Integrated Capture Points Guide

Interaction Management 9.0.x

10/26/2023

Table of Contents

eServices Integrated Capture Points Guide

The integrated capture points are a feature of Interaction Server that provides a mechanism for capturing new interactions from external source systems, and for issuing various requests to existing interactions.

The capture points, with the exception of the Web Service Capture Point, also produce notifications on changes to interactions.

The following video is from the iWD documentation and helps to demonstrate the purpose of integrated capture points:

Link to video

JMS Capture Point

The integrated JMS (Java Message Service) Capture Point functionality is supported in Interaction Server starting in release 8.0.2. This functionality enables Interaction Server to capture requests to Interaction Server from a JMS-compliant message queue and to send Interaction Server replies and interaction event notifications to JMS-compliant message queues, in the form of XML documents. In iWD 7.6.1, the JMS Capture Adapter was a separate component, but its functionality was integrated into Interaction Server 8.0.2.

Kafka Capture Point

The integrated Kafka Capture Point functionality is supported in Interaction Server starting in release 8.5.305. This capture point is similar to JMS Capture Point as it works with Interaction Server requests, replies, and notifications in the form of XML documents. However, it uses specified Kafka topics to capture requests from and to produce replies and notifications to systems that use Kafka as a message bus. This capture point is compatible with the iWD XML file capture adapter by means of configuration options and transformation scripts.

File Capture Point

The integrated File Capture Point is supported in Interaction Server starting in release 8.0.21. This capture point is similar to JMS Capture Point as it works with Interaction Server requests, replies, and notifications in the form of XML documents. However, it uses specified file directories to capture requests and to produce replies and notifications in the form of XML files. This capture point is compatible with the iWD XML file capture adapter by means of configuration options and transformation scripts.

JMS, Kafka, and File Capture Points - operation modes

For the JMS, Kafka, and File Capture Points, XML representation of requests and notifications enables two modes of operation:

- Native mode, in which requests and interaction events notifications are consumed and generated, respectively, in Interaction Server native XML format.

- iWD (intelligent Workload Distribution) compatibility mode, in which supplied transformation scripts convert between iWD XML representation and native Interaction Server XML representation, supporting the full iWD API functionality (such as task creation, updating, holding, canceling, and various task state change notifications). For information on how to set up transformation, see Transformation.

DB Capture Point

The integrated Database Capture Point, introduced in Interaction Server 8.1.0, is functionally equivalent to Database Capture Adapter in iWD 8.0. It provides the ability to capture new interactions and to propagate updates for existing interactions in the form of user defined queries to external databases. It also provides a mechanism of propagating interaction event notifications to the external system in the form of user-defined queries. No transformation scripts are used with Database Capture Point.

WS Capture Point

The integrated Web Service Capture Point, introduced in Interaction Server 8.1.2, provides a web service interface for interaction-related requests such as `submit`, `stop`, `update`, `hold`, `resume`, and `get info`, as well as for ping requests. It fully supports web service definition of the Web Service Capture Point in iWD 8.0 when operating in iWD compatibility mode. It can be configured to work with either HTTP or Secure HTTP, in both native and iWD-compatible modes. No transformation scripts are used with Web Service Capture Point.

Configure the Integrated Capture Point

The procedures in this section are applicable to all types of Capture Point. Differences in configuration between the Capture Points are mentioned specifically in the procedures where necessary. This section contains the following procedures:

Creating the Capture Point application

Purpose

- The Capture Point functionality is built within Interaction Server 8.1, which means that there is no separate installation package for any type of Capture Points. An `Application` object for the Capture Point must be configured in Configuration Manager or Genesys Administrator, however. One `Application` must be configured for each instance of the Capture Point. Interaction Server supports multiple capture points.

Prerequisite

- Interaction Server must be installed as described earlier in this guide.

Start

1. Login to Configuration Manager or Genesys Administrator, and import the required Capture Point application template from `<Interaction Server installation location>\CapturePointTemplates\`.

   Configuration Server 8.0.3 supports the Capture Point application type. Earlier releases of Configuration Server should use the Third Party Server application type for capture points. Also, if a version of Configuration Server earlier than 8.0.3 is being used, you should modify the corresponding XML metadata file when you import metadata into Genesys Administrator. In the XML metadata file, replace `type="163"` with `type="23"`.

2. Create a new `Application` object based on the template you imported . The `CapturePointId` will be automatically set to the name of the Capture Point application as configured in Configuration Manager or Genesys Administrator. In iWD compatibility mode, it will also be saved as the `IWD_capturePointId` property in user data. When the Capture Point is later configured in iWD Manager, the Capture Point ID must be the same as the application name in order to ensure accurate events history reporting and accurate filtering. (The Capture Point *Name* can be anything).

   The name of the Capture Point `Application` must start with a letter, contain only alpha-numeric characters and underscores, and cannot be longer than 16 characters and cannot contain spaces.

3. Because the Capture Point is integrated with Interaction Server, the host and port information is taken from Interaction Server (which must be listed as a connection on the `Connections` tab). On the `Server Info` tab, you can enter the host and port of Interaction Server in the `Host` field, but the information will actually be taken from the connection to Interaction Server, not the information entered on this tab.

