XML shows the general format of the **rules.xml** file. The **agent** element contains the properties for an agent. The **broker** element contains the properties for the broker. The **console** element contains the properties for the consoles.

```
<?xml version="1.0" encoding="UTF-8"?>
<rules>
   <agent guid="0" name="A1">
      ...
   </agent>
   <broker>
      ...
   </broker>
   <console>
      ...
   </console>
</rules>
```

You can also place a **rules.xml** file on the agent side or the console side. The settings in these files override any information for that agent or console on the broker side.

The following graphic shows an example configuration. The broker accesses the **rules.xml** file that is on the same computer. Multiple agents and consoles are connected to the broker. One of the agents has its own **rules.xml** file.



Note: The following properties can be configured only on the agent side because the agent reads them before making a connection to the broker:

- lisa.agent.agent.log
- lisa.agent.log.max.size
- lisa.agent.log.max.archives
- lisa.agent.security.manager.disabled
- lisa.agent.transaction.auto.start
- lisa.agent.transaction.auto.start.max.delay
- lisa.broker.encryption.token
- lisa.log.level

You can add *directives* to the **rules.xml** file to perform certain functions. For information about the directives, see the following topics:

- **category** directive: Category Settings (see page 43)
- **database** directive: Configure a Database Sink (see page 42)
- **exclude** directive: Excluding from Interception and Virtualization (see page 46)
- **feature** directive: Protocol Weight Configuration (see page 37)
- **intercept** directive: Adding a Method to Intercept (see page 45)
- **loadbalancer** directive: Load Balancers and Native Web Servers (see page 79)
- **response** directive: Java Agent Auto-Response Generation (see page 48)

- **track** directive: Add a Class for Tracking (see page 44)
- **virtualize** directive: Instrumentation Rules for VSE (see page 40)

Managing Agents in Groups

You can create a *group* that contains two or more agents.

You can then apply configuration changes at the group level, instead of making the same changes to each agent individually.

Note: An agent cannot belong to more than one group.

An agent that does not belong to a group is referred to as a *stand-alone* agent.

The following XML shows the general format of a **rules.xml** file that has a group. The **group** element contains the properties for the group. The **agent** element contains the properties for an agent. If the agent is part of a group, then the **agent** element appears within the **group** element. The **broker** element contains the properties for the broker. The **console** element contains the properties for the consoles.

```
<?xml version="1.0" encoding="UTF-8"?>
<rules>
   <group name="G1">
      ...
      <agent name="G1A1">
         ...
      </agent>
   </group>
   <agent guid="0" name="A1">
      ...
   </agent>
   <broker>
      ...
   </broker>
   <console>
      ...
   </console>
</rules>
```

If an **agent** element within a **group** element has one or more properties, the properties override the group settings.

Do not place the **broker** element or the **console** element inside the **group** element.

When you update an agent property from the DevTest Portal, the setting is saved to the stand-alone agent section of the **rules.xml** file.

Note: For detailed information about the DevTest Portal, see *Using CA Continuous Application Insight*.

Example: Configure One Group in rules.xml File

The following XML shows a **rules.xml** file that has one group. The group has three agents. This file also has a stand-alone agent.

```
<?xml version="1.0" encoding="UTF-8"?>
<rules>
   <group name="group1">
      <property key="lisa.agent.jms.poll.int" value="5000"/>
      <property key="lisa.agent.transaction.auto.start.max.delay"
value="60000"/>
      <property key="lisa.agent.virtualize.jit.enabled"
value="false"/>
      <intercept class="com.itko.examples.entity.Account"
method="setName" signature="(Ljava/lang/String;)V"/>
      <agent name="agent1" guid="1234567">
         <property key="lisa.agent.stats.alarm.threshold.permgen"
value="90"/>
         <property key="lisa.agent.stats.sampling.interval"
value="1000"/>
      </agent>
      <agent name="agent2" guid="2345678" />
      <agent name="agent3" guid="3456789" />
   </group>
   <agent guid="-2032180703" name="DEFAULT">
      ...
   </agent>
   <broker>
      ...
   </broker>
   <console>
      ...
   </console>
</rules>
```

Protocol Weight Configuration

The **feature** directive lets you modify the capture level for each protocol that the DevTest 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.

You can also modify the capture levels from the DevTest Portal. For more information, see *Using CA Continuous Application Insight.*

The **feature** directive has the following format:

```
<feature name="protocol_name" weight="weight"/>
```

The **feature** directive can be placed within the **group** element or the **agent** element of the **rules.xml** file.

Set the **weight** attribute to 0, 4, or 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.

The following example sets the JDBC protocol to the **Full Data** level:

```
<feature name="JDBC" weight="8"/>
```

Signature Specification

The following directives in the **rules.xml** file include a signature specification:

- exclude
- intercept
- respond
- virtualize

You use the signature specification to describe the characteristics of a Java method signature.

The first portion of the specification is the word **signature**, followed by an equal sign and a quotation mark.

The second portion of the specification contains the arguments, surrounded by parentheses. If the method has no arguments, you must still include the parentheses.

The third portion of the specification contains the return type, followed by a quotation mark. If the return type is void, use the letter V.

Do not include any spaces within the specification.

To specify a primitive type in the arguments or the return type, use one of the following letters:

Letter	**Primitive Type**
Z	boolean
B	byte
C	char
D	double
F	float
I	int
J	long
S	short

To specify a fully qualified class, do the following steps:

- Add the letter L at the beginning.
- Use a forward slash as the separator, instead of a dot.
- Add a semicolon at the end.

For example:

```
Ljava/lang/String;
```

Example: One Argument, Returns Void

Assume that you want to intercept the **onMessage()** method of the **javax.jms.MessageListener** interface. This method has the following signature:

- The argument is a **javax.jms.Message** object.
- The return type is void.

The signature specification in the intercept rule would be:

```
signature="(Ljavax/jms/Message;)V"
```

Example: No Arguments, Returns Primitive Type

Assume that you want to intercept the **getPriority()** method of the **javax.jms.MessageProducer** interface. This method has the following signature:

- The method has no arguments.
- The return type is an integer.

The signature specification in the intercept rule would be:

```
signature="()I"
```

Instrumentation Rules for VSE

You can customize the VSE functionality by adding the **virtualize** directive to the **rules.xml** file of the agent.

The **virtualize** directive can be placed within the **group** element or the **agent** element of the **rules.xml** file.

Adding a Class for Virtualization (Both Recording and Playback Modes)

```
<virtualize class="class_name"/>
```

Example:

```
<virtualize class="javax.ejb.SessionBean"/>
```

Telling the Agent how to Determine What Constitutes a Session for a Given Protocol

You perform this task by providing a code snippet that returns a session identifier.

```
<virtualize>
   <track class="class_name" method="method_name"
signature="signature" push="true|false">
      <code><![CDATA[" and ends with "]]></code>
   </track>
</virtualize>
```

The format of the signature is described in Signature Specification (see page 38).

Examples:

```
<virtualize>
   <track class="javax.servlet.http.HttpServlet" method="service"
signature="(Ljavax/servlet/http/HttpServletRequest;Ljavax/servlet/
http/HttpServletResponse;)V" push="false">
      <code><![CDATA["return $1.getSession().getId();"]]></code>
   </track>
</virtualize>
```

```
<virtualize>
   <track class="javax.ejb.SessionBean" method="setSessionContext"
signature="(Ljavax/ejb/SessionContext)V" push="true">
      <code><![CDATA["return
$1.getEJBObject().getHandle().toString();"]]></code>
```

```
   </track>
</virtualize>
```

```
<virtualize>
   <track class="javax.ejb.EntityBean" method="setEntityContext"
signature="(Ljavax/ejb/EntityContext)V" push="true">
      <code><![CDATA["return
$1.getPrimaryKey().toString();"]]></code>
   </track>
</virtualize>
```

These examples are hard-coded in the agent and thus unnecessary in your **rules.xml** file. However, these lines show how the process works so it can be implemented for any number of protocols, other than HTTP and EJB without recompiling the agent.

The value of the **class** attribute is the class from which we gain access to a session.

The value of the **method** and **signature** attributes determine the method that, when invoked, computes the session identifier. This computation uses the value of the **code** attribute. **$0** represents the source object. **$1**, **$2**, and so on, are the method arguments.

The **push** attribute determines how we store that session for later use by VSE frames.

- **push="true"** specifies that the container sets the session, and we keep a mapping of object to session.
- **push="false"** specifies that session is thread-scoped and we store in a thread-local variable.

The innermost session (identifier) is served back through VSE in the **com.itko.lisa.remote.vse.VSEFrame getSessionId()** method. For more information, see the JavaDocs in the **LISA_HOME\doc** directory.

Configure a Database Sink

The **database** directive lets you configure database access for console applications such as the CAI command-line tool. Console applications do not access the database through the broker.

You add the **database** directive to the **broker** element of the **rules.xml** file.

This setting stays in use for the lifetime of the agent.

```
<database driver="mySinkDriver" url="mySinkURL" user="mySinkUser"
password="mySinkPassword"/>
```

Example:

```
<database driver="org.postgresql.Driver"
url="jdbc:postgresql://localhost:5432/agtdb" user="postgres"
password="postgres"/>
```

Category Settings

The DevTest Java Agent assigns a category to each transaction frame. If the agent cannot determine the category, the default category is assigned.

You can configure the agent to associate an intercepted class or interface with one of the non-default categories. Add the **category** directive to the **rules.xml** file of the agent. Specify the class or interface name and the category number.

The **category** directive has the following format:

```
<category class="class_or_interface_name" value="category_number"/>
```

The **category** directive must be placed within the **broker** element.

The valid category numbers are:

- CATEGORY_DEFAULT = 0
- CATEGORY_THREAD = 5
- CATEGORY_THROWABLE = 10
- CATEGORY_GUI = 7
- CATEGORY_GUI_SWT = 9
- CATEGORY_LOGGING = 15
- CATEGORY_WEB_HTTP = 20
- CATEGORY_WEB_HTTPS = 21
- CATEGORY_WS_HTTP = 22
- CATEGORY_WS_HTTPS = 23
- CATEGORY_REST_HTTP = 24
- CATEGORY_RMI = 30
- CATEGORY_RMI_HTTP = 31
- CATEGORY_RMI_SSL = 32
- CATEGORY_EJB = 40
- CATEGORY_JDBC = 50
- CATEGORY_JCA = 55
- CATEGORY_JMS = 60
- CATEGORY_MQ = 61
- CATEGORY_WM = 62
- CATEGORY_TIBCO = 64
- CATEGORY_AMX = 70

- CATEGORY_FRAMEWORK = 80
- CATEGORY_CLIENT = 90
- CATEGORY_CICS = 100
- CATEGORY_WPS = 120
- CATEGORY_SAP = 130
- CATEGORY_SYNTHETIC_ROOT = 140
- CATEGORY_DN_DEFAULT = 1000
- CATEGORY_DN_REMOTING = 1010
- CATEGORY_DN_SQL = 1020

Example:

```
<category class="com.mycompany.GuiClass" value="7"/>
```

Note: While possible, it is not recommended to change the categories of transaction frames that are already assigned a non-default category by the agent.

Add a Class for Tracking

The **track** directive allows instances to be easily retrieved from the heap later.

The **track** directive can be placed within the **group** element or the **agent** element of the **rules.xml** file.

```
<track class="class_name"/>
```

Example:

```
<track class="java.io.File"/>
```

Adding a Method to Intercept

You can add a method for the DevTest Java Agent to intercept by adding the **intercept** directive to the **rules.xml** file of the agent.

The **intercept** directive has the following format:

```
<intercept class="class_name" method="method_name"
signature="signature"/>
```

The **intercept** directive can be placed within the **group** element or the **agent** element of the **rules.xml** file.

The format of the signature is described in Signature Specification (see page 38).

Class hierarchies are considered. Assume that class B extends class A, and both classes define method M. If you intercept method M of class A, then method M of class B is also captured.

By default, CA Continuous Application Insight does not capture getter and setter methods. To add a getter or setter method, include the **delay** attribute with the value set to true.

Examples

The following example adds the **setName()** method of the **com.itko.examples.entity.Account** class. The method takes in a **java.lang.String** object as an argument. The method returns void.

```
<intercept class="com.itko.examples.entity.Account"
method="setName" signature="(Ljava/lang/String;)V" delay="true"/>
```

The following example adds the **getTransactions()** method of the **com.itko.examples.entity.Account** class. The method takes in no arguments. The method returns a **java.util.Collection** object.

```
<intercept class="com.itko.examples.entity.Account"
method="getTransactions" signature="()Ljava/util/Collection;"
delay="true"/>
```

The following example adds the **getRequestTypes()** method of the **com.itko.examples.airline.ws.jaxws.Request** class. The method takes in no arguments. The method returns an array of **java.lang.String** objects.

```
<intercept class="com.itko.examples.airline.ws.jaxws.Request"
method="getRequestTypes" signature="()[Ljava/lang/String;"
delay="true"/>
```

Excluding from Interception and Virtualization

You can prevent the DevTest Java Agent from intercepting or virtualizing a method, class, or package by adding the **exclude** directive to the **rules.xml** file of the agent.

The **exclude** directive has the following format:

```
<exclude class="class_name" method="method_name"
signature="signature"/>
```

The **exclude** directive can be placed within the **group** element or the **agent** element of the **rules.xml** file.

The format of the signature is described in Signature Specification (see page 38).

Class hierarchies are not considered. Assume that class B extends class A, and both classes define method M. If you exclude class A, then method M of class A is not captured. However, method M of class B is captured.

If you want to exclude a class or package irrespective of method or signature, you can do either:

- Specify **method="*" signature="*"**
- Omit the **method** and **signature** attributes

If you add or remove the **exclude** directive, you must restart the agent.

By default, CA Continuous Application Insight does not capture getter and setter methods. You do not need to exclude getter and setter methods.

Examples

The following example excludes the **serviceComposition()** method of the **com.itko.examples.ejb3.OrdinaryBean** class. The method takes in no arguments. The method returns a **java.lang.String** object.

```
<exclude class="com.itko.examples.ejb3.OrdinaryBean"
method="serviceComposition" signature="()Ljava/lang/String;"/>
```

The following example excludes the **com.itko.examples.ejb3.OrdinaryBean** class.

```
<exclude class="com.itko.examples.ejb3.OrdinaryBean"/>
```

The following example excludes the **com.itko.examples.ejb** package. Notice the use of a single wildcard character.

```
<exclude class="com.itko.examples.ejb.*"/>
```

The following example excludes the **com.itko.examples** package and all its subpackages. Notice the use of two wildcard characters.

```
<exclude class="com.itko.examples.**"/>
```

Java Agent Auto-Response Generation

You can use the DevTest Java Agent to record transactions even when the back-end is not available.

To configure this feature, add the **respond** directive to the **rules.xml** file. The **respond** directive has the following format:

```
<respond class="class_or_interface_name" method="method_name"
signature="signature" source="string_value" args="string_value"
return="class_name"/>
```

The **respond** directive must be placed within the **agent** element of the **rules.xml** file.

The **class** attribute is the only required attribute.

Use the following attributes to specify the method or methods that you want to the agent to intercept:

- **class**: Defines the name of the class or interface that defines the method or methods. If you specify an interface, any class that implements the interface is also included.
- **method**: Defines the name of a method. If you do not include this attribute, all methods of the class or interface are included.
- **signature**: Defines the signature of the method. The format is described in Signature Specification (see page 38).
- **source**: If calling the **toString()** method on the object contains this string value, then intercept the method.
- **args**: If calling the **toString()** method on the arguments contains this string value, then intercept the method.

When the agent intercepts a method call that matches the specification, the agent performs the following actions:

- Makes an educated guess as to what type of object should be returned.
- Returns a generic version of the object. The object contains random values.

You can use the **return** attribute to specify the object type, thus overriding the first action.

If a value contains a less than sign, replace the less than sign with the following text:

```
&lt;
```

If a value contains a greater than sign, replace the greater than sign with the following text:

```
&gt;
```

Example: Auto-Response Generation for EJB Calls

The following Java method creates a JNDI context object, calls a server that is able to run EJBs, and invokes APIs.

```
public void doEJBCall() throws Exception
    {
        Properties props = new Properties();
        props.put(Context.INITIAL_CONTEXT_FACTORY,
"org.jnp.interfaces.NamingContextFactory");
        props.put(Context.PROVIDER_URL, "jnp://localhost:1099");

        /* 1st (optionally) remote call */
        Context ctx = new InitialContext(props);

        /* 2nd remote call */
        EJB3UserControlBeanRemote remote = (EJB3UserControlBeanRemote)
ctx.lookup("EJB3UserControlBean/remote");

        /* 3rd remote call */
        User u = remote.getUser("lisa_simpson");

        /* Local calls... */
        System.out.println(u.getEmail());
    }
```

Suppose that the server is not available. If you run this method, the code does not succeed because it cannot obtain a connection.

You can use the following **respond** directives to force the code to succeed. The first directive intercepts the creation of the **InitialContext** object. The second directive intercepts the **lookup()** method of the **Context** object. The third directive intercepts any method of the **EJB3UserControlBeanRemote** object.

```
<respond class="javax.naming.InitialContext" method="&lt;init&gt;"
args="1099"/>

<respond class="javax.naming.Context" method="lookup"
args="EJB3UserControlBean/remote"/>

<respond class="com.itko.examples.ejb3.EJB3UserControlBeanRemote"/>
```

Developing Against the Java Agent

The main interface to the DevTest Java Agent from the client side is **com.itko.lisa.remote.client.AgentClient**. This class has methods to invoke APIs on the agents or the broker, to discover agents and to be notified of their status changes (online/offline).

This class also gives access to the classes responsible for the agent interaction in the main areas of functionality:

- **com.itko.lisa.remote.client.AgentClient**
- **com.itko.lisa.remote.client.DiscoveryClient**
- **com.itko.lisa.remote.client.TransactionsClient**
- **com.itko.lisa.remote.client.VSEClient**

Note: You can view detailed information about these classes in the JavaDocs for the agent. The JavaDocs are located in the **LISA_HOME\doc** directory.

This section contains the following topics:

Agent General APIs

Note: You can view detailed information about the interface and class that this topic describes in the JavaDocs for the agent. The JavaDocs are located in the **LISA_HOME\doc** directory.