4. There is no installation package, so the Application object does not correspond to an installed component. Therefore, the information entered in the Start Info tab are not read. In order to save the Application object, the fields cannot be left blank, so you can enter any text in these fields.
5. If you are using a version of Configuration Server that does not support the Capture Point application type, then you must configure the following section and option for the Third Party Server Application object in order for Interaction Server to recognize it as a Capture Point:
   a. In the properties for the Third Party Server application, create a configuration option section called settings.
   b. In the settings section, add the option capture-point-type and set its value to:

- jms for the JMS Capture Point.
- kafka for the Kafka Capture Point.
- file for the File Capture Point.
- db for the Database Capture Point.
- webservice for the Web Service Capture Point.

If you do not create this option, the Third Party Server application will not be treated like a Capture Point.

For information about all configuration options for the Capture Point application and configuration options for Interaction Server that are related to the Capture Point functionality, refer to the *eServices 8.1 Reference Manual.*

- Add a connection to Interaction Server. Multiple Capture Point Application objects can connect to the same Interaction Server.
- Save the Application object.

End

Next Steps

- For the JMS Capture Point:
  - Verify your Java Configuration. See Java Configuration.
  - Set configuration options. There are four sample configurations that you can consult.
  - Configure a Capture Point Service in iWD Manager. See Configuring the Capture Point Service on this page.
- For the Kafka Capture Point:
  - Verify your Java Configuration. See Java Configuration.
  - Set configuration options. There is a sample configuration that you can consult.
  - Configure a Capture Point Service in iWD Manager. See Configuring the capture point service on this page.
- For the File Capture Point:
  - Verify your Java Configuration. See Java Configuration, if Groovy transformations are present (for example, in iWD compatibility mode).

    - Set configuration options.
    - Configure a Capture Point Service in iWD Manager. See Configuring the Capture Point Service on this page.
- For the Database Capture Point:
    - Set configuration options.
    - Configure a Capture Point Service in iWD Manager. See Configuring the Capture Point Service on this page.
    - Install a driver, configure ODBC and test the connection. See ODBC Drivers.
- For the Web Service Capture Point:
    - Set configuration options.
    - Configure a Capture Point Service in iWD Manager. See Configuring the Capture Point Service on this page.

Configuring the Capture Point Service

Purpose To create the service in iWD Manager. **Start**

1. Log into iWD Manager.

   For a detailed description of the iWD Manager interface, including logging in, the interface layout, and available functionality, refer to the *iWD Deployment Guide.*

2. In iWD Manager, select the Services navigation section.
3. Locate your Solution in the navigation tree. Expand the Services node in the navigation tree (if necessary), and click `New Service`.
4. From the templates drop down list,
    a. For iWD 8.1.0 or higher, select the Generic Capture Point service. The Capture Point ID must match the name of the Capture Point `Application` object that you configured in Creating the Capture Point application. Configure the remaining properties of the service. All other configuration for the Capture Point is done in Configuration Manager or Genesys Administrator by using configuration options.
    b. For iWD 8.0, select any Capture Point service. The service that you are creating will serve as a "dummy" Capture Point Service. The Capture Point ID must match the name of the Capture Point `Application` object that you configured in Creating the Capture Point application. The rest of the properties can be left at their default values, as they will not be used. All configuration for the Capture Point is done in Configuration Manager or Genesys Administrator by using configuration options.
3. When configuration is complete, click Save. Remember to deploy your changes in iWD Manager.

End

JMS Capture Point

The integrated JMS Capture Point is used to capture interactions from systems that use JMS as a message bus.

Prerequisites

The following prerequisites must be met in order to enable the JMS Capture Point functionality in Interaction Server:

- Licensing: Interaction Server will enable JMS Capture Point functionality only if the technical license (`iwd_jms_cp`) is present.
- Existing JMS-compliant message queue provider must be present.
- Java 8 is the minimum required version. Java 11 is recommended.
- JMS is supported by using Java Native Interface (JNI) and requires Java in order to work. If Java is not installed or not properly configured, JMS functionality will not be available. In addition to JMS API Java libraries, all required jar files for the specific provider need to be installed and accessible.

Outline

- Configuring
  - Start with the general procedure for creating a Capture Point Application object and a Capture Point Service.
  - Set configuration options for your particular environment.
  - Consult specific examples of configuring JMS Capture Points, listed in the next section.
- XML Representation—The integrated JMS Capture Point is capable of capturing interactions in the form of XML documents from JMS-compliant message queue providers. Consult the description of inbound and outbound XML messages.

Sample Configurations

This guide provides sample configurations for JMS Capture Point applications working with various JMS providers:

- OpenMQ
  - Setting up queues
  - Creating a Capture Point Application

JMS Capture Point Configuration Options

For JMS capture point configuration options, refer to eServices Options Reference Manual.

Endpoints

To enable endpoints functionality for the integrated Capture Point, you must add a tenant on the `Tenants` tab of the Capture Point `Application` and you must add a section called `endpoints` to the configuration options. You can add the `endpoints` section manually in Configuration Manager or by using Interaction Routing Designer (IRD) version 8.0.100.12 or later. The integrated Capture Point endpoints work in the same way as endpoints for media servers. Refer to the *eServices Reference Manual*, *Universal Routing Business Process User's Guide* and IRD Help for detailed descriptions.

OpenMQ—JMS Capture Point Queues

This page provides an example of setting up queues for the JMS Capture Point using the OpenMQ provider.

Setting up queues with Open Message Queue Administration Console

Start

1. Connect to the OpenMQ broker that is running.
2. Add the following queues using the `Add Broker Destination` dialog: `Inbound`, `Processed`, `Error`, and `Notification`.
3. For each queue that you have added, set `Max Number of Producers` and `Max Number of Active Consumers` to `Unlimited`.
4. Add a new Object Store and set the following JNDI Naming Service Properties:
   a. Set `java.naming.factory.initial` to `com.sun.jndi.fscontext.RefFSContextFactory`.
   b. Set `java.naming.provider.url` to `file:///D:/OpenMQExample`.

      This is the directory in which the `.bindings` file containing definitions will be saved.
3. Connect to the newly created object store.
4. Add a connection factory object using the `Add Connection Factory Object` dialog:
   a. Specify the lookup name, such as `ConnectionFactory`.
   b. Specify the `Factory Type` as `QueueConnectionFactory`.
   c. In the `Client Identification` tab, specify the `Default Username` and `Default Password` (for example, guest/guest).
4. Add destinations to the object store for all four queues that you defined previously:
   a. For the `Inbound` queue, specify the lookup name `inbound` and destination name `Inbound`.
   b. For the other queues, set the lookup names as `processed`, `error`, and `notification`.

      The lookup names can be different from the destination names.
3. After the above steps have been completed, the folder `D:/OpenMQExample/` contains the `.bindings` file with connection factory and queue definitions. Open the file, examine it for the presence of the defined queues and connection factory, and save it with File format set to `UNIX` so that it is possible to use it on UNIX operating systems.

End

Next Steps

- Create a Capture Point Application object in Configuration Manager.

OpenMQ—JMS Capture Point Application

This page provides an example to configuring a JMS Capture Point Application object when using the OpenMQ provider. This is a specific example of the more general configuration procedure titled "Configure Integrated CP."

Creating a Capture Point application in Configuration Manager (OpenMQ example)

Start

1. Create a Capture Point `Application` in the Configuration Manager named `CP_OpenMQ_solaris`.
2. On the `Options` tab create a section `settings`. In this section add the following options:

- `capture-point-type=jms`
- `inbound-queue-name=inbound` (the same as in the queue lookup name above)
- `error-queue-name=error`
- `processed-queue-name=processed`
- `notification-queue-name=notification`
- `xsl-inbound-transform-path=./iwd_scripts/iWD2IxnServerTransformer.groovy` (points to the default iWD Compatibility scripts)
- `xsl-outbound-transform-path=./iwd_scripts/IxnServer2iWDTransformer.groovy`
- `username=guest` (as configured in the connection factory)
- `password=guest`
- `jms-initial-context-factory=com.sun.jndi.fscontext.RefFSContextFactory`
- `jms-provider-url=file:///home/InteractionServer` (the path points to the folder where the `.bindings` file (in UNIX file format) is stored on the Interaction Server host)
- `jms-connection-factory-lookup-name=ConnectionFactory`

- On the `Connections` tab add the Interaction Server that will use this JMS Message queue.

End Next Steps

- Configure the Interaction Server options to load JVM and all of the required libraries.

OpenMQ—Interaction Server JVM

This page provides an example of configuring Interaction Server options to load JVM and all of the required libraries when using OpenMQ. See also the general description of configuring for Java.

Configuring Interaction Server to load JVM and required libraries (OpenMQ example)

Start

1. In the options of the Interaction Server to which the Capture Point `Application` is connected, create a section called `java-config` and add the option:

   ```
   jvm-path=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.181-7.b13.el7.x86_64/jre/lib/amd64/server/
   libjvm.so
   ```

   This is the full path to the `libjvm.so` (`jvm.dll` on Windows) on the host on which the Interaction Server is deployed.

2. Create a section called `jvm-options` and add the following option:

   ```
   -Djava.class.path=/home/OpenMQ_sol/mq/lib/imq.jar:/home/OpenMQ_sol/mq/lib/fscontext.jar:
       /home/OpenMQ_sol/mq/lib/jms.jar:./lib/ixn-java-aux.jar:./lib/groovy-
   all-2.4.21.jar:./lib/XmlTransformer/xercesImpl.jar:./lib/XmlTransformer/xsltc.jar.
   ```

   This option specifies the classpath to all of the Java archives that are necessary for JMS Capture Points on OpenMQ with iWD compatibility transformations to run. Note that the jar files `imq.jar`, `fscontext.jar`, and `jms.jar` are located in the Open MQ installation directory and are *not* supplied in the Interaction Server installation package.

3. Add the options `-Xoss1m` and `-Xss1m` to the `jvm-options` section. These options must have empty values.

End

TIBCO—JMS Capture Point Application

This example assumes the following:

- The host of the TIBCO message queue service is called **tibhost**.
- Queues called **inbound**, **error**, **notification**, and **processed** are defined.
- Both user name and password are **guest**.
- The connection factory is called **tibconnectionfact**.

Configuring a JMS Capture Point Application Object (TIBCO example)

1. On the **Options** tab, create a section called `settings`. In this section add the following options:
   - `capture-point-type=jms`
   - `inbound-queue-name=inbound` (the same as the queue name)
   - `error-queue-name=error`
   - `processed-queue-name=processed`
   - `notification-queue-name=notification`
   - `xsl-inbound-transform-path=./iwd_scripts/iWD2IxnServerTransformer.groovy` (points to the default iWD Compatibility scripts)
   - `xsl-outbound-transform-path=./iwd_scripts/IxnServer2iWDTransformer.groovy`
   - `username=guest`
   - `password=guest`
   - `jms-connection-factory-lookup-name=tibconnectionfact` (the name of the connection factory on TIBCO)
   - `jms-initial-context-factory=com.tibco.tibjms.naming.TibjmsInitialContextFactory`
   - `jms-provider-url=tibjmsnaming://tibhost:7222`
2. (Optional) In case the lookup of the Connection Factory over JNDI on Tibco EMS is protected by simple authentication, do these additional steps. In the **Options** tab, create a section called `jms-additional-context-attributes`. In this section add the following options:
   - `java.naming.security.principal=<username> // JNDI user`
   - `java.naming.security.credentials=<password> // JNDI password`
   - `java.naming.security.authentication=simple`

> **Important**
>
> <username> and <password> are the credentials used to authenticate on the JNDI server and may be different from those used to authenticate the connection to the JMS objects.