Most of the client APIs must specify an agent as their target. This specification is accomplished by passing a parameter of type **com.itko.lisa.remote.IAgentInfo**. This parameter represents an object that uniquely identifies an agent and some of its basic information.

```
/** A unique identifier for this agent or console. If it is named the guid is
persistent across VM lifespans */
 public long getGuid();
 public void setGuid(long guid);

 /** A human-readable name to identify this agent, manually given or generated
from system properties */
 public String getName();
 public void setName(String name);

 /** The name of the machine this object was generated on (if available) */
 public String getMachine();
 public void setMachine(String machine);

 /** The IP of the machine this object was generated on (if available) */
 public String getIp();
 public void setIp(String ip);

 /** The working directory of the JVM this object runs in */
 public String getWorkingDir();
 public void setWorkingDir(String workingDir);

 /** The classpath as it is returned by the java.class.path property */
 public String getClasspath();
 public void setClasspath(String classpath);

 /** The library path as it is returned by the java.library.path property */
 public String getLibpath();
 public void setLibpath(String libpath);

 /** For agents, returns the java class containing the main method that was invoked
*/
 public String getMainClass();
 public void setMainClass(String mainClass);

 /** A short-hand version of the command-line (for representation purposes as it
may not be accurate) */
 public String getCommandLine();
 /** Transient field to keep track of when this object is sent or received */
 public Date getGenerationTime();
 public void setGenerationTime(Date time);

 /** Transient field to keep track of a required password to invoke APIs on this
agent over JMS */
 public String getToken();
 public void setToken(String token);
```

We can now look at some of the APIs directly off **com.itko.lisa.remote.client.AgentClient**. This list is not exhaustive but covers most of the needs of most clients.

```
/** Gets the class responsible for all discovery and information for a given agent
*/
 public DiscoveryClient getDiscoveryClient();

 /** Gets the class responsible for all transaction related operations */
 public TransactionsClient getTransactionClient();

 /** Gets the class responsible for all VSE related operations */
 public VSEClient getVSEClient();

 /**
  * Get all the discovered agents - this is the main API to get IAgentInfos used
in all other API calls
  * @param tokens a map of agent guids or names to tokens, null if no agent has
token-enabled security
  * @return a set of IAgentInfo objects representing agents that are currently
online
  */
  public Set getRemoteAgentInfos(Map tokens);

 /**
  * Forward agent online information to registered listeners
  * @param info the agent that was just detected to come online
  */
  public void onAgentOnline(IAgentInfo info);

 /**
  * Forward agent offline information to registered listeners
  * @param info the agent that was just detected to go offline
  */
  public void onAgentOffline(IAgentInfo info);

 /**
  * Evaluate arbitrary code on the specified agent
  * @param info the agent this code will be evaluated against
  * @param input the code to execute. The variable '_agent' represents the Agent
instance.
```

```
 * @return the value returned by the code
 * @throws JMSInvocationExceptionthrown if the code throws on the agent
 */
public Object eval(IAgentInfo info, String input) throws JMSInvocationException
```

Agent Discovery APIs

The **com.itko.lisa.remote.client.DiscoveryClient** class provides methods pertaining to discovering data on an agent. You can get the **DiscoveryClient** class with the following call: **AgentClient.getInstance().getDiscoveryClient()**.

Note: You can view detailed information about this class in the JavaDocs for the agent. The JavaDocs are located in the **LISA_HOME\doc** directory.

```
/**
  * The system properties of the specified agent
  * @param info
  * @return
  * @throws JMSInvocationException
  */
 public Map getVMProperties(IAgentInfo info) throws JMSInvocationException;

 /**
  * Returns a list of StatsFrames recorded for the specified agent between the
from and the to dates.
  * @param agentInfo the agent for which to retrieve statistics
  * @param from      how far back to filter
  * @param to        how recently to filter
  * @return      the desired StatsFrames list ordered by decreasing time (starting
at the to date)
  */
 public List getStatistics(IAgentInfo agentInfo, Date from, Date to);

 /**
  * Returns a list of StatsFrames recorded for the specified agent between the
from and the to dates.
  * @param agentInfo the agent for which to retrieve statistics
  * @param from      how far back to filter
  * @param to        how recently to filter
  * @param interval  how to aggregate the results in seconds. 10 means average
the results of every 10 secs, etc...
  * @param limit     the maximum number of results
  * @return      the desired StatsFrames list ordered by decreasing time (starting
at the to date)
  */
 public List getStatistics(IAgentInfo agentInfo, Date from, Date to, int
interval, int limit);

 /**
  * TODO: (re)implement - currently will throw
  * @param info
  * @return
  * @throws JMSInvocationException
  */
 public Topology getTopology(IAgentInfo info) throws JMSInvocationException;
 /**
  * Exit points are MethodInfo that capture classes/methods that make network
calls down the stack
  * @param info
  * @return
  * @throws JMSInvocationException
  */
 public Set getExitPoints(IAgentInfo info) throws JMSInvocationException;

 /**
  * Returns the name of the J2EE container (or java if it's not a J2EE container)
  * @param info
  * @return
  * @throws JMSInvocationException
  */
 public String getServerInfo(IAgentInfo info) throws JMSInvocationException;

 /**
  * Returns the web applications deployed in the specified J2EE container
  * @param info
  * @return
  */
 public WebApplication[] getWebApps(IAgentInfo info) throws
JMSInvocationException;

 /**
  * Returns the JNDI hierarchy on the specified agent represented by a ClassNode
tree
  * @param info
  * @return
  * @throws JMSInvocationException
  */
 public ClassNode getJNDIRoot(IAgentInfo info) throws JMSInvocationException;

 /**
  * The current threads on the agent VM
  * @param info
  * @return
  * @throws JMSInvocationException
  */
 public ThreadInfo[] getThreadInfos(IAgentInfo info) throws
JMSInvocationException;

 /**
  * The current threads stacks on the agent VM
  * @param info
  * @return
  */
 public String[] dumpThreads(IAgentInfo info) throws JMSInvocationException;

 /**
  * The set of all files in the classpath of the specified agent
  * @param info
  * @return
  * @throws JMSInvocationException
  */
 public Set getClasspath(IAgentInfo info) throws JMSInvocationException;

 /**
  * Returns the class hierarchy under the specified path
  * @param info
  * @param fromPath
  * @return
  */
 public ClassNode getClassNodes(IAgentInfo info, String fromPath) throws
JMSInvocationException;

 /**
  * The class hierarchy found in the archive at the given url
  * @param info
  * @param url
  * @return
  * @throws JMSInvocationException
  */
 public ClassNode getArchiveNodes(IAgentInfo info, URL url) throws
JMSInvocationException;

 /**
  * A set containing data about the class (fields/methods/src)
  * @param info
  * @param className
  * @return
  * @throws JMSInvocationException
  */
 public Set getClassInfo(IAgentInfo info, String className) throws
JMSInvocationException;

 /**
  * Decompile and return the source to a class
  * @param info
  * @param clazz
  * @param loc decompile on the client or in the agent
  * @return
  * @throws JMSInvocationException
  */
 public String getClassSrc(IAgentInfo info, String clazz, boolean loc) throws
JMSInvocationException, IOException;
```

```
 /**
  * Returns the hierarchy this class belong to, i.e., all ancestors but also all
extenders/implementers
  * @param info
  * @param className
  * @return
  * @throws JMSInvocationException
  */
 public ClassNode[] getClassHierarchy(IAgentInfo info, String className) throws
JMSInvocationException;

/**
  * Returns (references to) all objects in the heap of the specified class - use
with caution
  * @param info
  * @param className
  * @return
  * @throws JMSInvocationException
  */
 public ClassNode getInstancesView(IAgentInfo info, String className) throws
JMSInvocationException;

 /**
  * Returns (references to) all objects on the heap tracked by the agent
  * @param info
  * @return
  * @throws JMSInvocationException
  */
 public ClassNode getTrackedObjects(IAgentInfo info) throws
JMSInvocationException;

 /**
  * A crude graph representation of an object (recursively computed fields)
  * @param info
  * @param clazz
  * @param hashCode
  * @return
  * @throws JMSInvocationException
  */
 public ClassNode getObjectGraph(IAgentInfo info, String clazz, int hashCode)
throws JMSInvocationException;

 /**
  * Gets the path from an object to a GC root
  * @param info
  * @param clazz
  * @param hashCode
  * @return
  * @throws JMSInvocationException
  */
 public ClassNode getRootPath(IAgentInfo info, String clazz, int hashCode) throws
JMSInvocationException;

 /**
  * Gets a file on the agent filesystem, downloads it to the client in a temp
location and return a handle to it
  * @param info
  * @param file
  * @return
  * @throws JMSInvocationException
  */
 public File getFile(IAgentInfo info, String file) throws
JMSInvocationException, IOException;
```

Agent Transaction APIs

Note: You can view detailed information about these classes in the JavaDocs for the agent. The JavaDocs are located in the **LISA_HOME\doc** directory.

A transaction is a code path executed by one or more servers as the result of a client request. A transaction is represented by a tree structure that is rooted at the client initiating the request. The nodes of the tree are **com.itko.lisa.remote.transactions.TransactionFrame** objects. These objects encapsulate information about a class, method, and arguments that were invoked as part of the server processing. Frames also contain ancillary information, such as the duration, the time of execution, and the thread in which it occurred.

You can think of a transaction as a method call stack. One difference is that transactions cross thread, process, or even computer boundaries. Another difference is that stacks contains all of the methods involved in the code execution of a thread, whereas transactions skip some levels and have frames only for chosen methods of interest. Such methods are referred to as *intercepted methods*.

The main way to obtain and work with **TransactionFrame** objects is through a few overloads of the following APIs supplied by **com.itko.lisa.remote.client.TransactionsClient,** which in turn can be obtained with the following call: **AgentClient.getInstance().getTransactionsClient()**.

```
/**
  * Start recording transactions
  * @param info the agent to start recording on
  */
 public void startXRecording(IAgentInfo info) throws JMSInvocationException;

 /**
  * Stop recording transactions
  * @param info the agent to stop recording on
  */
 public void stopXRecording(IAgentInfo info) throws JMSInvocationException;

 /**
  * Start tracking socket usage on the client to use in reconciling client and
server transactions.
  * This must be called prior to the call we're adding in addClientTransaction.
  * @param global install on all sockets or only on this thread
  */
 public void installSocketTracker(boolean global);

 /**
  * Stop tracking socket usage on the client to use in reconciling client and server
transactions.
  * This should be called after the call we're adding in addClientTransaction.
  * @param global uninstall on all sockets or only on this thread
  */
 public void uninstallSocketTracker(boolean global);

 /**
  * As a client, you can invoke this method to root a transaction tree at a new
client transaction created using
  * the specified parameters.
  * Note: you must call installSocketTracker before you initiate the client side
transaction you're adding here
  * and it is recommended you call uninstallSocketTracker after you're done with
the network call.
  * @param stamp     a unique string you can later use to identify the root
transaction
  *       (in calls to getTransactions for ex.)
  * @param stepInfo  a nice human-readable string that tells what the transaction
is doing (can be null)
  * @param args      the parameters the client passes to the transaction (can be
empty)
  * @param result    the result of the transaction (can be null)
  * @param duration  the client's view of the transaction duration
  */
 public void addClientTransaction(String  stamp, String stepInfo, Object[] args,
Object result, long duration);

 /** Delete all transactions and dependent data that originated from this client.
*/
 public void clearTransactions();

 /**
  * Gets a flat list of TransactionFrames received from all agents that satsfy
the filters passed as arguments
  * @param offset    offset
  * @param limit     max number of results
  * @param category  filter by transaction category (see
TranactionFrame.CATEGORY_XXX - 0 for no filter)
  * @param clazz     filter by class name (null or "" for no filter)
  * @param method    filter by method name (null or "" for no filter)
  * @param minTime   filter by frame duration greater than minTime
  * @return        a list of TransactionFrames satisfying the supplied criteria
ordered by decreasing time
  */
 public List getTransactions(int offset, int limit, int category, String clazz,
String method, int minTime);

 /**
  * Get a list of transaction trees rooted at the specified transaction(s) and
satisfying the specified criteria
  * @param offset    offset
  * @param limit     max number of results
  * @param stamps   an array of root client transaction stamps - returns all if
this is empty
  * @param minTime  duration below which transactions are pruned out of the results
  * @return ret      a list of transaction trees matching the criteria ordered
by decreasing time
  */
 public List getTransactionsTree(int offset, int limit, String[] stamps, int
minTime)
```

The parameters to these APIs do not specify an **Agent** or **AgentInfo** because transactions can span multiple agents.

Those APIs return lists of **com.itko.lisa.remote.transactions.TransactionFrame** (or trees thereof), so let us look at what data they encapsulate:

```
/** A unique identifier for this frame */
 public String getFrameId();
 public void setFrameId(String frameId);

 /** The frame id of this frame's parent frame
 public String getParentId();
 public void setParentId(String parentId);

 /** An identifier shared by all frames belonging to the same transaction (same
as global root frame id) */
 public String getTransactionId();
 public void setTransactionId(String transactionId);

 /** The parent TransactionFrame object */
 public TransactionFrame getParent();
 public void setParent(TransactionFrame parent);

 /** The list of child TransactionFrame objects */
 public List getChildren();
 public void setChildren(List children);

 /** The unique identifier of the Agent this frame was recorded in */
 public long getAgentGuid();
 public void setAgentGuid(long agentGuid);

 /** Increasing counter that helps order the frames (time may not be precise
enough) */
 public long getOrdinal();
 public void setOrdinal(long ordinal);

 /** Network incoming or outgoing frames set this to tell us what IP they are
talking from */
 public String getLocalIP();
 public void setLocalIP(String ip);
 /** Network incoming or outgoing frames set this to tell us what port they are
talking from */
 public int getLocalPort();
 public void setLocalPort(int port);

 /** Network incoming or outgoing frames set this to tell us what IP they are
talking to */
 public String getRemoteIP();
 public void setRemoteIP(String ip);

 /** Network incoming or outgoing frames set this to tell us what port they are
talking to */
 public int getRemotePort();
 public void setRemotePort(int port);

 /** The name of the thread this frame was recorded in */
 public String getThreadName();
 public void setThreadName(String threadName);

 /** Name of the class or interface this frame was recorded in (as per the
interception spec) */
 public String getClassName();
 public void setClassName(String className);

 /** Name of the class of the actual object this frame was recorded in */
 public String getActualClassName();

 /** Name of the method this frame was recorded in */
 public String getMethod();
 public void setMethod(String method);

 /** Signature of the method this frame was recorded in */
 public String getSignature();
 public void setSignature(String signature);

 /** Formatted representation of the object this frame was recorded in */
 public String getSource();
 public void setSource(Object source);

 /** Formatted representation of the arguments to the method this frame was
recorded in */
 public String[] getArguments();
 public void setArguments(Object[] arguments);

 /** Formatted representation of the result of the method this frame was recorded
in */
 public String getResult();
 public void setResult(Object result);
 /** Number of times this frame was duplicated within its parent */
 public long getHits();
 public void setHits(long hits);

 /** Server time at which the frame was recorded */
 public long getTime();
 public void setTime(long time);

 /** Wall clock duration this frame took to execute */
 public long getClockDuration();
 public void setClockDuration(long clockDuration);

 /** CPU duration this frame took to execute */
 public long getCpuDuration();
 public void setCpuDuration(long cpuDuration);

 /** Custom representation of state associated with this frame */
 public String getState();
 public void setState(Object state);

 /** Formatted LEK info as encoded/decoded by the LEKEncoder class */
 public String getLekInfo();
 public void setLekInfo(String lekInfo);

 /** Pre-computed category this frame belongs to - see
TransactionFrame.CATEGORY_XXX */
 public int getCategory();
 public void setCategory(int category);

 /** Bitwise or'ed combination of various internal pieces of information - see
TransactionFrame.FLAG_XXX */
 public long getFlags();
 public void setFlags(long flags);
```

Agent VSE APIs

VSE enables you to stub out processes, services, or parts of them along well-defined boundaries. The internal elements of these processes and services can be replaced with a layer run by DevTest according to custom user-defined rules. These layers that DevTest runs are known as a model. Usually, a default starting point for those rules is obtained from a recording of live system interactions.

DevTest already supports VSE for the HTTP protocol (allowing virtualization of web applications and web services), JMS, and JDBC to a certain extent. The agent provides APIs to enable virtualization directly from within server processes, thus making it protocol agnostic. Virtualization can be enabled for HTTP, JMS, and JDBC but also for RMI, EJB, or any custom Java objects.

DevTest (or any other client of the agent) can achieve this virtualization by using the following APIs defined in **com.itko.lisa.remote.client.VSEClient** as obtained by **AgentClient.getInstance().getVSEClient()**.

Note: You can view detailed information about this class in the JavaDocs for the agent. The JavaDocs are located in the **LISA_HOME\doc** directory.

```
/**
  * Returns a list of all class/interface names whose name matches the supplied
regular expression
  * or that extend/implement a class/interface whose name matches the supplied
regular expression
  * if implementing is true. Searching for annotations is supported through the
syntax:
  * class regex@annotation regex (e.g. ".*.Remote@.*.Stateless").
  * @param agentInfo
  * @param regex
  * @param impl
  * @return
  */
 public String[] getMatchingClasses(IAgentInfo info, String regex, boolean impl)
throws JMSInvocationException

 /**
  * Register a VSE callback with the specified agent whose onFrameRecord will be
invoked
  * in recording mode for all virtualized methods and whose onFramePlayback method
will be invoked
  * in playback mode for all virtualized methods.
  * @param info
  * @param callback
  */
 public void registerVSECallback(IAgentInfo info, IVSECallback callback);

 /**
  * Unegister a VSE callback with the specified agent.
```

```
     * @param info
     * @param callback
     */
    public void unregisterVSECallback(IAgentInfo info, IVSECallback callback);

    /**
     * Start calling our virtualization recording callback on the specified agent.
     * @param agentInfo
     * @throws RemoteException
     */
    public void startVSERecording(IAgentInfo agentInfo) throws
JMSInvocationException

    /**
     * Start calling our virtualization playback callback on the specified agent.
     * @param agentInfo
     * @throws RemoteException
     */
    public void startVSEPlayback(IAgentInfo agentInfo) throws
JMSInvocationException

    /**
     * Stop virtualizing on the specified agent.
     * @param agentInfo
     * @throws RemoteException
     */
    public void stopVSE(IAgentInfo agentInfo) throws JMSInvocationException

    /**
     * Virtualizes the specified class/interface and all its descendants on the
specified agent.
     * @param agentInfo
     * @param className
     * @return
     */
    public void virtualize(IAgentInfo agentInfo, String className) throws
JMSInvocationException
```

The interface for the callback APIs is defined by **com.itko.lisa.remote.vse.IVSECallback** and defines the following methods:

```
/**
  * This is the method that gets invoked by agents that have VSE recording turned
on
  * when a virtualize method gets called. The VSE frame has all the information
needed
  * to later replay the method in playback mode.
  * @param frame
  * @throws RemoteException
  */
```

```
 void onFrameRecord(VSEFrame frame) throws RemoteException;

 /**
  * This is the method that gets invoked by agents that have VSE playback turned
on
  * when a virtualize method gets called. The VSE frame has all the information
needed
  * to match an existing recorded frame so its result (and by reference arguments)
  * can be set appropriately.
  * @param frame
  * @return
  * @throws RemoteException
  */
 VSEFrame onFramePlayback(VSEFrame frame) throws RemoteException;
```

Finally, the **com.itko.lisa.remote.vse.VSEFrame** object is a POJO with getters and setters for the following properties:

```
/** Get/sets a unique identifier for this frame */
 public String getFrameId();
 public void setFrameId(String frameId);

 /** The agent id this frame originates from */
 public long getAgentGuid();
 public void setAgentGuid(long agentId);

 /** The thread name this frame method was invoked on */
 public String getThreadName();
 public void setThreadName(String threadName);

 /** The name of the class this frame method was invoked on */
 public String getClassName();
 public void setClassName(String className);

 /** A unique identifier that tracks objects for the span of the VM's life */
 public String getSourceId();
 public void setSourceId(String srcId);

 /** The session id of the innermost session-scoped protocol enclosing this frame
*/
 public String getSessionId();
 public void setSessionId(String sessionId);

 /** The name of the method that was invoked */
 public String getMethod();
 public void setMethod(String method);

 /** The XStream'ed arguments array to the method that was invoked */
 public String[] getArgumentsXML();
 public void setArgumentsXML(String[] argumentsXML);
 /** The XStream'ed result of the method that was invoked */
 public String getResultXML();
 public void setResultXML(String resultXML);

 /** The (server) time the method was invoked */
 public long getTime();
 public void setTime(long time);

 /** The time the method took to execute */
 public long getClockDuration();
 public void setClockDuration(long duration);

 /** Whether to use getCode or the ResultXML to compute the desired result in
playback mode */
 public boolean isUseCode();
 public void setUseCode(boolean useCode);

 /**
  * The code to execute on the server if isUseCode is true. This can be arbitrary
code
  * that has access to the object ($0) and method arguments ($1, $2,...)
  */
 public String getCode();
 public void setCode(String code);
```