3. On the **Connections** tab, add the Interaction Server that will use this JMS Message queue.

Next Steps

- Configure the Interaction Server options that are required to load JVM and the necessary libraries.

TIBCO—Interaction Server JVM

This page provides an example of configuring Interaction Server options to load JVM and all of the required libraries when using TIBCO. See also the general description of configuring for Java. This example assumes the following:

- The host of the TIBCO message queue service is called `tibhost`.
- Queues called `inbound`, `error`, `notification`, and `processed` are defined.
- Both user name and password are `guest`.
- The connection factory is called `tibconnectionfact`.

Configuring Interaction Server to load JVM and the required libraries (TIBCO example)

Start

1. On the `Options` tab of the Interaction Server Application, create a section named `java-config` and add the option:

   ```
   jvm-path=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.181-7.b13.el7.x86_64/jre/lib/amd64/server/
   libjvm.so
   ```

   This is the full path to the `libjvm.so` (`jvm.dll` if the operating system is Windows) on the host on which the Interaction Server is deployed.

2. Create a section named `jvm-options` and add the following option:

   ```
   -Djava.class.path=./lib/ixn-java-aux.jar:./lib/groovy-all-2.4.21.jar:./lib/XmlTransformer/
   xercesImpl.jar:./lib/XmlTransformer/xsltc.jar:/opt/tibco/ems/6.0/lib/jms.jar:/opt/tibco/
   ems/6.0/lib/tibjms.jar.
   ```

   This option specifies the class path to all of the Java archives that are necessary for JMS Capture Points on TIBCO with iWD compatibility transformations to run. Note that the jar files `tibjms.jar` and `jms.jar` are located in the TIBCO installation directory and are *not* supplied in the Interaction Server installation package.

3. Add the options `-Xoss1m` and `-Xss1m` to the `jvm-options` section. These options must have empty values.

End

ActiveMQ—JMS Capture Point Queues

This page provides an example of how to set up queues for the JMS Capture Point using the ActiveMQ provider.

Setting up queues for the JMS Capture Point using the ActiveMQ provider

1. Configure Apache ActiveMQ, as described on the Apache website.
2. Create a **jndi.properties** file with the following content:

```
connectionFactoryNames = ConnectionFactory

queue.inbound = inx.inbound

queue.error = inx.error

queue.processed = inx.processed

queue.notification = inx.notification
```

   Use the ZIP Utility to pack the **jndi.properties** file in the **amq-jndi.jar** file.

ActiveMQ—JMS Capture Point Application

This page provides an example of how to configure a JMS Capture Point Application object when using ActiveMQ.

Creating a Capture Point application

1. On the **Application Options** tab, edit the **setting** section:

```
capture-point-type=jms

inbound-queue-name=inbound

error-queue-name=error

processed-queue-name=processed

notification-queue-name=notification

xsl-inbound-transform-path=./iwd_scripts/iWD2IxnServerTransformer.groovy (points to
 the default iWD Compatibility scripts)

xsl-outbound-transform-path=./iwd_scripts/IxnServer2iWDTransformer.groovy

username=(as configured in the connection factory)

password=(as configured in the connection factory)

jms-initial-context-factory=org.apache.activemq.jndi.ActiveMQInitialContextFactory

jms-provider-url=tcp://<activemq_host>:<activemq_port>

jms-connection-factory-lookup-name=ConnectionFactory
```

2. On the **Connections** tab, add the Interaction Server that will use this JMS Message queue.

ActiveMQ—Interaction Server JVM

This page provides an example of configuring Interaction Server options to load JVM and all of the required libraries when using ActiveMQ. See also the general description of the configuration requirements for Java.

Configuring Interaction Server to load JVM and required libraries

In the Interaction Server Application object, on the **Applications Options** tab, add the following jar files to the `-Djava.class.path` option in the **jvm-options** section:

1. **activemq-all-<version>.jar**
2. **amq-jndi.jar**
3. **log4j-api-<version>.jar** (optional, refer to the Note below)

The **activemq-all-<version>.jar** file is located in the ActiveMQ installation directory and not supplied with Interaction Server.

> ### Important
> Since version 5.17.0 **activemq-all** also requires log4j-api library, refer to Maven Repository: org.apache.activemq » activemq-all for information about log4j-api version. The **log4j-api-<version>.jar** file is not supplied with Interaction Server and should be downloaded from public repositories (For example, Maven Repository: org.apache.logging.log4j » log4j-api).

WebSphereMQ-JMS CP Queues

This page provides an example of setting up queues for the JMS Capture Point when using IBM WebSphere MQ.

Setting up queues using IBM WebSphere MQ Explorer

Start

1. Start WebSphere MQ Explorer. Find the Object tree in the Navigator window.
2. Right-click the `Queue Managers` node and select `New` to create a new Queue Manager. Follow the steps in the resulting Wizard, choosing a name (for example, `my_QManager`) and unique listening port.
3. As the Object tree is updated, find the `Queues` node under the new Queue Manager. Right-click this node and select `New > Local Queue.`
4. Create Local Queues named `mq_inbound, mq_notifications, mq_errors and mq_processed.` Select `Persistent` for the `Default Persistence` setting.
5. With the `Queues` node selected in the Object tree, right-click `mq_inbound` in the Content pane and select `Put Test Message.` Enter any text of your choice in the `Message data` field, then click `Put message.` This test message will wait in the queue until the Capture Point retrieves it.
6. In the Object tree, right-click the `JMS Administered Objects` node and select `Add Initial Context.` Choose `File system` for the JNDI namespace location and select the directory where the corresponding storage file will be created.
7. The new node for initial context now appears in the Object tree. Select it and verify that the `Connection Factories` and `Destinations` nodes appear under it. If necessary, right-click and use the context menu to connect to the `InitialContext` object make these nodes visible.
8. Right-click `Connection Factories` and select `New > Connection Factory.` Enter or select the following values:
    a. Sample name—`my_ConnFactory`
    b. Messaging provider—`WebSphere MQ`
    c. Transport—`MQ Client`
    d. Base queue manager and Broker queue manager (last screen)—The Queue Manager that you created in Step 2.
    e. Host name and port—Correct values for your environment
6. Right-click `Destinations` and select `New > Destination` and create four new Destinations that correspond to the queues that you created in Step 4:
    a. Type—`Queue`
    b. Names—`jms-inbound, jms-errors, jms-notifications,` and `jms-processed.`
    c. On the last screen, select the proper `Queue Manager` and `Queue` objects.
4. Find the file named `.bindings` at the location established in Step 6. It will be referred to later on the sample configuration.

End

Next Steps

Configure the JMS Capture Point Application object.

WebSphereMQ—JMS CP Application

This page provides an example to configuring a JMS Capture Point Application object when using WebSphere MQ.

This is a specific example of the more general configuration procedure.

Procedure: Configuring the JMS Capture Point application in Configuration Manager (WebSphere MQ example)

Start

1. On the `Options` tab create a section called `settings`. In this section add the following options:

- `capture-point-type=jms`
- `inbound-queue-name=inbound` (the same as the corresponding Destination name)
- `error-queue-name=jms-error`
- `processed-queue-name=jms-processed`
- `notification-queue-name=jms-notifications`
- `xsl-inbound-transform-path=./iwd_scripts/iWD2IxnServerTransformer.groovy` (points to the default iWD Compatibility scripts)
- `xsl-outbound-transform-path=./iwd_scripts/IxnServer2iWDTransformer.groovy`
- `jms-connection-factory-lookup-name=my_ConnFactory` (the name of the connection factory that you created in WebSphere MQ)
- `jms-initial-context-factory=com.sun.jndi.fscontext.RefFSContextFactory`
- `jms-provider-url=file:///home/InteractionServer` (the path points to the folder where the `.bindings` file—in UNIX file format—is stored on the Interaction Server host)
- On the `Connections` tab, add the Interaction Server which will use this JMS Message queue.

End

Next Steps

Configure the Interaction Server options to load JVM and the necessary libraries.

WebSphereMQ—Interaction Server JVM

This page provides an example of configuring Interaction Server options to load JVM and all of the required libraries when using WebSphere MQ. See also the general description of configuring for Java.

Configuring Interaction Server options to load JVM and all of the required libraries (WebSphere MQ example)

Start

1. On the `Options` tab of the Interaction Server Application, create a section named `java-config` and add the option:

   `jvm-path=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.181-7.b13.el7.x86_64/jre/lib/amd64/server/libjvm.so` (the full path to the `libjvm.so`, or `jvm.dll` if the operating system is Windows, on the host on which the Interaction Server is deployed).

2. Create a section named `jvm-options` and add the following option:

   ```
   -Djava.class.path=./lib/ixn-java-aux.jar:./lib/groovy-all-2.4.21.jar:./lib/XmlTransformer/xercesImpl.jar:./lib/XmlTransformer/xsltc.jar:/usr/location/jms/mq/com.ibm.mq.allclient.jar:/usr/location/jms/mq/providerutil.jar:/usr/location/jms/mq/fscontext.jar:/usr/location/jms/mq/jms.jar
   ```

   The example classpath contains list of jars required for IBM MQ 9.0.0. If you use another version of IBM MQ, refer to the corresponding documentation.

3. Add the options `-Xoss1m` and `-Xss1m` to the `jvm-options` section. These options must have empty values.

End

Note on the first of the three options in Step 2: This option specifies the class path to Java archives that are necessary for JMS Capture Points on WebSphere MQ with iWD compatibility transformations to run. Note that in the class-path the jar files `com.ibm.mq.jar`, `com.ibm.mqjms.jar`, `fscontext.jar`, and `jms.jar` are located in some user-defined location, and are not supplied in the Interaction Server installation package. These files are installed on your host by the WebSphere MQ installation, server or client, and are typically located in the subdirectory `/opt/mqm` on Linux and `C:\Program Files\IBM\MQ` on Windows.

The connection to WebSphere MQ requires more than just the four jar files listed in the class-path, as these jar files depend on other jar files in the same directory. Therefore, the class-path should refer to them at the same location where they were placed by WebSphere MQ installation. Or, if they were copied, all files contained in the `./lib` directory should be copied to the new location.

ActiveMQ—SSL for JMS CP

This page provides an example of how to enable SSL with the ActiveMQ provider.

Enabling SSL with the ActiveMQ provider

> **Important**
> ActiveMQ client jar of version 5.14.2 or later is needed.