Agent API Examples

Agent API Example 1

The following code generates a transaction from client code and retrieves the transaction tree that it generated:

```
private static void testAddTransaction() throws Exception
 {
 String request =
"http://localhost:8080/examples/servlets/servlet/HelloWorldExample";
 TransactionsClient tc = AgentClient.getInstance().getTransactionClient();

 tc.installSocketTracker(false);
 long start = System.currentTimeMillis();

 String response = testMakeRequest(request);

 long end = System.currentTimeMillis();
 tc.uninstallSocketTracker(false);

 String frameId = tc.addClientTransaction("test", new Object[] { request },
response, end - start);
 TransactionFrame frame = tc.getTransactionTree(frameId);

 System.out.println(frame.isComplete() ? "Yes!" : "No!");
 }
```

The preceding code uses the following utility function, which has nothing to do with the agent but is listed for completeness:

```
/** Assuming Tomcat is running on localhost:8080 */
 private static String testMakeRequest(String url) throws IOException
 {
 StringBuffer response = new StringBuffer();
 HttpURLConnection con = (HttpURLConnection) new URL(url).openConnection();

 con.setRequestMethod("GET");
 con.setDoOutput(true);
 con.setUseCaches(false);

 BufferedReader br = new BufferedReader(new
InputStreamReader(con.getInputStream()));
 for (String line = br.readLine(); line != null; line = br.readLine())
response.append(line);
 br.close();

 return response.toString();
 }
```

Agent API Example 2

The following code registers a logging VSE callback:

```
AgentClient.getInstance().addListener(new IAgentEventListener()
 {
  public void onAgentOffline(final IAgentInfo info) {}
  public void onAgentOnline(final IAgentInfo info)
  {
   AgentClient.getInstance().getVSEClient().registerVSECallback(info, new
IVSECallback()
   {
    public void onFrameRecord(VSEFrame frame) { System.out.println("Recorded: "
+ frame); }
    public VSEFrame onFramePlayback(VSEFrame frame) {
System.out.println("Played back: " + frame); return frame; }
    public int hashCode() { return 0; }
    public boolean equals(Object o) { return o instanceof IVSECallback; }
  });

  try { AgentClient.getInstance().getVSEClient().startVSERecording(info); }
catch (JMSInvocationException e) {}
 });
```

Java Agent Extensions

You can create the following types of extensions for the DevTest Java Agent:

- Agent extensions (see page 70)
- Extending the Broker (see page 76)
- Java VSE extensions (see page 78)

Extending the Agent

You can customize the agent behavior by writing Java classes that implement the **com.itko.lisa.remote.transactions.interceptors.IInterceptor** interface or extend the **com.itko.lisa.remote.transactions.interceptors.AbstractInterceptor** class. For more information, see the JavaDocs in the **doc** folder of your installation directory.

```
public interface IInterceptor {
    /**
     * Whether this custom interceptor is currently disabled
     */
    public boolean isDisabled();

    /**
     * Returns whether the interception should return (true) or proceed (false)
     */
    public boolean block(boolean wayIn, Object src, String spec, String clazz,
String method, String signature, Object[] args, Object ret);

    /**
     * Called after method entry to possibly modify the current frame based on
interceptor logic.
     */
    public boolean preProcess(TransactionFrame frame, Object src, String spec,
String clazz, String method, String signature, Object[] args, Object ret);

    /**
     * Called before method exit to possibly modify the current frame based on
interceptor logic
     */
    public boolean postProcess(TransactionFrame frame, Object src, String spec,
String clazz, String method, String signature, Object[] args, Object ret);
}
```

If you want to stop data from being captured, overwrite the **block()** method. This technique is similar to adding the **exclude** directive to the **rules.xml** file, but provides more flexibility.

If you want the agent to perform logic immediately before it captures a method, overwrite the **preProcess()** method.

If you want the agent to perform logic immediately after it captures a method, overwrite the **postProcess()** method.

These methods are automatically invoked for all classes and methods that have been intercepted or tracked. To intercept a method or track a class, you can specify it using the documented syntax in the **rules.xml** file. You can also programmatically specify the interception in the constructor of the extension class. The advantage of the latter approach is that the extension is self-contained.

The first argument to the **block()** method is **boolean wayIn**. The **block()** method is called twice per method: once on entry, once on exit. When the entry call is made, the value of the **wayIn** argument is true. When the exit call is made, the value of the **wayIn** argument is false.

The following table describes the arguments that are common to the **block()**, **preProcess()**, and **postProcess()** methods:

Argument	**Description**
Object src	The object that the method is being called on.
String spec	The class or interface name that was specified to instrument the API.
String clazz	The name of the class that defines the intercepted method.
String method	The name of the intercepted method.
String signature	The signature (in JVM format) of the intercepted method.
Object[] args	The arguments being passed to the intercepted method.
Object ret	The return value of the intercepted method. When the wayIn argument is true, the return value is null.

The difference between the **src**, **spec**, and **clazz** arguments can be explained with an example.

Assume that you have the following interface and class definitions:

```
public interface A {
     void m();
}

public class B implements A {
     void m() {}
}

public class C extends B {
}
```

If the rules specify **intercept("A", "m", "*")** and the code calls **C.m()**, then the following information is true:

- **spec** is A
- **clazz** is B
- **src** is an instance of C

Deployment

To deploy an extension, compile it and package it in a JAR file with a manifest containing the following entry:

```
Agent-Extension: extension class name
```

When you drop this JAR in the Agent JAR directory, the agent automatically picks it up. If you add the file after the agent has started, a hot load mechanism helps to ensure that the file is applied. If you update the extension JAR as the agent is running, the JAR classes are reloaded dynamically. Dynamic reloading makes it easy and fast to test your extension code without restarting the server.

The extension JARs get loaded by a classloader that can see all the classes that are used in the extension class. You can compile your extension source against **LisaAgent.jar** and all container JARs that define classes you want to use. You do not need to use reflection in your extension.

Examples

The following example uses the **block()** method to prevent the agent from capturing any method whose thread name starts with **Event Sink Thread Pool**.

```
public class MyInterceptor extends AbstractInterceptor {
   /** Returns true if this call should not be intercepted */
   public boolean block(boolean wayIn, Object src, String spec, String clazz,
String method, String signature, Object[] args, Object ret) {
      if (Thread.currentThread().getName().startsWith("Event Sink Thread
Pool")) {
         return true;
      }
      return super.block(wayIn, src, spec, clazz, method, signature, args, ret);
   }

   ...

}
```

The following example uses the **postProcess()** method to capture data for the Response row in the Transactions window. This example calls the **setResponse()** method of the **com.itko.lisa.remote.transactions.TransactionFrame** class. For more information, see the JavaDocs in the **doc** folder of your installation directory.

```
public class MyInterceptor extends AbstractInterceptor {

   ...

   public boolean postProcess(TransactionFrame frame, Object src, String spec,
String clazz, String method, String signature, Object[] args, Object ret) {
      if (clazz.equals("com.itko.lisa.training.NotCaptured") &&
method.equals("doRequest")) {

         frame.setResponse((String) ret);
      }
      return super.postProcess(frame, src, spec, clazz, method, signature, args,
ret);
   }
```

The following example shows how to print the XML contents of a WebMethods **com.wm.data.IData** object as it gets invoked in a flow of Integration Server. The agent already supports WebMethods, so an extension is not necessary for it.

```
package com.itko.lisa.ext;

 import java.io.ByteArrayOutputStream;
 import com.itko.lisa.remote.transactions.interceptors.AbstractInterceptor;
 import com.itko.lisa.remote.transactions.TransactionFrame;
 import com.wm.app.b2b.server.ServiceManager;
 import com.wm.app.b2b.server.BaseService;
 import com.wm.data.IData;
 import com.wm.util.coder.IDataXMLCoder;

 public class IDataInterceptor extends AbstractInterceptor {
```

```
 public IDataInterceptor() {
     super.intercept("com.wm.app.b2b.server.ServiceManager", "invoke",
"(Lcom/wm/app/b2b/server/BaseService;Lcom/wm/data/IData;Z)Lcom/wm/data/IData;
");
 }

 public boolean preProcess(TransactionFrame frame, Object src, String spec,
String clazz, String method, String signature, Object[] args, Object ret) {
     if (ServiceManager.class.getName().equals(clazz) &&
"invoke".equals(method)) {
         doCustomLogic((BaseService) args\[0\], (IData) args\[1\], false);
     }

     return super.preProcess(frame, src, spec, clazz, method, signature, args,
ret);
 }

 public boolean postProcess(TransactionFrame frame, Object src, String spec,
String clazz, String method, String signature, Object[] args, Object ret) {
     if (ServiceManager.class.getName().equals(clazz) &&
"invoke".equals(method)) {
         doCustomLogic((BaseService) args\[0\], (IData) ret, true);
     }

     return super.postProcess(frame, src, spec, clazz, method, signature, args,
ret);
 }

 private void doCustomLogic(BaseService flow, IData p, boolean output) {
     ByteArrayOutputStream baos = new ByteArrayOutputStream(63);

     try {
         new IDataXMLCoder().encode(baos, p);
     } catch (IOException e) {
         e.printStackTrace();
     }

     System.out.println("Flow: " + flow.getNSName());
     System.out.println((output ? "Output" : " Input") + "Pipeline: " + baos);

 }
 }
```

To build this extension yourself, compile this code against **LisaAgent.jar**, **wm-isclient.jar**, and **wm-isserver.jar** then JAR the class file with a manifest containing the following line:

```
Agent-Extension: com.itko.lisa.ext.IDataInterceptor
```

Adding Tags to Transaction Frames

You can associate a transaction frame with arbitrary key/value pairs. The key/value pairs are known as *tags*.

TransactionFrame objects provide a **setTag()** method. The method has two string arguments: the key and the value.

The following code shows how to add a tag from within the **postProcess()** method of an agent extension:

```
// somewhere in postProcess
frame.setTag("tagname", "tagvalue");
```

For example, you can create an extension that upon login to a web site grabs the value of the request parameter **username** and calls **frame.setTag("username", value)**.

The tags are stored in the **FRAME_TAGS** table in the database.

You can filter by tags in the CAI Console. For more information, see *Using CA Continuous Application Insight*.

Extending the Broker

Broker extensions are loaded and invoked by the broker after a transaction fragment is received from an agent.

Broker extensions let you perform the following tasks:

- Change data that is contained in frames.
- Add or eliminate certain frames.
- Customize the stitching algorithm. The stitching algorithm defines how transaction fragments are assembled.

To extend the broker, implement the **com.itko.lisa.remote.plumbing.IAssemblyExtension** interface. This interface defines the **onTransactionReceived()** method. For more information, see the JavaDocs in the **doc** folder of your installation directory.

```
public interface IAssemblyExtension {
     /**
      * The method invoked prior to assembly
      * @param frame the partial transaction root received from an agent
      * @return true to bypass normal assembly (i.e. if the extension wants to
take care of it itself)
      */
     public boolean onTransactionReceived(TransactionFrame frame);
 }
```

To deploy a broker extension, compile the extension and package it in a JAR file with a manifest that contains the following entry:

```
Broker-Extension: extension class name
```

When you drop this JAR in the broker (registry) directory, the broker automatically picks it up. If you add the file after the broker (registry) has started, a hot load mechanism helps to ensure that the file is applied. If you update the extension JAR as the broker (registry) is running, the JAR classes are reloaded dynamically. Dynamic reloading makes it easy and fast to test your extension code without restarting the broker (registry).

Example

A typical usage example is when the normal stitching algorithm that uses TCP/IPs and ports is confused by a load balancer sitting between agents and is assigned a virtual IP, or when agents use a native library to do IO and we do not have direct access to the IPs and ports in use. In that case, the address and port fields of frames are either left blank or incorrect and we must assemble the frames in an extension:

```
import com.itko.lisa.remote.plumbing.IAssemblyExtension;
import com.itko.lisa.remote.transactions.TransactionFrame;
import com.itko.lisa.remote.utils.Log;
import com.itko.lisa.remote.utils.UUID;

public class LoadBalancerExtension implements IAssemblyExtension {

    private TransactionFrame m_lastAgent1Frame;
    private TransactionFrame m_lastAgent2Frame;

    public boolean onTransactionReceived(TransactionFrame frame) {

    if (frame.getClassName().equals("Class1") &&
frame.getMethod().equals("method1")) {
        m_lastAgent2Frame = frame;
        if (m_lastAgent1Frame.getFrameId() == m_lastAgent2Frame.getParentId())
{
            stitch(m_lastAgent1Frame, m_lastAgent2Frame);
        }
    }

    if (frame.getClassName().equals("Class2") &&
frame.getMethod().equals("method2")) {
        m_lastAgent1Frame = frame;
        if (m_lastAgent1Frame.getFrameId() == m_lastAgent2Frame.getParentId())
{
            stitch(m_lastAgent1Frame, m_lastAgent2Frame);
        }
    }

        return false;
    }

    private void stitch(TransactionFrame parent, TransactionFrame child) {
    String newFrameId = UUID.newUUID();
    parent.setFrameId(newFrameId);
    child.setParent(parent);
    parent.getChildren().add(child);
}
}
```

Extending Java VSE

In a Java VSE extension, you can overwrite any of the following methods:

- **onPreRecord():** This method is called during recording before the virtualized method starts executing.
- **onPostRecord():** This method is called during recording after the virtualized method stops executing.
- **onPreHijack():** This method is called during playback before the virtualized method goes to VSE.
- **onPostHijack():** This method is called during playback after the virtualized method returns from VSE.
- **onNewStream():** This method is called every time that an object is converted to XML.

Example

The following example uses the **onPostRecord()** method to change the name of a class during recording. This example calls the **setClassName()** method of the **com.itko.lisa.remote.vse.VSEFrame** class. For more information, see the JavaDocs in the **doc** folder of your installation directory.

```
public class MyInterceptor extends AbstractVSEInterceptor {

   ...

   public boolean onPostRecord(VSEFrame frame, Object src, String clazz, String
method, String signature, Object[] args, Object ret) {
      frame.setClassName(clazz.replaceAll("\\.", "_"));
      return super.onPostRecord(frame, src, clazz, method, signature, args,
ret);
   }

}
```

Load Balancers and Native Web Servers

The process of assembling partial transactions into complete transactions is referred to as *stitching*.

When agents are deployed on a network that has load balancers or native web servers, the stitching algorithm that CAI uses might not function properly. In this scenario, add the **loadbalancer** directive to the **rules.xml** file of the broker. For each appliance installed between agents, specify the IP address of the appliance. For example:

```
<loadbalancer ip="172.16.0.0"/>
<loadbalancer ip="172.31.255.255"/>
```

The **loadbalancer** directive must be placed within the **broker** element.

An alternate approach is to enable the **Always wait** property.

You can configure this property from the Agents window of the DevTest Portal. The property appears in the Settings tab.

Java Agent Security

The DevTest Java Agent provides a security mechanism for extensions and remote code invocation.

The agent installs its own security manager. All custom agent code runs under special permissions that the security manager enforces. If the application already uses its own security manager, then the agent security manager wraps the existing security manager and delegates to it for regular application code.

Extensions

Agent extensions for CA Continuous Application Insight and CA Service Virtualization run with the following restrictions:

- File access is limited to the agent directory and the temp directory.
- Process creation is disabled.
- Process exit is disabled.

As a result, the system under test is sandboxed from the rest of the computer.

Some applications can explicitly check for the security manager being null and take a different code path depending on the result. In this situation, the agent security manager can cause insurmountable issues. You can disable the agent security manager by using either of the following approaches:

- Specify **–Dsecurity.manager.disabled=true** on the command line.
- Ensure that the **Disable security manager** property is enabled.

You can configure this property from the Agents window of the DevTest Portal. The property appears in the Settings tab.

Note: Disabling the security manager disables authorization checks. However, you can still enable token authentication.

Remote Code Invocation

All agent APIs work through remote code invocation, because the agent is located in a remote system.

Security for remote code invocation is handled with tokens.

You use the **token** option to define one or two tokens for an agent. The first token represents an admin role. The second token represents a user role. If you specify two tokens, use a colon to separate them.

Examples:

- **token=asdf1** specifies an admin token with a value of **asdf1**.
- **token=asdf1:asdf2** specifies an admin token with a value of **asdf1**, and a user token with a value of **asdf2**.

The admin or user token can include any character except for commas, equal signs, and space characters. The maximum length of a token is 16 characters.

The token concept can be used with consoles. You can specify it with the command-line syntax **-Dlisa.token=xxxx** or **-Dlisa.token=xxxx:xxxx**. For consoles, there is no custom security manager, so both tokens have all permissions. Security is achieved by the fact that not supplying the console token prevents you from doing anything.

Configure the Java Agent to Use SSL

You can use the Secure Sockets Layer (SSL) to secure communication between the agent and the broker.

The broker inherits the settings that the registry has for SSL. The agent uses the same encrypted password and the same keystore.

Choose one of the following approaches.

Default Keystore

This approach uses the default keystore in DevTest.

Follow these steps:

1. Open the **local.properties** file in the **LISA_HOME** directory and uncomment the following line:

   ```
   lisa.net.default.protocol=ssl
   ```

2. Save the **local.properties** file.
3. Start or restart the registry.
4. When specifying the broker URL, use the **ssl** scheme. For example:

   ```
   ssl://localhost:2009
   ```

Custom Keystore

This approach is based on a custom keystore.

Note: For detailed information about the DevTest Portal, see Using CA Continuous Application Insight.

Follow these steps:

1. Configure a custom keystore by following the instructions in Using SSL to Secure Communication.
2. Place the keystore in the **LISA_HOME\agent** directory.
3. Open the DevTest Portal.
4. Select Settings, Agents in the left navigation pane.
5. In the left portion, select the agent.
6. Click the Settings tab.

7. Set the following security properties to configure a custom keystore that mirrors the DevTest properties. You can copy the encrypted passwords from the **local.properties** file.
   - Keystore location
   - Keystore password (encrypted)
   - Truststore location
   - Truststore password (encrypted)
8. In the left portion, select the broker.
9. Set the same properties as for the agent. The path to the keystore will be different.
10. When you specify the broker URL, use the **ssl** scheme. For example:

    ```
    ssl://localhost:2009
    ```

Java Agent Log Files

You can use the following log files to help troubleshoot (see page 85) issues:

- The agent log file is named **devtest_agent_pid.log**. This file is written to the same directory as the **LisaAgent.jar** file.
- Each time the broker is started, a log file named **devtest_broker_pid.log** is created. In addition, the broker has a consolidated log file named **pfbroker.log**. These files are written to the same directory as the main DevTest log files.
- The console log files are named **devtest_agent_console_pid.log**. DevTest Workstation has a log file, DevTest Portal has a log file, each simulator has a log file, and VSE has a log file. These files are written to the same directory as the main DevTest log files.

The **pid** portion of the file names is the process identifier of the process that is doing the logging.

Note: For information about the main DevTest log files and their location, see *Administering.*

The following configuration properties let you control the logging behavior:

Java logging level

Sets the log level at startup.