1. Prepare the TLS certificates, as described in the Genesys Security Deployment Guide.
2. Copy cert.jks and truststore.jks into the following folder: <Apache ActiveMQ installation directory>/conf.
3. Open the file **activemq.xml** in the folder <Apache ActiveMQ installation directory>/conf and add the following lines:

```
        <transportConnectors>

        ...

        <transportConnectorname="ssl"uri="ssl://0.0.0.0:<ssl_port>?trace=true≠edClient
         Auth=true"/>

        ...

    </transportConnectors>

    <sslContext>
        <sslContextkeyStore="file:${activemq.base}/conf/cert.jks"

        keyStorePassword="YourKeyStorePassword"

        trustStore="file:${activemq.base}/conf/truststore.jks"

        trustStorePassword="YourTrustStorePassword"/>

    </sslContext>
```

4. Restart Active MQ to let it read new configuration.
5. In the Capture Point application options, in the **settings** section, edit the following options:
   - `jms-provider-url=ssl://<activemq_host>:<ssl_port>`
   - `jms-initial-context-factory=org.apache.activemq.jndi.ActiveMQSslInitialContextFactory`
6. In the Capture Point application, create a section **jms-additional-context-attributes** with following options:
   - `connection.ConnectionFactory.keyStore=<path to local keystore file>`
   - `connection.ConnectionFactory.keyStorePassword=<local keystore password>`

- `connection.ConnectionFactory.keyStoreType=jks`
- `connection.ConnectionFactory.trustStore=<path to local truststore file>`
- `connection.ConnectionFactory.trustStorePassword=<local keystore password>`
- `connection.ConnectionFactory.trustStoreType=jks`

7. To debug SSL, add the following option into the Interaction Server Application options, section **java-options**:
   ```
   -Djavax.net.debug=ssl:handshake,data,trustmanager,record
   ```

OpenMQ—SSL for JMS CP

This section provides an example of enabling SSL with the OpenMQ provider.

Outline

In general, configuration of an SSL connection consists of the following major steps:

1. Prepare the certificates.
2. Configure the JMS provider to operate in SSL mode.
3. Configure the options in Interaction Server's `jvm-options` section and add required JARs to the class path.
4. Configure the JMS Capture Point.

Configure Capture Point to use SSL (OpenMQ example)

Prerequisites This example assumes that an instance of Open MQ is configured and operating with a JMS Capture Point, without SSL.

Start The first several steps involve configuring the OpenMQ broker.

1. Generate a self-signed broker certificate:
    a. Run keytool to generate a key store (if one does not already exist) to generate a self-signed certificate:

    ```
    <OpenMQ installation dir>\mq\bin>imqkeytool
    ```

    b. Answer all the prompts and remember the chosen passwords. By default, the keystore will be called `keystore` and will be located in `<OpenMQ installation dir>\etc\mq`.

3. Add `ssljms` to active broker services:
    a. Locate the file `<OpenMQinstallation>\var\mq\instances\imqbroker\props\config.properties`.
    b. At the end of the file, add the following line: `imq.service.activelist=ssljms,admin,httpjms`
    c. Set the SSL port by adding the following line: `imq.ssljms.tls.port=1756`
    d. Restart the broker.

    The broker will prompt the user for a keystore password.

5. Update the connection factory properties: In the `.bindings` file, find the line `{Your connection factory lookup name}/RefAddr/44/Content=`

and change it to
`{Your connection factory lookup name}/RefAddr/44/Content=mqssl\://{your broker host}\:1756`
where 1756 is the same port as that set in the broker properties. This operation can be done using the OpenMQ Administration Console by selecting the corresponding connection factory and adding `mqssl://{your broker host}:1756` to the Message Server Address List properties on its `Connection Handing` tab.

The next steps involve configuring Interaction Server.

4. Export the broker certificate to a trust store:
   a. Export the broker certificate with the following command:

      ```
      keytool -export -alias imq -keystore keystore -file openmqbroker.cer
      ```

   b. Copy the `.cer` file to Interaction Server's host and import it to a local trust store:

      ```
      keytool -import -keystore truststore.jks -file openmqbroker.cer -alias
       openmqbroker
      ```

3. Add the following to the Interaction Server `jvm-options` section:

   ```
   -Djavax.net.ssl.trustStore= {Path to the local trust store}/truststore.jks
   -Djavax.net.ssl.trustStorePassword={your local trust store password}
   -Djavax.net.ssl.trustStoreType=jks
   ```

   For debugging purposes, you can also add the following option, which prints debug information to the console:
   `-Djavax.net.debug=ssl:handshake,data,trustmanager,record`

4. Finally, configure the JMS Capture Point by adding the following to the `jms-additional-context-attributes` section:

   ```
   java.naming.security.protocol=ssl java.naming.security.authentication=simple
   ```

End

It should be noted that in this example, the JNDI naming service used has all of the relevant context stored in a `.bindings` file and does not have any mechanism of authorization and authentication. With other JNDI services, the user accessing JNDI may have to provide a username and a password, which can be different from the JMS connection credentials. If this is the case, the JMS Connection credentials must be specified in the JMS Capture Point `settings` section as `username` and `password`, while the JNDI username and password must be specified in the `jms-additional-context-attributes` section as `java.naming.security.principal` and `java.naming.security.credentials` respectively.

TIBCO—JMS Capture Point and SSL connection

This page provides an example of setting up an SSL connection between TIBCO and Interaction Server for JMS Capture Point.