Maximum log size

Sets the maximum size of an agent log before it gets rolled over.

Maximum log archives

Sets the maximum number of rolled-over archived logs to keep.

Agent log

Overrides the default location.

You can configure these properties from the Agents window of the DevTest Portal. The properties appear in the Settings tab.

If you need to configure these properties from the **rules.xml** file, the corresponding property names are as follows:

- **lisa.agent.java.logging.level**
- **lisa.agent.log.max.size**
- **lisa.agent.log.max.archives**
- **lisa.agent.agent.log**

Java Agent Troubleshooting

Important: Investigate the target environment ahead of time and ensure that it has been tested, or test it yourself. Most Java agent issues are OS or JVM-dependent, instead of application-dependent. Be sure to follow the instructions in this documentation. If that does not help, this topic reviews the most common issues.

Most serious issues involving the DevTest Java Agent occur at start-up (for example, the agent is not found, or the process crashes or hangs). Generally, after you get past issues at start-up, you are in good shape. From there, it is a matter of tweaking the configuration or writing extensions.

Notes:

- Bouncing the servers to try various things in the agent environment can be difficult. Typically, it is simpler, and a worthwhile exercise, to test running **java <agent options> -version** on the target computer and the JVM. This exercise indicates whether the issue is OS/JVM-specific or container/application-specific.
- For a command-line utility that can help you with the installation process, see Using the Agent Install Assistant (see page 18).

Error at startup: Error occurred during initialization of VM. Could not find agent library in absolute path...

Verify that the library is indeed in that path and is using the same architecture as Java (both 32 bit or 64 bit).

Also verify that the library is not missing any dependencies. For example, use **depends.exe** on Win32, **otool -L** on Mac OS X, or **ldd -d** on Linux and UNIX. If libraries are missing, ensure **LD_LIBRARY_PATH** is set to include the directories where they reside. Containers can override **LD_LIBRARY_PATH** in their start-up scripts. Do not assume that it is correctly set if you set it in a shell instead of from inside the script or a container-specific administration tool.

If **ldd** does not return cleanly, the agent does not run properly. Therefore, verifying **ldd** is the first thing to get right. If you cannot find an agent version for the operating system, consider using the pure Java agent.

The agent exits immediately (or shortly after starting the process).

If you see the message **LISA AGENT: VM terminated**, it is likely that the process ended normally. Several containers have launcher processes, and it is normal for them to exit quickly.

If you do not see this message or a crash dump happens (after **ldd** returns cleanly), you could have a legitimate agent bug. Notify Support and try the pure Java agent. If it still crashes or produces a dump, you could have a JVM bug. One such occurrence is for the IBM JVM 1.5 on some operating systems, caused by trying to instrument threads. In that case, try supplying the command-line JVM argument **-Dlisa.debug=true**.

The agent exits with the message "GetEnv on jvmdi returned -3 (JNI_EVERSION)".

Try specifying the command-line option: **-Xsov** to instruct the JVM to use its debug-enabled libraries. If that does not work (for example, you get an invalid option error message), then this operating system is not supported.

The agent exits with the message "UTF ERROR" ["../../../src/solaris/instrument/EncodingSupport_md.c":66]: ..."

The message can vary depending on the exact version of the operating system, the JVM, or both. The message typically has one of the following elements: UTF ERROR ["../../../src/solaris/instrument/EncodingSupport_md.c":66]: Failed to complete iconv_open() setup.

This issue is due to a bug in some Solaris JVMs when some language packs are not installed. To fix this issue, install the en-US language pack by running **pkg install SUNWlang-enUS** and then **export LANG=en_US.UTF-8**. Alternatively, you can try using the native agent.

Agent hangs or throws numerous exceptions at startup (LinkageErrors, CircularityErrors, and so on).

A side effect of instrumenting Java bytecode using Java is that it can subtly change some of the class-loading order for early classes (**java.*** and such), resulting in a deadlock or bytecode verification errors.

We have eliminated all known occurrences of these issues for all combinations of JVMs and operating systems. However, it is possible that you have encountered an untested combination. Notify Support of this issue. If this issue is a hang, include a thread dump with your support issue. Produce a thread dump by entering CTRL+Break on Windows, CTRL+\ or kill -3 <pid> on UNIX/Linux. You can also try the Java agent, as the class-loading order is slightly different. If a class or package seems involved every time in the hung thread, try adding an **exclude** directive for it in the **rules.xml** file.

Agent throws java.lang.VerifyErrors or hot swapping has no effect.

Some older 1.4 JVMs have bugs in their support for hot swapping (instrumenting classes after they are loaded).

In that case, turn off hot swapping by disabling the **Enable hot instrumentation** property.

You can configure this property from the Agents window of the DevTest Portal. The property appears in the Settings tab.

You must determine in advance the classes, methods, or both that you want to intercept or virtualize, add the rules for these classes or methods in the **rules.xml** file, and bounce the server. This process is more tedious than doing it on a live server, but it is the only known way to work around this issue.

Sometimes the **java.lang.VerifyError** is not even seen in the logs, but the agent behaves as though the designated class has not been instrumented or yields random results, including crashes.

Agent starts but the consoles or broker cannot see the agent.

Typically, the cause is a firewall or port issue between the agent and the broker.

The top of the agent log can include the following warning: **Can't connect to broker at tcp://ip:port**. Verify that the IP address and port that the agent is using are correct. Then ensure that the broker is listening on the specified port on the specified IP address. **netstat -ano | grep port** should show a port listening on the supplied ip or 0.0.0.0. Finally, look for firewall issues by running **telnet ip port** from the agent computer to ensure that it can see the broker. If it can see the broker, the broker is likely in a bad state. Review the registry and broker logs (and restart it if necessary).

Agent causes some operations to time out.

Some JVMs (IBM JVMs in particular) do not behave properly after some of their networking classes are instrumented. As a result, networking calls can fail or time out without apparent reason.

To prevent those classes from being instrumented, try supplying the command-line JVM argument **-Dlisa.debug=true**.

If the issue is not resolved, edit the **rules.xml** file to exclude the following networking packages:

```
<exclude class="java.net.**"/>
<exclude class="java.nio.**"/>
<exclude class="sun.nio.**"/>
```

java.lang.NoClassDefFoundError on com/itko/lisa/remote/transactions/TransactionDispatcher.class

Verify that **LisaAgent.jar** is available and has read permissions and that it is not corrupted. The easiest way to verify is to run **java -jar LisaAgent.jar -v**.

Verify whether the application uses OSGi. If it does (as in JBoss 7), add **com.itko** to the system or bootstrap packages. The method for adding **com.itko** is container-dependent, which makes it difficult to provide specific instructions. However, it is typically a configuration file with a property that specifies a list of packages or a similar JVM argument.

Security-related exceptions are thrown when the agent is enabled.

You may see SecurityExceptions or PermissionExceptions thrown by the application only when the agent is turned on with the security enabled. This setting is now the default setting.

The reason and workaround are explained in Java Agent Security (see page 80).

Abnormal resource consumption (CPU, memory, file handles, ...).

If the CPU usage is abnormally high or spikes periodically, note the period of the spikes because it will help determine the faulty thread or threads. Also try turning off CAI and VSE.

If you get OutOfMemory errors, monitor the Java heap usage. If it exceeds the **-Xmx** limit, then increase that limit. However, avoid increasing the limit if it is already high and well in excess of normal application usage without the agent. In that case, generate a heap dump. You can use the WAS HeapDump utility for WebSphere or the free Eclipse MAT tool for older versions. If the memory usage is below the **-Xmx** limit at the time of the error, it is likely a leak in native code. In this case, notify Support.

If you get unexplained IOExceptions (such as too many file handles), especially on UNIX or Linux, verify the ulimit on the box (**ulimit -n -H** and **ulimit -n -S**). If it is low, consider asking the administrator of the box to raise it (under 4096 is considered low for modern J2EE apps). Do not forget to restart afterward.

The agent tries to keep the number of file handles under that limit by triggering garbage collection when the limit is approached. If the agent cannot read the limit at startup, the agent uses a default value of 1024. Using this default value can result in excessive garbage collection and semi-random, frequent CPU spikes. The following messages in the logs indicate this situation:

```
    Max (or preferred) handles limit approaching - triggering GC...
```

In that case, increase the value of the **Maximum number of handles** property.

You can configure this property from the Agents window of the DevTest Portal. The property appears in the Settings tab.

I can see the agent started, but CAI data is missing or incomplete.

The data lifecycle process includes the following stages:

- Transaction capture in the agent
- Transfer to the broker
- Partial transaction assembly in the broker
- Transfer to the consoles
- Persistence to the database
- Retrieval from the database

Missing or incomplete data can be the result of an issue in any of these stages.

Review the agent log for capture exceptions. Review the broker and console logs for transfer or persistence exceptions.

If there are no exceptions, and you are still unable to locate the missing data, turn on debug or dev logging in the agents. If you do not see statements such as "Sent partial transaction," then CAI is probably not turned on.

If none of these steps provide conclusive results, contact Support.

I turned on Java VSE recording or playback, but VSE does not receive any requests from the agent.

First, verify that the agent is in VSE record or playback mode. Open the agent log and search for "Starting VSE record/playback..."

Then look for exceptions in the agent logs and the VSE logs. If they are clean, it is possible that the class you think is virtualized is not virtualized. Look in the agent log for a statement such as "Virtualized com.xxx...." If the statement is missing, it is possible that the application has not yet loaded the class. Another possibility is that if you are using an early version of Java 1.4, some flavors do not support hot swapping. Java VSE uses hot swapping by default. In that case, specify the virtualized classes in the **rules.xml** file of the agent and restart it.

Other issues with Java VSE.

If you encounter any issues (functional or abnormal resource usage) while doing Java VSE, turn off CAI on the agent side.

You can turn off CAI by disabling the **Auto-start** property. Then restart the JVM.

You can configure this property from the Agents window of the DevTest Portal. The property appears in the Settings tab.

Do not turn off CAI on the broker side or Java VSE stops working altogether.

Case functionality is not working.

First, ensure that the agent is connected to a broker.

If the agent is connected to a broker, review the HTML source of the page that has the issue. Verify that the bottom of the page has a block of JavaScript that is easily identifiable by the variable names it uses (for example, **com_itko_pathfinder_defectcapture_xxx**). If the block is missing, review the agent log for possible exceptions. Other reasons for the absence of the block include:

- The page HTML is unusual. For example, HTML tags are missing.
- The agent has issues capturing it entirely, which can happen with untested containers or static pages.

If the block is present, verify whether a native web server (such as Apache) or a load balancer is present in front of the Java container. If it is the case, configure it to forward the request of the CAI JavaScript and resource files to the Java container. Those will have the word **defectcapture** as part of their URL. IT administrators generally know how to perform this task.

DevTest is configured to use an IBM DB2 database. In the DevTest Portal, I selected a JDBC node in a path graph. The Statements and Connection Url columns are blank.

Disable progressive streaming by adding the string **progressiveStreaming=2** to the JDBC connection URL. For example:

```
lisadb.pool.common.url=jdbc:db2://myhostname:50000/dbname:progressiveStreaming=2;
```

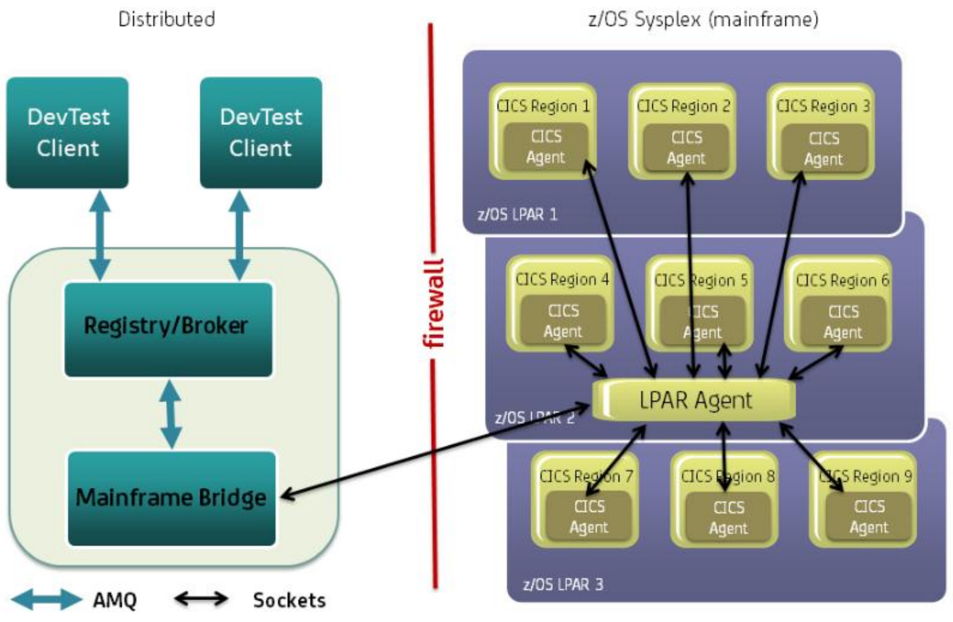
Chapter 2: Mainframe Bridge

The mainframe bridge is a component that enables DevTest clients to communicate with DevTest mainframe agents.

This section contains the following topics:

Mainframe Bridge Architecture

The following graphic shows how the mainframe bridge interacts with other components.



After a connection is made, traffic goes from DevTest clients to the broker to the mainframe bridge to the actual mainframe. Traffic can also go in the reverse order.

Mainframe Bridge Rules

The following agent properties must be configured for the mainframe bridge:

Code page

Defines the codepage that is used to convert between ASCII and EBCDIC.

Default: Cp1047

Initial packet buffer size

Specifies the size of the initial buffer to handle packets from the mainframe, in bytes. The buffer size increases if packets of a larger size are received.

Default: 1024*65.

Packet trace LPAR

Specifies whether packets from and to the LPAR Agent are traced.

Default: Tracing is disabled.

Packet trace client

Specifies whether packets from and to the workstations are traced.

Default: Tracing is disabled.

Client connect timeout

Specifies the number of seconds to wait between retry connects if the bridge is started in client mode and the connection to the LPAR Agent fails.

Default: 30

You can configure these properties from the Agents window of the DevTest Portal. The properties appear in the Settings tab.

Mainframe Bridge Properties

The following properties must be added to the **local.properties** file in the **LISA_HOME** directory.

lisa.mainframe.bridge.enabled

Specifies whether the mainframe bridge is started when the registry is started. The bridge runs inside the registry and not as an external process.

Default: false

lisa.mainframe.bridge.mode

Specifies the mode in which the mainframe bridge is run. The mode defines the peer relationship with the LPAR agents. The bridge can be run in client mode or server mode:

- In client mode, the LPAR Agent must be configured to run in server mode. The bridge initiates the connection to the LPAR Agent.
- In server mode, the LPAR Agent must be configured to run in client mode. The bridge waits for connections from the LPAR Agent.

Values:

- client
- server

Default: server

lisa.mainframe.bridge.port

In server mode, this property specifies the “well-known port” that the bridge listens on for connections from the LPAR Connection. This property is ignored in client mode.

Values: Any valid port number.

Default: 61617

lisa.mainframe.bridge.server.host

In client mode, this property defines the host to which the bridge initiates a connection. This property is ignored in server mode.

Values: Any valid port number.

Default: localhost

lisa.mainframe.bridge.server.port

In client mode, this property defines the port to which the bridge initiates a connection on the specified host. This property is ignored in server mode.

Values: Any valid port number.

Default: 3997

lisa.mainframe.bridge.connid

This property is reserved for future use.

DevTest z/OS Agent Console

The Mainframe Bridge provides a console, available through the Server Console, that provides limited visibility into, and management of, the DevTest CICS Agent. This facility allows a user or administrator without z/OS access to perform limited management of the CICS Agent. A z/OS or CICS systems programmer must perform comprehensive management of the CICS Agent.

The z/OS Agent Console provides the following capabilities for limited visibility into, and management of, the CICS Agent:

- Ensure that a DevTest CICS Agent is active
- View some attributes of the CICS Agent
- View some attributes of the CICS region
- List CICS Agent virtualization recording and playback activities
- Cancel/reset CICS Agent virtualization recording and playback activities

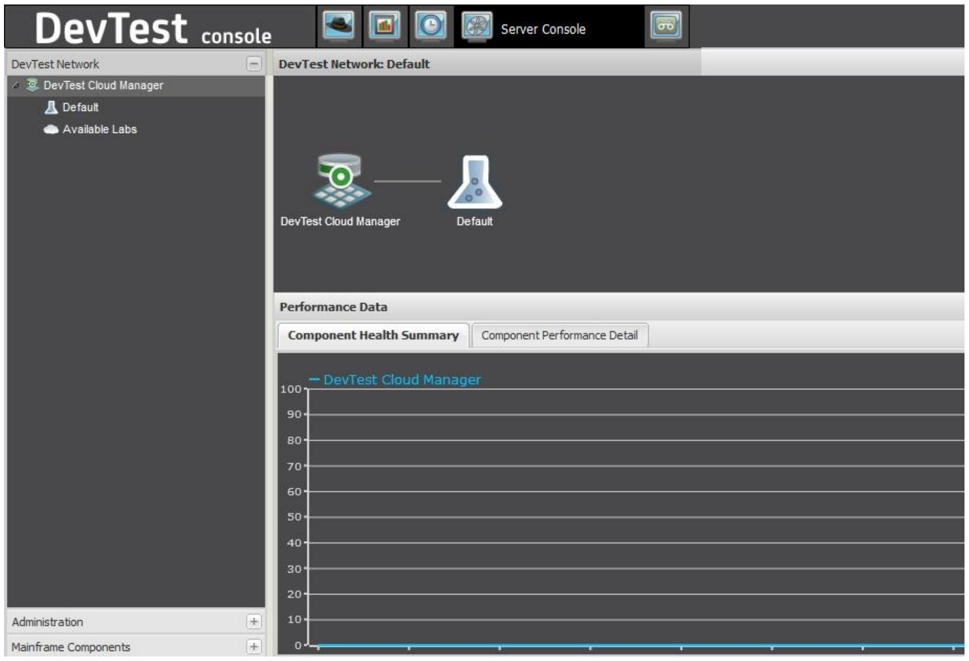
The z/OS Agent Console is not intended to be a substitute for a z/OS or CICS systems programmer performing agent operations.

The console does not allow:

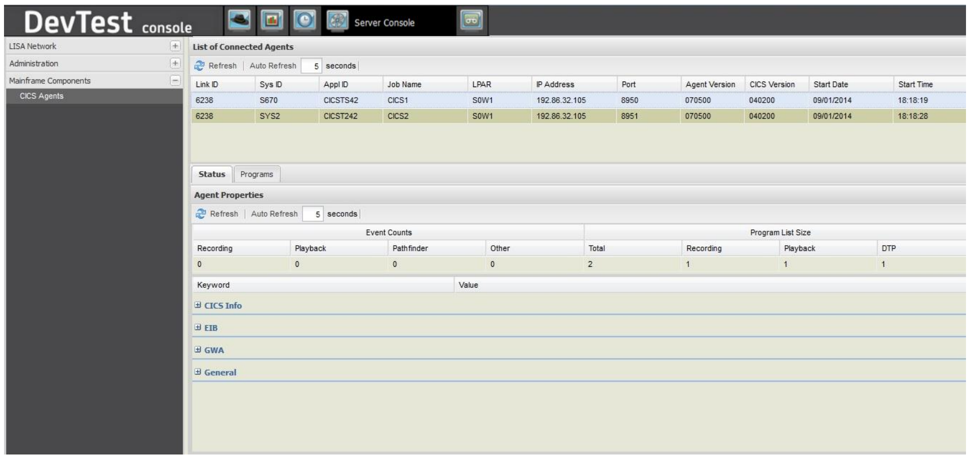
- Any DevTest z/OS agents to be started, stopped, or restarted
- Any DevTest z/OS agent configurations to be changed

Launching the DevTest Console

In the DevTest Console, a Mainframe Components tree exists if the Mainframe Bridge is enabled.