Prerequisites

- TIBCO EMS Community Edition 10.2
- Interaction Server 9.0.010.07 with OpenJDK 16

One-way TLS connection

This section describes a sample configuration for setting up a one-way TLS connection.

Preparing a server certificate

1. Run the OpenSSL command:

   ```
   "/C=RU/ST=SPb/L=SPb/CN=${your_tibcohost_full_name_aka_fqdn}" -newkey rsa:2048 -keyout
   tibcoserver.key.pem -out tibcoserver.pem
   ```

   In the above command as well as in the subsequent commands given in this document, the following representation is used:

   - C=<your country code>
   - ST=<your state or region code>
   - L=<your city or location code>

   You must also replace **${your_tibcohost_full_name_aka_fqdn}** with a fully qualified domain name (FQDN) of the machine where TIBCO EMS resides.

2. Provide a password, for example `tibcoserver`, if prompted.
3. When the OpenSSL command completes, two files are generated: **tibcoserver.pem** and **tibcoserver.key.pem**. Copy the files to the machine where TIBCO EMS resides and ensure that the TIBCO process has access to these files.

Configuring the TIBCO server

1. Create a new server configuration file under the **/bin** directory, for example **ssl.conf**.
2. Specify the path to the two files that you obtained using the OpenSSL command as explained in the **Preparing a server certificate** section. Additionally, specify the password that you previously provided, for example `tibcoserver`.

```
listen = ssl://7243
ssl_server_identity = /server_machine/local/path/to/tibcoserver.pem
ssl_server_key = /server_machine/local/path/to/tibcoserver.key.pem
ssl_password = tibcoserver

# uncomment text below if you need debug details
# console_trace=DEFAULT,+SSL,+SSL_DEBUG

# uncomment text below if you need debug details
# log_trace=DEFAULT,+SSL,+SSL_DEBUG

logfile = tibco.log
```

> **Important**
>
> If you are using other parameters in your regular configuration file, move them into **ssl.conf**.

3. If you have an insecure port or other ports configured on your server, specify them using the `listen` parameter, for example:

```
listen = ssl://7243,tcp://7222
```

Configuring an SSL connection factory

The TIBCO clients must obtain initial connection parameters from pre-configured connection factories. To create a new entry, use the following steps:

1. Locate the **factories.conf** file under the **/bin** directory.
2. Create a new entry, for example:

```
[SSLQueueConnectionFactory]
  type                    = queue
  url                     = ssl:/${your_tibcohost_full_name_aka_fqdn}:7243
  ssl_verify_host         = enable

  #uncomment line below if the hostname in URL is different from that in the
certificate
  #ssl_expected_hostname   = my.tibco.expected.url

  ssl_trusted             = /server_machine/local/path/to/tibcoserver.pem
```

Ensure that **${your_tibcohost_full_name_aka_fqdn}** is same as the hostname that you used during the certificate generation. Otherwise, Interaction Server will not validate the server certificate.

You can also use `ssl_expected_hostname`.

Creating queues on TIBCO

Create the following queues by editing the **queues.conf** file in the TIBCO directory:

- inbound
- error
- processed
- notification

Creating a user on TIBCO

Create a user genesys with the password `tibcoclient`.

Configuring Interaction Server

1. In the Interaction Server settings, locate the **jvm-config** section. The section contains the **jvm-path** option. Using this option, specify the full path to the local **jvm.dll** file. For example:

   ```
   jvm-path=C:\Program Files\OpenJDK\jdk-16.0.2\bin\server\jvm.dll
   ```

   > **Important**
   > We recommend that you use JDK 11 or a higher version.

2. Locate the **jvm-options** section. Append the path of TIBCO libraries to the value of **-Djava.class.path**. For example:

   ```
   -Djava.class.path=lib\samples-9.0.0.jar;lib\ixn-java-aux.jar;lib\groovy-
   all-2.4.21.jar;lib\XmlTransformer\xercesImpl.jar;lib\XmlTransformer\xsltc.jar;lib\
   KafkaEventLogger\kafka-clients-3.1.0.jar;lib\KafkaEventLogger\
   KafkaEventLogger.jar;lib\KafkaEventLogger\slf4j-api-1.7.36.jar;lib\KafkaEventLogger\
   avro-1.11.1.jar;lib\KafkaEventLogger\jackson-core-2.13.4.jar;lib\KafkaEventLogger\
   jackson-databind-2.13.4.2.jar;lib\KafkaEventLogger\jackson-
   annotations-2.13.4.jar;C:\3rd party jars\tibco\tibemsd_sec.jar;C:\3rd party jars\
   tibco\tibjms.jar;C:\3rd party jars\tibco\tibjmsadmin.jar;C:\3rd party jars\tibco\
   tibjmsapps.jar;C:\3rd party jars\tibco\tibrvjms.jar;C:\3rd party jars\tibco\
   jms-2.0.jar;
   ```

   Ensure that the classpath contains all .jar files that are supplied with Interaction Server in the folder **lib** in the installation directory.

Configure the Capture Point object

1. Under the Interaction Server installation directory, locate the folder **CapturePointTemplates** and its contents (files) with names starting with **JMSCapturePoint**. If you are using desktop Configuration Manager, import the Application Template using the .apd file. If you are using GAX, import the template using the .xml file.
2. Create a new application based on the imported template.