To expand the Mainframe Components tree, click the +. The connected z/OS CICS agents are displayed.



List of Connected Agents

Link ID	Sys ID	Appl ID	Job Name	LPAR	IP Address	Port	Agent Version	CICS Version	Start Date	Start Time
6238	S670	CICSTS42	CICS1	S0W1	192.86.32.105	8950	070500	040200	09/01/2014	18:18:19
6238	SYS2	CICST242	CICS2	S0W1	192.86.32.105	8951	070500	040200	09/01/2014	18:18:28

Agent Properties

Event Counts				Program List Size			
Recording	Playback	Pathfinder	Other	Total	Recording	Playback	DTP
0	0	0	0	2	1	1	1

Keyword	Value
⊞ CICS Info	
⊞ EIB	
⊞ GWA	
⊞ General	

Select an agent in the upper pane and view or manage the agent in the lower pane.

The lower pane has two tabs; Status and Programs.

Below the tabs are some agent transaction counts. A "transaction" in this case is a single CICS event, such as a CICS LINK, or DTP ALLOCATE, not an entire CICS transaction.

Event Counts

Recording

Displays the number of recorded transactions since the agent was initialized.

Playback

Displays the number of virtualized transactions since the agent was initialized.

Pathfinder

Displays the number of transactions reported to CAI since the agent was initialized.

Other

Displays the number of miscellaneous transactions since the agent was initialized.

Program List Size

Total

Displays the total number of entries in the program list. The program list contains an entry for each transaction that is recorded or virtualized.

Recording

Displays the number of transactions in the program list that are in recording mode.

Playback

Displays the number of transactions in the program list that are in playback (virtualize) mode.

DTP

Displays the number of transactions in the program list (both recording and playback) that are Distributed Transaction Processing transactions.

Status Tab

The Status tab shows agent attributes.

The CICS Info tree has some CICS configuration information from the CICS region the agent is running in.

The EIB tree has the CICS Agent (CICS transaction) Execute Interface Block.

The GWA tree has the CICS Agent Global Work Area. This area is used to communicate with the various CICS exits used by the CICS Agent.

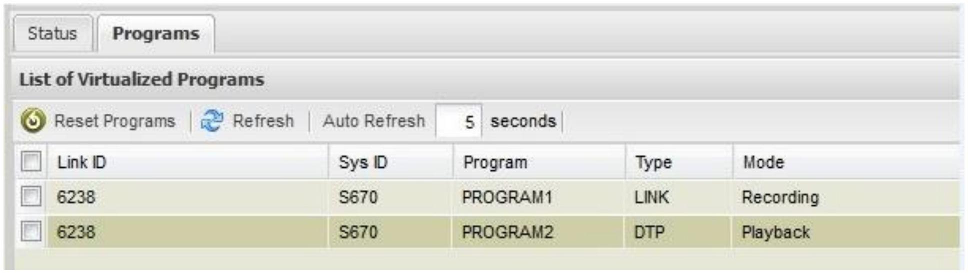
The General tree has the agent version, transaction ID, and so forth.

Program Tab

The Program tab shows all transactions (CICS LINK, DTP, and so forth) that are deployed to the CICS Agent.

In the example, there are two programs:

1. PROGRAM1 is in recording mode for CICS LINKs.
2. PROGRAM2 is in playback mode for Distributed Transaction Processing commands.



Link ID	Sys ID	Program	Type	Mode
6238	S670	PROGRAM1	LINK	Recording
6238	S670	PROGRAM2	DTP	Playback

If one of these programs has been "orphaned" in the agent, it can be removed by clearing the check box and clicking the Reset Programs button. This action removes it from the CICS Agent.

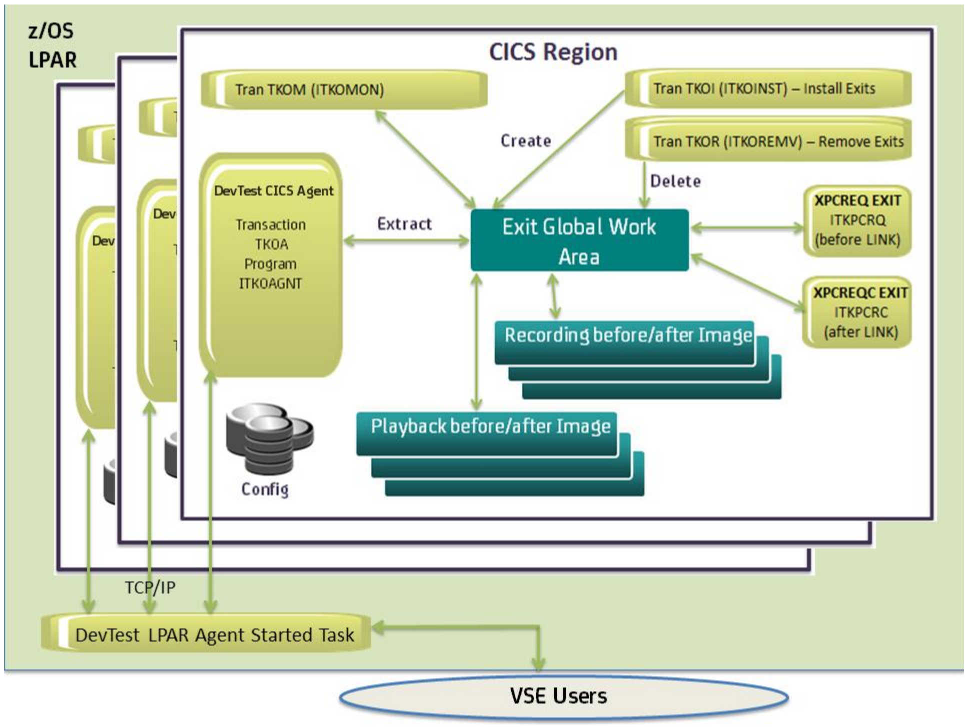
Chapter 3: DevTest CICS Agent

The DevTest CICS Agent is a z/OS CICS resident agent that provides functions to support the VSE. The agent is implemented as a combination of long-running task TKOA and CICS exits XPCREQ, XPCREQC, XEIIN, and XEIOUT. The agent can be activated automatically through the CICS PLT, or manually through the TKOI transaction to install and TKOR transaction to remove. The only configuration that is required is the location of the DevTest LPAR Agent.

This section contains the following topics:

CICS Agent Overview

The following graphic shows the components of the DevTest LPAR Agent and DevTest CICS Agent.



How to Install the CICS Agent

The DevTest CICS Agent package consists of the following files:

- iTKOCICSAgentDoc.pdf
- iTKOCICSAgentLoadCICS32
- iTKOCICSAgentLoadCICS41
- iTKOCICSAgentLoadCICS42
- iTKOCICSAgentLoadCICS51
- iTKOCICSAgentLoadCICS52
- iTKOCICSAgentCntl

The steps for the installation are:

1. FTP the files to the target z/OS system (see page 148).
2. Restore the Partitioned Datasets (PDSs) (see page 103).
3. Ensure that the TCP/IP interface is installed and started in CICS (see page 103).
4. Create the CICS Agent configuration file (see page 103).
5. Define CICS Agent resources to CICS (see page 104).
6. Add exit enable/disable to the CICS PLT (see page 105).
7. Modify CICS startup JCL (see page 106).

FTP the Files to the Target z/OS System

Use the load library that matches your CICS version:

- For CICS Version 3.2, use iTKOCICSAgentLoadCICS32
- For CICS Version 4.1, use iTKOCICSAgentLoadCICS41
- For CICS Version 4.2, use iTKOCICSAgentLoadCICS42
- For CICS Version 5.1, use iTKOCICSAgentLoadCICS51
- For CICS Version 5.2, use iTKOCICSAgentLoadCICS52

A simple method to determine your version of CICS is to run the following transaction and check the value of CICSTslevel. CICSTslevel has the version/release/maintenance level in the format vvrrmm: CECI INQUIRE SYSTEM CICSTslevel.

The files iTKOCICSAgentLoadCICSvvv and iTKOCICSAgentCntl are FTP'd to the target z/OS system in binary mode with the z/OS attributes indicated in the example that follows.

The following example of command-line FTP shows the proper transfer to z/OS of distribution files data sets **iTKOCICSAgentCntl** and **iTKOCICSAgentLoadCICS42** to z/OS data sets ITKO.CICSAGNT.CNTL.UNLOAD and ITKO.CICSAGNT.LOAD.UNLOAD. You can change the /OS file names for your data set naming conventions.

```
ftp> quote SITE LRECL=80 RECFM=FB BLKSIZE=3120 TRACKS PRI=10 SEC=1
200 SITE command was accepted
ftp> bin
200 Representation type is Image
ftp> put iTKOCICSAgentLoadCICS42 'ITKO.CICSAGNT.LOAD.UNLOAD'
200 Port request OK.
125 Storing data set ITKO.CICSAGNT.LOAD.UNLOAD
250 Transfer completed successfully.
ftp> put iTKOCICSAgentCntl 'ITKO.CICSAGNT.CNTL.UNLOAD'
200 Port request OK.
125 Storing data set ITKO.CICSAGNT.CNTL.UNLOAD
250 Transfer completed successfully.
```

Restore the Partitioned Data sets (PDSs)

The CICS Agent load library PDS and control (JCL) library are restored with the TSO RECEIVE command, which typically runs in a batch job.

The following sample JCL restores the PDSs from the FTP'd files data sets **ITKO.CICSAGNT.LOAD** and **ITKO.CICSAGNT.CNTL** on volume VOL001. The data set names can be changed for your site data set naming conventions.

```
//job
//*
//* Unload the LOAD Library with TSO RECEIVE
//*
//UNLOAD EXEC PGM=IKJEFT01,DYNAMNBR=10
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSTSIN DD *
RECEIVE INDSNAME('ITKO.CICSAGNT.LOAD.UNLOAD')
   DATASET('ITKO.CICSAGNT.LOAD') VOLUME(VOL001)
RECEIVE INDSNAME('ITKO.CICSAGNT.CNTL.UNLOAD')
   DATASET('ITKO.CICSAGNT.CNTL') VOLUME(VOL001)

/*
```

Ensure the TCP/IP Interface is Installed

The CICS Agent requires that the CICS TCP/IP interface is installed and started. To verify that the TCP/IP interface is installed, use the transaction: EZAO, INQUIRE, CICS.

Create the CICS Agent Config File

The CICS agent communicates with the LPAR Agent through TCP/IP. The CICS Agent configuration file identifies the IP address and TCP port number of the LPAR Agent. This file is accessed as a Transient Data (TD) Queue during agent startup. This data set has attributes: DSORG=PS, RECFM=FB, LRECL=80.

Lines in the file with an asterisk ('*') in column 1 are ignored. The first record without an asterisk in column 1 contains the IP address and port number of the LPAR Agent. This record must be in the following format:

Bytes 1 to 15	**Byte 16**	**Bytes 17 to 21**	**Bytes 22 to 80**
IP address of LPAR Agent	blank	Port number of LPAR Agent	unused

The member CONFIG of the control PDS contains the following example:

```
* ITKO CICS Agent Configuration
* Columns 1 - 15 contain LPAR Agent IP address (15 chars)
* Columns 17 - 21 contain LPAR Agent port number (5 chars)
192.168.0.100   2998
```

Define CICS Agent Resources to CICS

The following table lists the resources that must be defined to CICS:

Name	Type	Description
TKOC	TDQUEUE	Configuration Transient Data Queue
TKOI	TRANSACTION	Transaction to install CICS exits and start agent transaction (TKOA). Typically run during PLTPI processing.
TKOR	TRANSACTION	Transaction to remove CICS exits and stop agent transaction (TKOA). Typically run during PLTSD processing.
TKOA	TRANSACTION	Agent transaction. Typically started by TKOI and stopped by TKOR, although it may be started and stopped by the monitor transaction (TKOM).
TKOM	TRANSACTION	Monitor transaction. Used to manually start and stop exits and agent and monitor agent processing.
ITKOAGNT	PROGRAM	Agent program. Started by TKOA.
ITKOINST	PROGRAM	Exit installation program. Started by TKOI.
ITKOREMV	PROGRAM	Exit removal program. Started by TKOR.
ITKPCRQ	PROGRAM	CICS XPCREQ (before CICS LINK exit). Enabled by TKOI, disabled by TKOR.
ITKPCRC	PROGRAM	CICS XPCREQC exit (after CICS LINK exit). Enabled by TKOI, disabled by TKOR.
ITKOXEIN	PROGRAM	CICS XEIIN exit (before CICS Command exit) Enabled by TKOI, disabled by TKOR.
ITKOXEIO	PROGRAM	CICS XEIOUT exit (after CICS Command exit) Enabled by TKOI, disabled by TKOR.
ITKOTABN	PROGRAM	CICS XPCABND exit (Task Abend exit) Enabled by TKOI, disabled by TKOR.
ITKOMON	PROGRAM	Monitor program.

JCL for a job to install these resources through DFHCSDUP is in the control PDS named DFHCSDUP.

Add Exit Enable/Disable to the CICS PLT

You can start the CICS Agent and exits manually with the TKOI transaction.

You can start the CICS Agent and exits manually with the TKOR transaction.

A simple way to enable the CICS exits and start the agent running is to place entries in the PLTSI (CICS startup). A PLTPI example follows. If the SIP has PLTPI=SI, then the following table would be assembled as DFHPLTSI. The CICS TCP/IP interface is started before the DevTest exits and agent.

```
DFHPLT TYPE=INITIAL,SUFFIX=SI
*
* The following programs are run in the second pass of PLTPI
*
         DFHPLT TYPE=ENTRY,PROGRAM=DFHDELIM
* The following programs are run in the third pass of PLTPI
*
         DFHPLT TYPE=ENTRY,PROGRAM=EZACIC20
         DFHPLT TYPE=ENTRY,PROGRAM=ITKOINST
         DFHPLT TYPE=FINAL
         END
```

A way to disable the DevTest exits and stop the agent when it shuts down is to include an entry in the PLTSD (CICS shutdown). A PLTSD example follows. If the SIP has PLTSD=SD, then the following table is assembled as DFHPLTSD. The DevTest exits and agent are stopped before the CICS TCP/IP interface.

```
DFHPLT TYPE=INITIAL,SUFFIX=SD
*
* The following programs are run in the first pass of PLTSD
*
         DFHPLT TYPE=ENTRY,PROGRAM=ITKOREMV
         DFHPLT TYPE=ENTRY,PROGRAM=EZACIC20
*
         DFHPLT TYPE=ENTRY,PROGRAM=DFHDELIM
* The following programs are run in the second pass of PLTSD
*
         DFHPLT TYPE=FINAL
         END
```

Change CICS Startup JCL

Add the DevTest CICS Agent load modules to CICS by copying them to an existing DFHRPL load library. Alternatively, you could add the DevTest CICS Agent load library to the DFHRPL concatenation.

JCL must also be added to define the agent configuration file (from Step 5 (see page 103)). If the data set is named ITKO.AGENT.CONFIG and the TD Queue TKOC was defined with DDNAME(ITKOACFG), the following JCL statement could be used:

```
//ITKOACFG DD DSN=ITKO.AGENT.CONFIG,DISP=SHR
```

CICS Agent Storage Requirements

The DevTest CICS agent is a long running task. The agent requires approximately 16K of program storage in ESDSA and approximately 1K of dynamic storage in ECDSA.

The DevTest CICS exits require approximately 34K of program storage in ERDSA and approximately 3K of dynamic storage (for each transaction) in ECDSA.

The recorded images of a CICS LINK are built in ECDSA when using the VSE recorder.

The exits build recorded images and pass to the CICS agent. The CICS agent sends them to the LPAR agent and then frees them, so they are short lived.

The size of a recorded image is approximately 330 + 2 * ((COMMAREA length or sum of all CONTAINER lengths) + INPUTMSG length).

The amount of ECDSA required to accommodate the recorded images depends on the average size that is calculated in the previous paragraph, and the transaction rate.

Playback images of a CICS LINK are also built in ECDSA when a Virtual Service is deployed.

At the beginning of a virtualized CICS LINK, the exits build a playback request and pass it to the CICS agent. The CICS agent forwards it to the LPAR agent.

When the CICS agent receives the playback response, the response is matched to the original request. Both the request and response are passed to the exits, which free them and complete the CICS LINK.

The size of a playback request and response is approximately 300 + (COMMAREA length or sum of all CONTAINER lengths).

The amount of ECDSA required to accommodate the playback images depends on the average size that is calculated in the previous paragraph, and the transaction rate.

CICS Exit Considerations

The DevTest CICS Agent uses the following CICS exit points:

- XPCREQ (before CICS LINK exit)
- XPCREQC (after CICS LINK exit)
- XEIIN (before CICS Command exit)
- XEIOUT (after CICS Command exit)
- XPCABND (Transaction ABEND)

XPCREQ and XPCREQC are used for CICS LINK virtualization.

XEIIN and XEIOUT are used for CICS Distributed Transaction Processing (DTP) virtualization.

XPCABND (and the other exits) is used for CA Continuous Application Insight.

XPCREQ and XPCREQC are required. If you do not require DTP virtualization or CAI, you can remove XEIIN and XEIOUT. To remove them, delete ITKOXEIN and ITKOXEIO from the load library (or rename them).

If you do not require CAI, you can remove XPCABND. To remove it, delete ITKOTABN from the load library (or rename it).

Coexistence with Other Exits

If any exits of type XPCREQ, XPCREQC, XEIIN, XEIOUT, or XPCABND are already installed, the DevTest exits may be able to coexist with them.

CICS calls the exits in the order in which they are enabled. You can control the order with the placement of the ITKOINST entry in the PLT.

DevTest CICS Agent Exit Behavior Summary

- The DevTest CICS Agent exits can be entered with a previous return code (addressed by UEPCRCA) that is nonzero. In this case, the exit returns with the return code (R15) set to the previous return code (which another exit set). The DevTest exits can coexist with other exits if they are not the first exit that CICS invokes.
- The DevTest CICS LINK exits (XPCREQ and XPCREQC) use the token (UEPPCTOK) passed by CICS to the exits to pass a DevTest control block address. If another XPCREQ or XPCREQC corrupts that address, then an abend can occur, or the DevTest XPCREQC exit can write a one-time message: **ITKO0014 - ERROR - Corrupt token in ITKPCRC Exit**.
- If the DevTest CICS Agent **before** exits (XPCREQ and XEIIN) must bypass a command to virtualize it, R15 returns a return code of **bypass** (UERCBYP). If CICS invokes another exit of the same type after the DevTest exit, the exit should propagate the bypass return code back to CICS. If it does not, the command is not bypassed, and virtualization fails.
- If the **recursion** indicator (addressed by UEPRECUR) is nonzero, the DevTest XPCREQ and XPCREQC exits return immediately. This return eliminates issues that can arise if another exit of this type uses a CICS LINK command.
- The XPCABND (Transaction ABEND) exit always returns with UEPCRCA in R15, so it should coexist with other XPCABND exits. The only exception to this rule is the event of an XPI function returning **task purged** (see the following).
- If any of the DevTest CICS Agent exits receive a return code of **task purged** from the CICS XPI (Exit Programming Interface), it returns with **purge** (UERCPURG) in R15. The exit overrides the previous return code (addressed by UEPCRCA) with UERCPURG. If CICS invokes another exit of the same type after the DevTest exit, the exit should propagate the purge return code back to CICS.

CICS Agent User Exits

ITKOUEX1 - DevTest CICS Agent Initialization Exit

This exit is called during the DevTest CICS Agent initialization. The exit can be used to:

- Indicate the user group that this CICS region is a member of
- Indicate the CICSPlex that this CICS region is a member of
- Allocate storage and pass the address/length to other exits using the token parameter
- Perform other initialization as required by other user exits

This exit is invoked through a CICS LINK, so it can be written in any language that CICS supports. During initialization, a CICS LINK to ITKOUEX1 is attempted.

If this exit is not necessary, take one of the following actions:

- Do not include a module that is named ITKOUEX1 in the DFHRPL concatenation.
- Disable the exit in the PPT.

Parameters are passed in COMMAREA.

The following table describes the parameters:

Parameter	**Length in Bytes**	**Input or Output**	**Data Type**	**Description**
Token	8	Input/Output	Unknown	Eight bytes that may be used by the user exits. Maintained across exit calls.
Group	8	Output	Character	Set by exit. Identifies a user group of which this CICS is a member.
CICSPlex	8	Output	Character	Set by exit. Identifies a CICSPlex of which this CICS is a member.
Sysid	4	Input	Character	CICS SYSID from CICS ASSIGN SYSID().
Applid	8	Input	Character	CICS APPLID from CICS ASSIGN APPLID().
Jobname	8	Input	Character	CICS job name from CICS INQUIRE SYSTEM JOBNAME().
LPAR	8	Input	Character	LPAR name from CVTSNAME in which this CICS resides.
Sysplex	8	Input	Character	SYSPLEX name from ECVTSPLX in which this CICS resides.

ITKO.CICSAGNT.CNTL(ITKOUEX1) contains a sample COBOL ITKOUEX1. This sample sets the user group to the first four characters of the CICS job name.

ITKO.CICSAGNT.CNTL(ITKOEX1C) contains the COBOL COMMAREA copybook.

ITKO.CICSAGNT.CNTL(ITKOEX1A) contains the Assembler COMMAREA copybook.

ITKO.CICSAGNT.CNTL(COMPUEX1) contains COBOL compile JCL.

ITKOUEX2 - DevTest CICS Agent Termination Exit

This exit is called during the DevTest CICS Agent termination. It can be used to perform any cleanup that user exits require (for example, free storage and close files).

This exit is invoked through a CICS LINK, so it can be written in any language that CICS supports. During the agent termination, a CICS LINK to ITKOUEX2 is attempted. If this exit is not necessary, take one of the following actions:

- Do not include a module that is named ITKOUEX2 in the DFHRPL concatenation.
- Disable the exit in the PPT.

Parameters are passed in COMMAREA.

The following table describes the parameters:

Parameter	Length in Bytes	Input or Output	Data Type	Description
Token	8	Input/Output	Unknown	Eight bytes that may be used by the user exits. Maintained across exit calls.

ITKO.CICSAGNT.CNTL(ITKOUEX2) contains sample COBOL ITKOUEX2.

ITKO.CICSAGNT.CNTL(ITKOEX2C) contains the COBOL COMMAREA copybook.

ITKO.CICSAGNT.CNTL(ITKOEX2A) contains the Assembler COMMAREA copybook.

ITKO.CICSAGNT.CNTL(COMPUEX2) contains the COBOL compile JCL.

ITKOUEX3 - LISA CICS COMMAREA Consolidation Exit

This exit is used to manage a COMMAREA that contains addresses that refer to storage outside of the COMMAREA. This exit consolidates the storage fragments with the COMMAREA to create a single contiguous COMMAREA.

This exit is called:

- During the CICS LINK recording, for both the before and after images, to create a replacement COMMAREA with no addresses referencing data outside the COMMAREA. The exit is responsible for GETMAINing the new storage area. The DevTest CICS Agent will FREEMAIN the area. Addresses in the COMMAREA that point to locations within the COMMAREA are also problematic. In this case, the exit should convert the addresses to relative offsets during recording and revert them to real addresses during playback.
- During CICS LINK playback (virtualization) to restore the original COMMAREA from the DevTest response.
- During CICS Agent initialization (after user exit 1) so the exit can perform any additional initialization tasks.
- During CICS Agent termination (before user exit 2) so that the exit can perform any additional termination tasks.

This exit is invoked through a CICS LINK, so it can be written in any language that CICS supports. However, assembler language is best suited to address manipulation and variable length data movement. During initialization, a CICS LINK to ITKOUEX3 is attempted.

If this exit is unnecessary, take one of the following actions:

- Do not include a module named ITKOUEX3 in the DFHRPL concatenation.
- Disable the exit in the PPT.

Parameters are passed in COMMAREA.

The following table describes the parameters:

Parameter	Length in Bytes	Input or Output	Data Type	Description

Token	8	Input/Output	Unknown	Eight bytes that may be used by the user exits. Maintained across exit calls.
Call Type	1	Input	Binary	12 indicates Playback 16 indicates Termination
Image Type	1	Input	Binary	4 indicates a Before Image 8 indicates an After Image
COMMAREA Address	4	Input	Address	Address of COMMAREA passed on the application CICS LINK.
COMMAREA Length	2	Input	Binary	Length of COMMAREA passed on the application CICS LINK.
Calling Program	8	Input	Character	Name of program issuing the CICS LINK command.
Target Program	8	Input	Character	Name of program being LINKed to.
New COMMAREA Address	4	Input/Output	Address	For recording: The address of the new consolidated COMMAREA is placed here by this exit. This exit typically GETMAINs the area. For Playback: The address of the new consolidated COMMAREA from the VSE response is placed here. This exit copies the contents to the application COMMAREA.
New COMMAREA Length	2	Input/Output	Binary	For recording: The length of the new consolidated COMMAREA is placed here by this exit. For Playback: The length of the new consolidated COMMAREA from the VSE response is placed here for this exit to use.

Return Code	1	Output	Binary	0 indicates that the COMMAREA was successfully consolidated. The DevTest CICS Agent will use the **New COMMAREA Address** and **New COMMAREA Length** to create the recorded image. 4 indicates that this exit was successful; however there is no new COMMAREA created. The DevTest CICS Agent uses the COMMAREA from the CICS LINK to create the recorded image. 8 indicates that this exit failed and there is no new COMMAREA. The DevTest CICS Agent returns to the application from the CICS LINK with EIBRCODE, EIBRESP and EIBRESP2 as indicated in the following parameters. 12 indicates all the processing of return code 8, and the indication that this exit should be removed from future DevTest CICS Link recording and playback.
EIBRCODE	6	Output	Binary	The EIBRCODE value to be used when the **Return Code** is 8 or 12.
EIBRESP	4	Output	Binary	The EIBRRESP value to be used when the **Return Code** is 8 or 12.
EIBRESP2	4	Output	Binary	The EIBRRESP2 value to be used when the **Return Code** is 8 or 12.

Assembler language sample ITKOUEX3 is in ITKO.CICSAGNT.CNTL(ITKOUEX3).

This sample manages a COMMAREA in the format that the Software AG Natural language produces when calling COBOL with the Natural parameter CALLRPL=(ALL,3) or CALLRPL=(ALL,4). With this format of COMMAREA, the COMMAREA length is 12 (when CALLRPL=(ALL,3) is used) and 16 (when CALLRPL=(ALL,4) is used). The first fullword in the COMMAREA is the address of a parameter list. The high-order bit set to 1 in the address indicates the last parameter. The third fullword is the address of a list of parameter lengths.

ITKO.CICSAGNT.CNTL(ITKOEX3A) contains the Assembler COMMAREA copybook.

ITKO.CICSAGNT.CNTL(ASMBUEX3) contains the assembly JCL.

CICS Agent Operation

Starting the DevTest CICS Agent and Exits

The CICS Agent and exits are started during CICS startup PLT processing. The agent and exits can also be started manually by running the TKOI transaction or with the monitor transaction TKOM. The TKOI transaction installs the exits and starts the agent transaction, TKOA.

Stopping the DevTest CICS Agent and Exits

The CICS Agent and exits are stopped during CICS shutdown PLT processing. The agent and exits can also be stopped manually by running the TKOR transaction or with the monitor transaction TKOM.

In rare circumstances, the CICS Agent transaction, TKOA, can fail to terminate. If you suspect this is the case, use the CEMT I TAS command to confirm whether TKOA is still running. If it is, use the CEMT SET TAS(nnn) PURGE or CEMT SET TAS(nnn) FORCEPURGE command as appropriate to terminate TKOA.

Impact on Virtualized CICS LINK Commands

In rare cases, virtualization of a CICS LINK can fail. In this event, the CICS LINK returns to the application with the "Invalid Request" (INVREQ) condition: EIBRCODE=x'E00000000000' and EIBRESP=16. This situation typically results in a transaction abend, unless the application has INVREQ condition handling implemented. You can use EIBRESP2 to determine the exact cause of the virtualization failure:

500 / x'1F4'

No playback response was received from the VSE. This is most likely due to a timeout. The timeout interval is set to 10 seconds. Contact Customer Support if the timeout value must be changed.

501 / x'1F5'

DevTest CICS Agent is down. The "After CICS LINK" exit detected that the DevTestCICS Agent is down. No playback response can be expected.

502 / x'1F6'

Virtualization failed. A playback response was received from the Service Virtualization and updating the CICS LINK COMMAREA or CHANNEL/CONTAINERs failed. See the CICS job log for a CICS command failure message.

503 / x'1F7'

The DevTest LPAR Agent was shut down, resulting in cancellation of all pending playback requests.

Impact on Virtualized CICS DTP/MRO Commands

In rare cases, virtualization of a CICS DTP/MRO command (ALLOCATE, FREE, SEND, RECEIVE, CONVERSE, or EXTRACT ATTRIBUTES) can fail. In this event, the CICS command returns to the application with the "Invalid Request" (INVREQ) condition: EIBRCODE=x'E00000000000' and EIBRESP=500. This situation typically results in a transaction abend, unless the application has INVREQ condition handling in place. You can use EIBRESP2 to determine the exact cause of the virtualization failure:

501 / x'1F5'

DevTest CICS Agent is down. The "After CICS LINK" exit detected that the DevTest CICS Agent is down. No playback response can be expected.

502 / x'1F6'

No response was received from Service Virtualization.

503 / x'1F7'

An invalid response was received from the Service Virtualization. See the CICS job log for a CICS command failure message.

504 / x'1F8'

The session table was full.

505 / x’1F9’

An attempt to add the playback request to the playback queue failed.

506 / x’1FA’

The XEIIN WAIT timeout expired before a response was received from the Service Virtualization.

507 / x’1FB’

A GETMAIN for the request or response failed.

Monitor Transaction (TKOM)

With the Monitor Transaction (TKOM), you can:

- Check the status of the agent and exits
- View the Global Work Area (GWA) in hexidecimal
- View some of the GWA fields formatted/decoded
- Check the recording and playback counts
- Check the program table counts
- View the program table

- Start and stop the agent
- Start and stop the exits
- Check the current depth (count) and high watermarks (Max) of the three main queues:

  Recording Queue (RecQ):

  - Defines the queue of recorded images.

  Playback Queue (PlbQ):

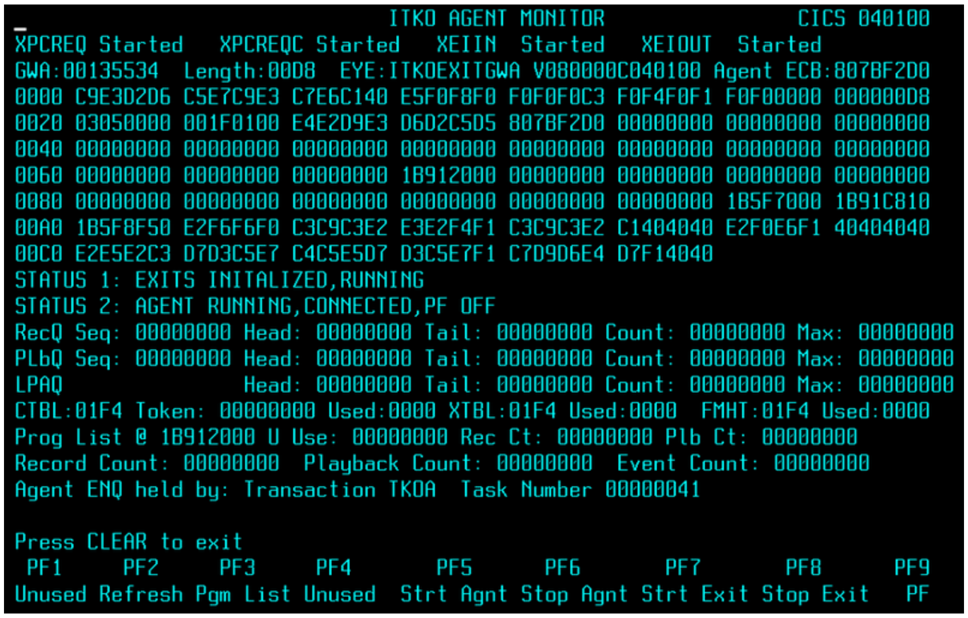
  - Defines the queue of playback images to be sent to the DevTest LPAR Agent.

  LPAR Agent Queue (LPAQ):

  - Defines the queue of playback images waiting for a response from the DevTest LPAR Agent.

- Determine the task number of the agent transaction

The Monitor Transaction displays as follows:



```
_                               ITKO AGENT MONITOR                CICS 040100
XPCREQ Started   XPCREQC Started   XEIIN  Started   XEIOUT  Started
GWA:00135534  Length:00D8  EYE:ITKOEXITGWA V080000C040100 Agent ECB:807BF2D0
0000 C9E3D2D6 C5E7C9E3 C7E6C140 E5F0F8F0 F0F0F0C3 F0F4F0F1 F0F00000 000000D8
0020 03050000 001F0100 E4E2D9E3 D6D2C5D5 807BF2D0 00000000 00000000 00000000
0040 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
0060 00000000 00000000 00000000 1B912000 00000000 00000000 00000000 00000000
0080 00000000 00000000 00000000 00000000 00000000 00000000 1B5F7000 1B91C810
00A0 1B5F8F50 E2F6F6F0 C3C9C3E2 E3E2F4F1 C3C9C3E2 C1404040 E2F0E6F1 40404040
00C0 E2E5E2C3 D7D3C5E7 C4C5E5D7 D3C5E7F1 C7D9D6E4 D7F14040
STATUS 1: EXITS INITALIZED,RUNNING
STATUS 2: AGENT RUNNING,CONNECTED,PF OFF
RecQ Seq: 00000000 Head: 00000000 Tail: 00000000 Count: 00000000 Max: 00000000
PLbQ Seq: 00000000 Head: 00000000 Tail: 00000000 Count: 00000000 Max: 00000000
LPAQ               Head: 00000000 Tail: 00000000 Count: 00000000 Max: 00000000
CTBL:01F4 Token: 00000000 Used:0000 XTBL:01F4 Used:0000  FMHT:01F4 Used:0000
Prog List @ 1B912000 U Use: 00000000 Rec Ct: 00000000 Plb Ct: 00000000
Record Count: 00000000  Playback Count: 00000000  Event Count: 00000000
Agent ENQ held by: Transaction TKOA  Task Number 00000041

Press CLEAR to exit
 PF1     PF2     PF3     PF4       PF5      PF6       PF7       PF8      PF9
Unused Refresh Pgm List Unused  Strt Agnt Stop Agnt Strt Exit Stop Exit   PF
```

CICS Agent Messages

Messages are written to the operator console and CICS job log by the DevTest CICS Agent and exits.

The DevTest exit installer (Transaction TKOI, Program ITKOINST) writes the following messages:

- ITKO0001 - iTKO exit init failed. EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999
- ITKO0002 - iTKO exit [XPCREQ | XPCREQC] at xxxxxxxx GWA at xxxxxxxx Len 9999 is enabled and started
- ITKO0003 - iTKO exit [XPCREQ | XPCREQC] was enabled but not started
- ITKO0004 - iTKO exit GWA error - Address is xxxxxxxx Length is 9999

The DevTest exit uninstaller (Transaction TKOR, Program ITKOREMV) writes the following messages:

- ITKO0005 - iTKO exit remove failed. EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999
- ITKO0006 - iTKO exit [XPCREQ | XPCREQC] disabled

The CICS XPCREQ (before CICS LINK) Exit (module ITKPCRQ) or the CICS XPCREQC (after CICS LINK) exit (module ITKPCRC) write the following messages:

- ITKO0007 - GWA missing or too small for [ITKPCRQ | ITKPCRC] Exit
- ITKO0008 - Program program-name Error EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999
- ITKO0009 - Recording Abandoned of calling-program LINK to called-program Tran xxxx Task 9999999 Term xxxx
- ITKO0010 – [Recorded Image | Playback Image | LPAR Agent] PLO failed in [ITKPCRQ | ITKPCRC] Exit
- ITKO0011 - BEFORE/AFTER image creation failed with reason code xxx
- ITKO0012 - Image update failed with reason code 999
- ITKO0013 - PDU Parsing Error in ITKPCRC Exit
- ITKO0014 - ERROR - Corrupt token in ITKPCRC Exit

The DevTest CICS Agent (Transaction TKOA, Program ITKOAGNT) writes the following messages:

- ITKO0101 - Agent waiting for exits
- ITKO0102 - Agent initialization complete
- ITKO0103 - Agent shutting down
- ITKO0104 - iTKO [Recorded Image | Playback Image | LPAR Agent Img] PLO failed in Agent
- ITKO0105 - iTKO Agent started while another agent is running
- ITKO0106 - iTKO Agent connected to LPAR Agent
- ITKO0107 - Agent Config LPAR Agent IP: nnn.nnn.nnn.nnn Port 99999
- ITKO0108 - iTKO Agent GWA error- Address is xxxxxxxx Length is 9999
- ITKO0109 - Socket func xxxxxxxx failed. RETCODE=xXXXXXXXX ERRNO=999999
- ITKO0110 - Agent CICS cmd failed. EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999
- ITKO0111 - iTKO Agent disconnected from LPAR Agent
- ITKO0112 - iTKO Agent error sending recorded image to LPAR Agent
- ITKO0113 - iTKO Agent error sending playback image to LPAR Agent
- ITKO0114 - iTKO Agent error sending response to LPAR Agent
- ITKO0115 - iTKO Agent error receiving request from LPAR Agent
- ITKO0116 - iTKO Agent error - invalid LPAR Agent IP address
- ITKO0117 - iTKO Agent released nnnnnn recorded images
- ITKO0118 - iTKO Agent canceled nnnnnn playback images
- ITKO0119 - iTKO Agent canceled nnnnnn pending playback images
- ITKO0120 - ITKO Agent released playback image: token xxxxxxxxxxxxxxxx Reason xxx
- ITKO0121 - iTKO Agent error reading configuration
- ITKO0122 - Agent TCP/IP API Timeout
- ITKO0123 - Agent waiting for TCP/IP API
- ITKO0124 - ITKO Agent response timeout for Token xxxxxxxxxxxxxxxx
- ITKO0125 - Call to ITKOUEX1/ITKOUEX2/ITKOUEX3 failed
- ITKO0126 - Handshake error encountered. Shutting down Agent

The CICS XEIIN (before CICS command) Exit (module ITKOXEIN) or the CICS XEIOUT (after CICS command) exit (module ITKOXEIO) write the following messages:

- ITKO200 - ITKOXEIN XPI INQ_APPLICATION_DATA Failed
- ITKO201 - ITKOXEIN/ITKOXEIO XPI Getmain Dynamic Stg Failed

- ITKO202 - ITKOXEIN/ITKOXEIO XPI Freemain Dynamic Stg Failed
- ITKO203 - ITKOXEIN/ITKOXEIO XPI Getmain Image Stg Failed
- ITKO204 - ITKOXEIN/ITKOXEIO XPI Freemain Image Stg Failed
- ITKO205 - ITKOXEIN Task table full - discarding image
- ITKO206 - ITKOXEIN XPI WAIT Failed
- ITKO207 - ITKOXEIN XPI Freemain plbk req/rsp failed
- ITKO208 - ITKOXEIO Add to recording queue failed
- ITKO209 - ITKOXEIN/ITKOXEIO XPI Freemain SET() Stg Failed
- ITKO210 - ITKOXEIN XPI Getmain SET() Stg Failed

More information:

DevTest CICS Agent Message Descriptions (see page 120)

DevTest CICS Agent Message Descriptions

Click the following links for descriptions of each message:

ITKO0001

ITKO0001 - iTKO exit init failed. EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999

Message Type:

Fatal Error

Description:

A CICS command failed and ITKOINST was aborted. The hexidecimal EIBFN value identifies the CICS command that failed. The command that failed is:

EIBFN	CICS Command
x0C02	GETMAIN
x1008	START
x2202	ENABLE PROGRAM
x2206	EXTRACT EXIT
x6C02	WRITE OPERATOR
x8802	INQUIRE EXITPROGRAM

Action:

Using the EIBRCDE, EIBRESP, and EIBRESP2 codes, determine the cause of the failure and correct it. Some possible causes of failure:

- The user running the TKOI transaction does not have the authority to execute CICS system programming interface (SPI) commands (such as EXTRACT EXIT).
- A START command for the agent transaction, TKOA, failed because TKOA was not defined or was defined incorrectly.

ITKO0002

ITKO0002 - iTKO exit [XPCREQ | XPCREQC] at xxxxxxxx GWA at xxxxxxxx Len 9999 is enabled and started

Message Type:

Informational

Description:

The indicated exit (XPCREQ or XPCREQC) was installed at the indicated address with a Global Work Area at the indicated address of the indicated length. The exit was enabled and started.

Typically, two of these messages appear: one for XPCREQ and one for XPCREQC. Both have the same GWA address and length.

Action:

None

ITKO0003

ITKO0003 - iTKO exit [XPCREQ | XPCREQC] was enabled but not started

Message Type:

Fatal Error

Description:

The indicated exit (XPCREQ or XPCREQC) was enabled; however, it was not possible to start the exit. The ITKO recording and playback capability do not function properly.

An ITKO0001 message typically accompanies this message.

Action:

Diagnose the accompanying ITKO0001 message and correct the issue.

Remove the exits with TKOR and reinstall with TKOI.

ITKO0004

ITKO0004 - iTKO exit GWA error - Address is xxxxxxxx Length is 9999

Message Type:

Fatal Error

Description:

The ITKO exits were allocated a Global Work Area that is too small.

An ITKO0001 message can accompany this message.

Action:

Report the error to CA support.

ITKO0005

ITKO0005 - iTKO exit remove failed. EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999

Message Type:

Fatal Error

Description:

A CICS command failed and ITKOREMV was aborted. The hexidecimal EIBFN value identifies the CICS command that failed. The command that failed is:

EIBFN	CICS Command
x0C02	FREEMAIN
x1004	DELAY
x2204	DISABLE PROGRAM
x2206	EXTRACT EXIT
x6C02	WRITE OPERATOR
x8802	INQUIRE EXITPROGRAM

Action:

Using the EIBRCDE, EIBRESP, and EIBRESP2 codes, determine the cause of the failure and correct it. This failure can occur if the user running the TKOR transaction lacks the authority to execute CICS system programming interface (SPI) commands (such as EXTRACT EXIT).

ITKO0006

ITKO0006 - iTKO exit [XPCREQ | XPCREQC] disabled

Message Type:

Informational

Description:

The indicated exit (XPCREQ or XPCREQC) was disabled.

Action:

None

ITKO0007

ITKO0007 - GWA missing or too small for [ITKPCRQ | ITKPCRC] Exit

Message Type:

Fatal Error

Description:

The exit Global Work Area for the exits does not exist or is too small.

Action:

Report the error to CA support.

ITKO0008

ITKO0008 - Program program-name Error EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999

Message Type:

Fatal Error

Description:

A CICS command failed in the indicated exit program. The hexidecimal EIBFN value identifies the CICS command that failed. The command that failed is:

EIBFN	CICS Command
0202	ADDRESS
0208	ASSIGN INVOKINGPROG
0C02	GETMAIN
0C02	FREEMAIN
3414	GET CONTAINER
3416	PUT CONTAINER
6C02	WRITE OPERATOR
9626	STARTBROWSE CONTAINER
9628	GETNEXT CONTAINER

Action:

Using the EIBRCDE, EIBRESP, and EIBRESP2 codes, determine the cause of the failure and correct it. One possible cause of the failure is that a GETMAIN failed due to a CICS short on storage condition.

ITKO0009

ITKO0009 - Recording Abandoned of calling-program LINK to called-program Tran xxxx Task 9999999 Term xxxx

Message Type:

Warning

Description:

When recording a CICS LINK from calling program to called program for the indicated transaction, task, and terminal, no resource was available to handle the recorded image. An accompanying ITKO0008 message can identify the resource (such as a GETMAIN failure for the recorded image). The DevTest LPAR Agent could have disconnected from the DevTest CICS Agent.

Action:

Investigate the condition causing the recorded image to be dropped.

ITKO0010

ITKO0010 - [Recorded Image | Playback Image | LPAR Agent list] PLO failed in [ITKPCRQ | ITKPCRC] Exit

Message Type:

Warning

Description:

The PLO (z/OS Perform Locked Operation) command failed.

PLO is used to serialize the manipulation of three queues/lists used to communicate between the exits and the agent. The Recorded Image queue contains recorded images to for the agent to process. The Playback Image queue contains playback requests for the agent to process. The LPAR Agent List contains the playback responses that the agent receives to be passed to the exits.

For a Recorded Image queue entry, the exit discards the recorded image.

For a Playback Image queue entry, the image is discarded and the transaction continues without playback (virtualization).

For an LPAR Agent List entry (playback response), the image is orphaned on the LPAR Agent List by the exit.

Action:

Report the error to Support.

ITKO0011

ITKO0011 - BEFORE/AFTER image creation failed with reason code xxx

Message Type:

Error

Description:

In exit XPCREQ (module ITKPCRQ) or exit XPCREQC (module ITKOPCRC), the creation of a before or after image failed with the indicated reason code.

The following reason codes are possible:

8 - No image was passed to the create routine.

12- Neither a before or after image in the create routine.

16- Exceeded PDU length inserting INPUTMSG in the create routine.

20- Exceeded PDU length inserting COMMAREA in the create routine.

24- Exceeded PDU length inserting Container in the create routine.

28- Exceeded PDU length inserting Origin Data in the create routine.

32- MVCL logic error in the create routine.

36- No TKOCLNK in the image that was passed to the create routine.

40- CPDUVARN is zero in the create routine.

44- Exceeded PDU length inserting EIB in the create routine.

Action:

Report the error to ITKO Support.

ITKO0012

ITKO0012 - Image update failed with reason code 999

Message Type:

Error

Description:

In the exit XPCREQ (module ITKPCRQ) or exit XPCREQC (module ITKOPCRC), the update an after image failed with the indicated reason code.

The following reason codes are possible:

4- A CICS command error occurred.

5- No image was passed to the update routine.

6- No after EIB in the image that was passed to the update routine.

7- No TKOCLNK in the image that was passed to the update routine.

8- CVARTYP not CVARLINK in the image that was passed to the update routine.

9- CLNKEYE not 'CLNK' in the image that was passed to the update routine.

10-CVARLEN is zero in the image that was passed to the update routine.

11-CTNREYE not 'CTNR' in the image that was passed to the update routine.

Action:

Report the error to Support.

ITKO0013

ITKO0013 - PDU Parsing Error in ITKPCRC Exit

Message Type:

Error

Description:

In the exit XPCREQC (module ITKPCRC), the update of a COMMAREA that is expanded in User Exit 3 failed.

Action:

Report the error to Support.

ITKO0014

ITKO0014 - ERROR - Corrupt token in ITKPCRC Exit

Message Type:

Error

Description:

In the CICS LINK after exit XPCREQC (module ITKPCRC), a token (UEPPCTOK) was passed that does not address the proper DevTest CICS Agent control block. The cause is probably a conflicting XPCREQ or XPCREQC exit. This message is only written once.

Action:

Investigate the XPCREQ and XPCREQC exits installed in the CICS region.

ITKO0101

ITKO0101 - AGENT waiting for exits

Message Type:

Informational

Description:

The DevTest CICS Agent was started before the exits were enabled. The agent waits for the exits to be enabled before continuing.

Action:

None.

ITKO0102

ITKO0102 - AGENT initialization complete

Message Type:

Informational

Description:

The DevTest CICS Agent has completed initialization and starts connecting to the DevTest LPAR Agent.

Action:

None.

ITKO0103

ITKO0103 - AGENT shutting down

Message Type:

Informational

Description:

The DevTest CICS Agent is shutting down.

Action:

None.

ITKO0104

ITKO0104 - iTKO [Recorded Image | Playback Image | LPAR Agent Img] PLO failed in Agent

Message Type:

Warning

Description:

The PLO (z/OS Perform Locked Operation) command failed.

PLO is used to serialize the manipulation of three queues/lists used to communicate between the exits and the agent. The Recorded Image queue contains recorded images for the agent to process. The Playback Image queue contains playback requests for the agent to process. The LPAR Agent List contains the playback responses that the agent receives to be passed to the exits.

For Recorded and Playback Image queue entries, the image is skipped and the PLO is retried on the next pass of the Recorded/Playback Image queues.

For an LPAR Agent List entry (playback response), the image is discarded and the user transaction issuing the CICS LINK receives an INVREQ response (EIBRCODE=x’E00000000000’, EIBRESP=16 and EIBRESP2=500).

Action:

Report the error to Support.

ITKO0105

ITKO0105 - iTKO Agent started while another agent is running

Message Type:

Warning

Description:

A DevTest CICS Agent (transaction TKOA) was started while another instance of the agent is running. Only one agent can be running in a CICS region, so the second agent stops.

Action:

None.

ITKO0106

ITKO0106 - iTKO Agent connected to LPAR Agent

Message Type:

Informational

Description:

The DevTest CICS Agent has established a TCP/IP connection to the DevTest LPAR Agent.

Action:

None.

ITKO0107

ITKO0107 - Agent Config LPAR Agent IP: nnn.nnn.nnn.nnn Port 99999

Message Type:

Informational

Description:

The DevTest CICS Agent read the configuration (TD Queue TKOC) and is using the indicated IP address and port number to connect to the DevTest LPAR Agent.

Action:

None

ITKO0108

ITKO0108 - iTKO Agent GWA error- Address is xxxxxxxx Length is 9999

Message Type:

Fatal Error

Description:

The exit Global Work Area (GWA) does not exist or has an incorrect length. The DevTest CICS Agent stops.

Action:

Report the error to Support.

ITKO0109

ITKO0109 - Socket func [INITAPI | SELECTEX | SEND | RECV | CONNECT | CLOSE | PTON | TERMAPI] failed. RETCODE=xXXXXXXXX ERRNO=999999

Message Type:

Warning

Description:

An error occurred on the TCP/IP connection to the LPAR Agent.

The message **Socket func RECV   failed. RETCODE=x00000000 ERRNO=000000** typically indicates a disconnect from the LPAR Agent. An ITKO0111 message accompanies this message. An error in the SEND function can also indicate an LPAR Agent disconnect.

The ERRNO values are documented in the IBM publication z/OS Communications Server IP CICS Sockets Guide.

Action:

Determine if the error is due to a DevTest LPAR Agent disconnect or other environmental issue and correct the issue. Otherwise, report the error to Support.

ITKO0110

ITKO0110 - Agent CICS cmd failed. EIBFN=xXXXX EIBRCDE=xXXXXXXXXXXXX EIBRESP=9999 EIBRESP2=9999

Message Type:

Fatal Error

Description:

A CICS command failed in the DevTest CICS Agent. The hexidecimal EIBFN value identifies the CICS command that failed. The command that failed is:

EIBFN	CICS Command
0202	ADDRESS
0208	ASSIGN
0C02	GETMAIN/ FREEMAIN
0E02	LINK
1204	ENQ
1206	DEQ
2206	EXTRACT EXIT
4E02	INQUIRE PROGRAM
5402	INQUIRE SYSTEM
6C02	WRITE OPERATOR
8802	INQUIRE EXITPROGRAM

Action:

Using the EIBRCDE, EIBRESP and EIBRESP2 codes, determine the cause of the failure and correct it if possible. One possible cause of the failure is that the user ID running the TKOA transaction does not have the authority to execute CICS system programming interface (SPI) commands (such as EXTRACT EXIT).

ITKO0111

ITKO0111 - iTKO Agent disconnected from LPAR Agent

Message Type:

Informational

Description:

The DevTest CICS Agent has disconnected from the DevTest LPAR Agent.

Action:

None.

ITKO0112

ITKO0112 - iTKO Agent error sending recorded image to LPAR Agent

Message Type:

Error

Description:

The DevTest CICS Agent was in the process of sending a recorded image and the TCP/IP connection with the DevTest LPAR Agent was broken. The recorded image is dropped.

Action:

Investigate the cause of the DevTest LPAR Agent disconnect.

ITKO0113

ITKO0113 - iTKO Agent error sending playback image to LPAR Agent

Message Type:

Error

Description:

The DevTest CICS Agent was in the process of sending a playback request image and the TCP/IP connection with the LPAR Agent was broken. The recorded image is dropped, and the CICS LINK in progress completes with an INVREQ response (EIBRCODE=x’E00000000000’, EIBRESP=16 and EIBRESP2=500).

Action:

Investigate the cause of the DevTest LPAR Agent disconnect.

ITKO0114

ITKO0114 - iTKO Agent error sending response to LPAR Agent

Message Type:

Error

Description:

The DevTest CICS Agent was in the process of sending a response to the DevTest LPAR Agent and the TCP/IP connection was broken.

Action:

Investigate the cause of the CICS LPAR Agent disconnect.

ITKO0115

ITKO0115 - iTKO Agent error receiving request from LPAR Agent

Message Type:

Error

Description:

The DevTest CICS Agent was in the process of receiving a request from the DevTest LPAR Agent and the TCP/IP connection was broken.

Action:

Investigate the cause of the DevTest LPAR Agent disconnect.

ITKO0116

ITKO0116 - iTKO Agent error - invalid LPAR Agent IP address

Message Type:

Fatal Error

Description:

The DevTest LPAR Agent IP address in the DevTest CICS Agent configuration file (TD Queue TKOC) is not a valid dotted-decimal notation IPv4 address.

Action:

Correct the configured IP address.

ITKO0117

ITKO0117 - iTKO Agent released nnnnnn recorded images

Message Type:

Warning

Description:

The DevTest LPAR Agent disconnected from the DevTest CICS Agent while recorded images were queued. The DevTest CICS Agent removes the pending recorded images from the queue and releases them.

Action:

Investigate the cause of the DevTest LPAR Agent disconnect.

ITKO0118

ITKO0118 - iTKO Agent canceled nnnnnn playback images

Message Type:

Warning

Description:

The DevTest LPAR Agent disconnected from the DevTest CICS Agent while playback request images were queued. The DevTest CICS Agent removes the pending playback request images from the queue and releases them. Each playback request image represents a CICS LINK to be virtualized. These CICS LINK commands complete with an INVREQ response (EIBRCODE=x'E00000000000', EIBRESP=16 and EIBRESP2=500).

Action:

Investigate the cause of the DevTest LPAR Agent disconnect.

ITKO0119

ITKO0119 - iTKO Agent canceled nnnnnn pending playback images

Message Type:

Warning

Description:

The DevTest LPAR Agent disconnected from the DevTest CICS Agent while pending playback response images were expected. The DevTest CICS Agent removes the pending playback response images from the queue and releases them. Each pending playback response image represents a CICS LINK to be virtualized. These CICS LINK commands complete with an INVREQ response (EIBRCODE=x'E00000000000', EIBRESP=16 and EIBRESP2=500).

Action:

Investigate the cause of the DevTest LPAR Agent disconnect.

ITKO0120

ITKO0120 - iTKO Agent released playback image with token xxxxxxxxxxxxxxxx

Message Type:

Warning

Description:

The DevTest CICS Agent received a playback response from the DevTest LPAR Agent that does not match a pending playback request. The CICS Transaction waiting on the CICS LINK virtualization could not be identified. The playback image is released (discarded).

The following reason codes are possible. Codes 5 and 6 indicate that a playback response was received from the Virtual Service Environment after the CICS Agent 30-second timeout. An ITKO0124 message typically accompanies this message.

1- PDU too short.

2- TKOCPDU Format Error.

3- TKOCPDU Token prefix is not ITKO.

4- TKOCPDU Token suffix is x'00000000'.

5- LPAR Agent Queue is empty - request may have timed out.

6- Not found on LPAR Agent Queue - request may have timed out.

7- CVAR Format error.

8- No CICS Command header.

9- No after image EIB.

10-Request CPDU error.

11-Matching image on LPAR Agent Queue already posted.

12-CMALEYE not 'CMAL'.

13-CMFREYE not 'CMFR'.

14-CMSDEYE not 'CMSD'.

15-CMRCEYE not 'CMRC'.

16-CMCVEYE not 'CMCV'.

17-CMEXEYE not 'CMEX'.

18-The Receive command has no received data block.

19-The Converse command has no received data block.

20-The Receive command data block too short.

21 Converse command data block too short.

Action:

For reason codes 5 and 6, investigate the timeout. For all other reason codes, report the error to Support.

ITKO0121

ITKO0121 - iTKO Agent error reading configuration

Message Type:

Fatal Error

Description:

An error occurred reading the configuration TD Queue TKOC.

Action:

Check the definition and contents of TD Queue TKOC.

ITKO0122

ITKO0122 - Agent TCP/IP API Timeout

Message Type:

Fatal Error

Description:

The DevTest CICS Agent waits for the CICS TCP/IP interface expired and the DevTest CICS Agent stopped.

Action:

Ensure that the CICS TCP/IP interface is started and restart the agent.

ITKO0123

ITKO0123 - Agent waiting for TCP/IP API

Message Type:

Warning

Description:

The DevTest CICS Agent is waiting for the CICS TCP/IP interface to initialize. The DevTest CICS Agent retries 30 times in 10-second intervals (5 minutes). If the CICS TCP/IP interface does not initialize before the wait times out, the DevTest CICS Agent stops.

Action:

No action is required.

ITKO0124

ITKO0124 - ITKO Agent response timeout for Token xxxxxxxxxxxxxxxx

Message Type:

Warning

Description:

A playback image was sent to the Virtual Service Environment and a response was not received before the 30-second timeout. The playback request is purged and the CICS command terminates with an INVREQ response.

Action:

Check the VSE for errors.

ITKO0125

ITKO0125 - Call to ITKOUEX1/ITKOUEX2/ITKOUEX3 failed

Message Type:

Warning

Description:

The CICS LINK to ITKOUEX1, ITKOUEX2, or ITKOUEX3 failed. See the accompanying ITKO0110 message.

Action:

Correct the issue that the ITKO0110 message identified.

ITKO0126

ITKO0126 - Handshake error encountered. Shutting down Agent.

Message Type:

Error

Description:

The handshake between the DevTest CICS Agent and the DevTest LPAR agent failed. This failure is typically due to an incorrectly configured port number for the DevTest LPAR Agent in the DevTest CICS Agent configuration. The error causes the DevTest CICS Agent to connect to a server that is not a DevTest LPAR Agent.

Action:

Correct the DevTest CICS Agent configuration.

ITKO0200

ITKO200 - ITKOXEIN XPI INQ_APPLICATION_DATA Failed

Message Type:

Error

Description:

The XEIIN exit (module ITKOXEIN) received an error in the XPI INQ_APPLICATION_DATA command.

Action:

Report the error to Support.

ITKO0201

ITKOXEIN/ITKOXEIO XPI Getmain Dynamic Stg Failed

Message Type:

Error

Description:

The XEIIN exit (module ITKOXEIN) or XEIOUT exit (module ITKOXEIO) received an error in the XPI GETMAIN command while allocating dynamic storage.

Action:

Investigate the CICS short on storage condition.

ITKO0202

ITKOXEIN/ITKOXEIO XPI Freemain Dynamic Stg Failed

Message Type:

Error

Description:

The XEIIN exit (module ITKOXEIN) or XEIOUT exit (module ITKOXEIO) received an error in the XPI FREEMAIN command while releasing dynamic storage.

Action:

Report the error to Support.

ITKO0203

ITKOXEIN/ITKOXEIO XPI Getmain Image Stg Failed

Message Type:

Error

Description:

The XEIIN exit (module ITKOXEIN) or XEIOUT exit (module ITKOXEIO) received an error in the XPI GETMAIN command while allocating storage for a recorded image.

Action:

Investigate the CICS short on storage condition.

ITKO0204

ITKOXEIN/ITKOXEIO XPI Freemain Image Stg Failed

Message Type:

Error

Description:

The XEIIN exit (module ITKOXEIN) or XEIOUT exit (module ITKOXEIO) received an error in the XPI FREEMAIN command while releasing image storage.

Action:

Report the error to Support.

ITKO0205

ITKOXEIN Task table full - discarding image

Message Type:

Error

Description:

The XEIIN exit (module ITKOXEIN) tried to allocate an entry in the task table and the table was full.

Action:

Report the error to Support.

ITKO0206

ITKOXEIN XPI WAIT Failed

Message Type:

Error

Description:

The XEIIN exit (module ITKOXEIN) received an error in the XPI WAIT command while waiting for a playback response.

Action:

Report the error to Support.

ITKO0207

ITKOXEIN XPI Freemain plbk req/rsp failed

Message Type:

Error

Description:

The XEIIN exit (module ITKOXEIN) received an error in the XPI FREEMAIN command while releasing a playback request or response.

Action:

Investigate the CICS short on storage condition.

ITKO0208

ITKOXEIO Add to recording queue failed

Message Type:

Error

Description:

The XEIOUT exit (module ITKOXEIO) tried to add a recorded image to the recording queue and the attempt failed.

Action:

Report the error to Support.

ITKO0209

ITKOXEIN/ITKOXEIO XPI Freemain SET() Stg Failed

Message Type:

Error

Description:

The XEIIN exit (module ITKOXEIN) or XEIOUT exit (module ITKOXEIO) received an error in the XPI FREEMAIN command while releasing storage allocated for the SET parameter in a RECEIVE or CONVERSE command.

Action:

Report the error to Support.

ITKO0210

ITKOXEIN XPI Getmain SET() Stg Failed

Message Type:

Error

Description:

The XEIIN exit (module ITKOXEIN) received an error in the XPI GETMAIN command while allocating storage for the SET parameter in a RECEIVE or CONVERSE command.

Action:

Report the error to Support.

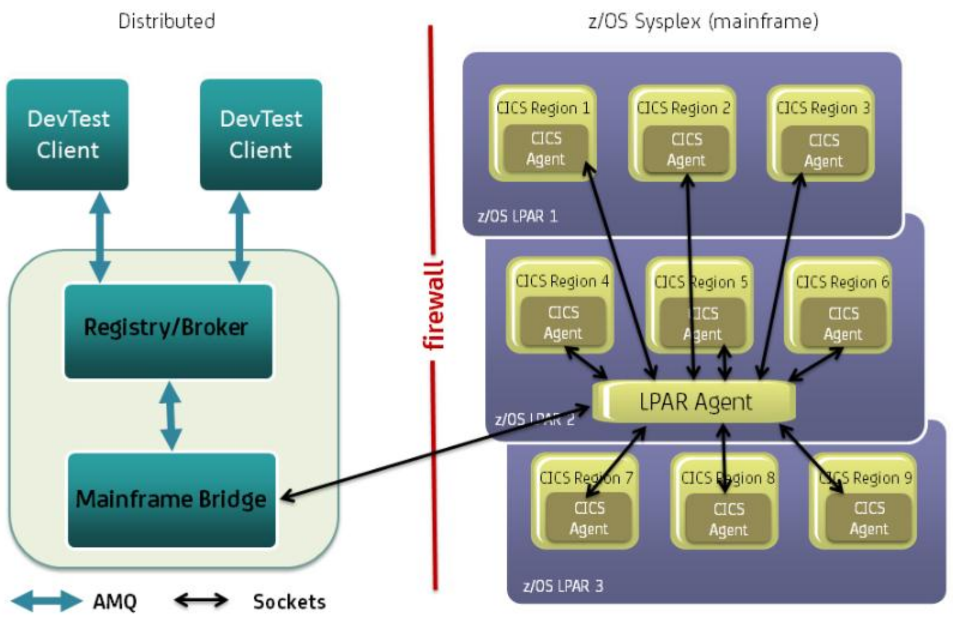
Chapter 4: DevTest LPAR Agent

The DevTest LPAR Agent is a z/OS resident agent that runs as a started task. The agent can be installed on any LPAR in the z/OS Sysplex. The agent provides a single point of contact for the DevTest registry (and therefore DevTest clients) to communicate with other DevTest z/OS agents. The agent enables DevTest to know which z/OS CICS agents are up and running and some basic configuration information about those agents. The LPAR Agent also provides a means to group DevTest z/OS CICS agents if multiple DevTest LPAR agents are defined to the z/OS Sysplex. The agent can operate in client mode (where it makes the connection to the DevTest registry). The agent can also operate in server mode (where it accepts connections from DevTest registries).

This section contains the following topics:

LPAR Agent Overview

The following graphic shows the components of the DevTest LPAR Agent and DevTest CICS Agent.



How to Install the LPAR Agent

The LPAR Agent package consists of three files:

- iTKOLPARAgentDoc.pdf
- iTKOLPARAgentLoad
- iTKOLPARAgentCntl

The steps for the installation are as follows:

1. FTP the Files to the Target z/OS System (see page 148).
2. Restore the Extended Partitioned Datasets (PDSEs) and Partitioned Dataset (PDS) (see page 149).
3. Set up the LPAR Agent to Run as a Started Task (see page 150).
4. Create the LPAR Agent Configuration Member (see page 151).

FTP the Files to the Target z/OS System

The files iTKOLPARAgentLoad and iTKOLPARAgentCntl are FTP'd to the target z/OS system in binary mode with the z/OS attributes indicated in the example that follows.

The following example of command-line FTP shows the proper transfer to z/OS of data sets ITKO.LPARAGNT.CNTL.UNLOAD and ITKO.LPARAGNT.LOAD.UNLOAD. The z/OS file names can be changed for your data set naming conventions.

```
ftp> quote SITE LRECL=80 RECFM=FB BLKSIZE=3120 TRACKS PRI=40 SEC=5
200 SITE command was accepted
ftp> bin
200 Representation type is Image
ftp> put iTKOLPARAgentLoad 'ITKO.LPARAGNT.LOAD.UNLOAD'
200 Port request OK.
125 Storing data set ITKO.LPARAGNT.LOAD.UNLOAD
250 Transfer completed successfully.
ftp> quote SITE LRECL=80 RECFM=FB BLKSIZE=3120 TRACKS PRI=10 SEC=1
200 SITE command was accepted
ftp> put iTKOLPARAgentCntl 'ITKO.LPARAGNT.CNTL.UNLOAD'
200 Port request OK.
125 Storing data set ITKO.LPARAGNT.CNTL.UNLOAD
250 Transfer completed successfully.
```

Restore the PDSEs and PDS

The LPAR Agent load library PDSE and control (JCL) library PDS are restored with the TSO RECEIVE command, which typically runs in a batch job.

The following sample JCL restores the Extended Partitioned Data Sets (PDSE) and Partitioned Data Set (PDS) from the FTP'd data sets ITKO.LPARAGNT.LOAD and ITKO.LPARAGNT.CNTL on volume VOL001. The data set names can be changed for your site data set naming conventions.

```
//job
//*
//* Unload the LOAD Library with TSO RECEIVE
//*
//UNLOAD EXEC PGM=IKJEFT01,DYNAMNBR=10
//SYSTSPRT DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//SYSOUT DD SYSOUT=*
//SYSTSIN DD *
RECEIVE INDSNAME('ITKO.LPARAGNT.LOAD.UNLOAD')
   DATASET('ITKO.LPARAGNT.LOAD') VOLUME(VOL001)
RECEIVE INDSNAME('ITKO.LPARAGNT.CNTL.UNLOAD')
   DATASET('ITKO.LPARAGNT.CNTL') VOLUME(VOL001)
/*
```

Set up the LPAR Agent

The LPARAGNT member in ITKO.LPARAGNT.CNTL can be used as a model for the proc that runs the LPAR Agent as a started task. After the new proc member is created, it must be copied to a PROCLIB from where tasks can be started. Four JCL statements make up the proc:

PROC:

Statement that specifies the name of the started task.

EXEC:

Statement that specifies the program name and parameters.

STEPLIB:

The PDSE load library that contains the LPAR Agent load module.

config:

The DD statement that specifies a PDS member where the LPAR Agent configuration is defined.

The PARM specified in the EXEC statement defines environment variables. The PARM also specifies the DD name of the statement that points to the LPAR Agent configuration data member. A forward slash (‘/’) delimits the PARM data. Values before the slash specify run-time options and values after the slash specify the command-line arguments to the application. The environment variable is a run-time option and the LPAR Agent configuration data is a command-line argument.

Currently the only environment variable that the LPAR Agent recognizes is a time zone. The variable is optional and if not specified, defaults to UTC. This variable is used to calculate date and time values in the messages that the LPAR Agent produces. The valid values are the standard UNIX TZ strings, for example, EST5EDT, CST6CDT, MST7MDT, PST8PDT.

The following sample proc is in the LPARAGNT member of ITKO.LPARAGNT.CNTL:

```
//LPARAGNT    PROC

//AGENT       EXEC
PGM=LPARAGNT,PARM='ENVAR("TZ=CST6CDT")/"//DD:CONFIG"'

//STEPLIB     DD DISP=SHR,DSN=ITKO.LPARAGNT.LOAD

//CONFIG      DD DISP=SHR,DSN=ITKO.LPARAGNT.CNTL(CONFIG)

//SYSUDUMP    DD SYSOUT=*
```

Create the LPAR Agent Config Member

The configuration data that the DevTest LPAR Agent requires is specified as keyword/value pairs in plain text. Specify the PDS member in a DD statement in the proc that is used to start the LPAR Agent. Specify the DD name in the PARM value in the EXEC statement. The file is parsed as follows:

- A hash character ('#') is used as a comment indicator.
- Any blank lines or lines starting with a hash character are ignored.
- An equal sign separates the keywords and values.
- Any white space is trimmed from the beginning and ending of both the keywords and values.
- Keywords are not case-sensitive, while most values have the case preserved.
- A list exists of valid keywords that the LPAR Agent understands.

The keywords that can be used to configure the LPAR Agent are:

Name

Specifies the name of the LPAR Agent. This name is used for things like logging.

Default: DevTest LPAR Agent

Mode

Specifies whether the LPAR Agent is in **server** (keyword: Server) or **client** (keyword: Client) mode. In Server mode, the agent accepts connections from the DevTest registry on the port *PortClient*. This value must be the opposite of the **lisa.mainframe.bridge.mode** property in **lisa.properties**.

PortClient

When running in server mode, defines the value of the well-known port to which clients connect. This port number must match the value that the **lisa.mainframe.bridge.server.port** property in **lisa.properties** specifies.

Default: 3997

Server

When running in client mode, specifies the IP address of the server to which to connect. This IP address typically matches the IP address of the DevTest Server host running the registry.

ServerPort

When running in client mode, defines the port number that the DevTest Server registry uses. This port must match the value that the **lisa.mainframe.bridge.port** property in **lisa.properties** specifies.

PortAgent

Defines the value of the well-known port that CICS agents connect to. This port number must match the value that all CICS agents connecting to this LPAR Agent configure.

Default: 3998

InitialTrace

Specifies whether packet tracing is initially turned on.

Values:

- **True:** Tracing is turned on for all connections
- **False:** Tracing is turned off for all connections

Limits: These values are not case-sensitive.

Default: False

BufferSize

Defines the initial buffer size allocated. This value is typically set to twice the size of the largest data payload expected. The buffer size can be reallocated up to the configured maximum when packets of a larger size than this value are received.

Default: 65536

MaxBufferSize

Defines the maximum packet size to be accepted. A packet larger than this value is discarded and a message is logged. A value of 0 disables the size verification, and packets of any size from a client are accepted.

Default: 0

The following sample configuration file for running in **server** mode is in the CONFIGSV member of ITKO.LPARAGNT.CNTL:

```
#

# A sample server mode configuration file

#

Name             = DevTest LPAR Agent



#

# Mode can be "Client" or "Server"

#

Mode             = Server

PortClient       = 3997
```

```
#


#

# The port used as a server for the agent connections

#

PortAgent        = 3998


#

BufferSize       = 65536

MaxBufferSize    = 65536

#
```

The following sample configuration file for running in **client** mode is found in the CONFIGCL member of ITKO.LPARAGNT.CNTL:

```
#

# A sample client mode configuration file

#

Name              = DevTest LPAR Agent


#

# Mode can be "Client" or "Server"

#

Mode              = Client

Server            = 10.0.0.1

ServerPort        = 61617


#

# The port used as a server for the agent connections

#
```

```
PortAgent         = 3998


#

BufferSize        = 65536

MaxBufferSize     = 65536

#
```

LPAR Agent Storage Requirements

The DevTest LPAR agent runs as a standalone address space with a minimum REGION of 3 MB. A larger REGION may be required if the recorded images are excessively large or the arrival rate is high.

LPAR Agent Operation

Starting the DevTest LPAR Agent

In a typical operation, the LPAR Agent is set up as a started task and is started when the LPAR IPLs. The standard MVS START operator command (‘S’) can be used to start the LPAR Agent explicitly.

Stopping the DevTest LPAR Agent

The LPAR Agent responds to the standard MVS STOP operator command (‘P’). You can also stop the agent by issuing the “SHUTDOWN” message through the management socket connection or through the MVS MODIFY operator command (‘F’). The LPAR Agent terminates identically for each technique.

Monitoring and managing the LPAR Agent

The LPAR Agent can be monitored and managed to a limited extent through standard MVS operator commands.

To use standard MVS operator commands, issue the requests to the LPAR Agent using the MODIFY command (‘F’). The response is written to the console log. The supported LPAR Agent commands are:

?

Displays a list of available commands.

Usage: F LPARAGNT,?

L

Lists the current connections and the associated connection IDs.

U**sage:** F LPARAGNT,L

T

Toggles packet tracing.

Usage: F LPARAGNT,T <id> <state>

<id>

Defines the connection ID found in the "L" command.

<state>

(Optional) Specifies the state of the packet trace for a connection.

Issuing the T command without the state returns the tracing state for that connection.

Values: "ON" and "OFF"

SHUTDOWN

Shuts down the LPAR Agent.

Usage: F LPARAGNT,SHUTDOWN

The following output is an example of the **L** command:

```
F LPARAGNT,L

+ITKO9003 - Command accepted

+ID    Type IP              Port         In       Out  Connected

+3    AT   192.168.0.100   6891         1        0  12/15/11 09:38:44

+4    A    192.168.0.100   6892         1        0  12/15/11 09:38:45
```

The Type is **A** for "agent connection" or **C** for "client connection". If a **T** follows the **A** or **C**, it means that packet tracing is enabled for that connection. The IP address and Port number identify the source of the connection. In and Out show the number of packets that were received from and sent to the peer, respectively. The Connected date and time shows when the connection was created. The Connected date and time use the time zone value that is specified on the PARM of the proc EXEC statement.

The following output is an example of the T command:

```
F LPARAGNT,T 3

+ITKO9003 - Command accepted

+Tracing for ID 3 is ON

F LPARAGNT,T 3 OFF

+ITKO9003 - Command accepted

+Tracing for ID 3 is now OFF
```

LPAR Agent Messages

Messages are written to the operator console and to data sets in the JES log for the started task.

The following messages can be written to the operator console. These messages are informational only.

- ITKO9001 - LPAR Agent process initialized and waiting for connections
- ITKO9002 - The Operator command was too large
- ITKO9003 - The command was accepted
- ITKO9004 - The Start command was accepted
- ITKO9005 - The Stop command was accepted
- ITKO9006 - LPAR Agent process terminating

A sample JES log (DD SYS00001 of the started task):

```
2014:10:09 19:05:38 - Starting via operator command.

2014:10:09 19:05:38 - LPAR Agent started. Version: V800  Compiled: Feb
7 2014 13:08:00

2014:10:09 19:05:38 - Using configuration file: //DD:CONFIG

2014:10:09 19:05:38 - ---: Name            = CA DevTest LPAR Agent

2014:10:09 19:05:38 - ---: InitialTrace    = False

2014:10:09 19:05:38 - ---: Mode            = Server

2014:10:09 19:05:38 - ---: PortClient      = 4997

2014:10:09 19:05:38 - ---: PortAgent       = 3998

2014:10:09 19:05:38 - ---: BufferSize      = 65536

2014:10:09 19:05:38 - ---: MaxBufferSize   = 65536

2014:10:09 19:05:38 - ---: AppKASeconds    = 60

2014:10:09 19:05:38 - ---: AppKALimit      = 5

2014:10:09 19:05:38 - Configuration file processing successful.

2014:10:09 19:05:38 - LPAR Agent named CA DevTest LPAR Agent

2014:10:09 19:05:38 - Agent socket ready.

2014:10:09 19:05:38 - Client socket ready.

2014:10:09 19:05:38 - Memory for message buffers allocated.
```

```
2014:10:09 19:05:38 - Known interfaces: 192.86.32.105

2014:10:09 19:05:38 - Initial configuration complete.

2014:10:09 19:06:45 - ID: 2 - Accepted agent connection from
192.168.0.100.

2014:10:09 19:06:48 - ID: 3 - Accepted agent connection from
192.168.0.100.

2014:10:09 19:06:48 - ID: 4 - Accepted client connection from
192.168.0.101.
```

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 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 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.*