3. Copy the file **tibcoserver.pem** to the Interaction Server machine.
4. Set the following options:

```
[jms-additional-context-attributes]
com.tibco.tibjms.naming.ssl_debug_trace=true
com.tibco.tibjms.naming.ssl_trace=true
com.tibco.tibjms.naming.security_protocol=ssl
com.tibco.tibjms.naming.ssl_password=tibcoclient
com.tibco.tibjms.naming.ssl_trusted_certs=/IXN machine/local/path/to/tibcoserver.pem
java.naming.security.credentials=tibcoclient
java.naming.security.principal=genesys

[settings]
copy-original-properties-in-reply=false
error-queue-name=error
inbound-queue-name=inbound
include-ids-in-duplicate-error=false
jms-connection-factory-lookup-name=SSLQueueConnectionFactory
jms-initial-context-factory=com.tibco.tibjms.naming.TibjmsInitialContextFactory
jms-provider-url=ssl://${your_tibcohost_full_name_aka_fqdn}:7243
notification-queue-name=notification
processed-queue-name=processed
```

You can leave other options as they are.

Verifying your setup

1. Start TIBCO using the command:

```
tibemsd.exe -ssl_trace -ssl_debug_trace -config ssl.conf
```

Ensure that TIBCO reports **Server is ready**.

2. Open the Interaction Server's log file and check the following lines:

```
Std 23213 Capture point 'Tibco': started session on queue 'notification'
 Std 23213 Capture point 'Tibco: started session on queue 'inbound'
```

These two records indicate that Interaction Server successfully connected to the TIBCO EMS.

Two-way TLS connection (Mutual TLS)

This section describes a sample configuration for setting up a mutual TLS connection.

Prerequisite

A one-way TLS connection is already configured using the instructions given in the One-way TLS connection section.

Preparing a client certificate

1. Run the OpenSSL command:

```
openssl.exe req -x509 -days 365 -subj "/C=RU/ST=SPb/L=SPb/CN=
 ${your_tibcohost_full_name_aka_fqdn} " -newkey rsa:2048 -keyout tibcoclient.key.pem
 -out tibcoclient.pem
```

You must replace **${your_tibcohost_full_name_aka_fqdn}** with a fully qualified domain name (FQDN) of machine where TIBCO EMS resides.

2. Provide a password, for example `tibcoclient`, if prompted.
3. When the OpenSSL command completes, two files are generated: **tibcoclient.pem** and **tibcoclient.key.pem**. Copy **tibcoclient.pem** to the machine where TIBCO server resides and ensure that the TIBCO process has access to it.
4. Combine the private and public keys into a single file:

```
openssl pkcs12 -export -in tibcoclient.old.pem -inkey tibcoclient.old.key.pem -out
 tibcoclient.p12
```

5. Provide a password, for example `tibcoclient`, if prompted.
6. When the command completes, a single file **tibcoclient.p12** is generated. Copy this file to both the Interaction Server and TIBCO machines.

Updating the TIBCO EMS Server configuration file

Add the following lines to the **ssl.conf** file under the /bin directory:

```
ssl_server_trusted =/Tibco/machine/local/path/to/tibcoclient.pem
ssl_require_client_cert = enable
```

Updating the connection factory

Update **factories.conf** with the following line:

```
ssl_identity = /Tibco/machine/local/path/to/tibcoclient.p12
```

Updating the Capture Point object

Add the following options:

```
[jms-additional-context-attributes]
com.tibco.tibjms.naming.ssl_identity=/IXN machine/local/path/to/tibcoclient.p12
[settings]
username=genesys
password=tibcoclient
```

Verifying your setup

1. Start TIBCO using the command:

```
tibemsd.exe -ssl_trace -ssl_debug_trace -config ssl.conf
```

Ensure that TIBCO reports **Server is ready**.

2. Open the Interaction Server's log file and check the following lines:

```
Std 23213 Capture point 'Tibco': started session on queue 'notification'
 Std 23213 Capture point 'Tibco: started session on queue 'inbound'
```

These two records indicate that Interaction Server successfully connected to the TIBCO EMS with mutual TLS.

Kafka Capture Point

The integrated Kafka Capture Point is used to capture interactions from systems that use Kafka as a message bus.

Prerequisites

The following prerequisites must be met in order to enable the Kafka Capture Point functionality in Interaction Server:

- Licensing: Interaction Server will enable Kafka Capture Point functionality only if the technical license (`iwd_jms_cp`) is present.
- Existing Kafka cluster must be present. It should be running brokers of version “0.10.1.0” or newer.
- Java 8 is the minimum required version. Java 11 is recommended.
- Kafka is supported by using Java Kafka client and requires Java in order to work. If Java is not installed or not properly configured, Kafka functionality will not be available. All required jar files are provided in IP. These jars should be correctly specified in the Java Class Path option in Interaction Server. See Java configuration.

Outline

- Configuring
  - Start with the general procedure for creating a Capture Point Application object and a Capture Point Service.
  - Set configuration options for your particular environment.
  - Consult specific examples of configuring Kafka Capture Points, listed in the next section.
- XML Representation—The integrated Kafka Capture Point is capable of capturing interactions in the form of XML documents from the [Apache] Kafka messaging system. Consult the description of inbound and outbound XML messages.

Kafka Capture Point, just like JMS Capture Point, provides the following guarantees:

- "At least once" processing of the inbound messages including sending replies to them.
- "At most once" sending of the unsolicited notification events.

Kafka Capture Point supports secured communications via TLS protocols.

Sample Configuration

This guide provides sample configurations for Kafka Capture Point application.

Kafka Capture Point Configuration Options

For Kafka capture point configuration options, refer to eServices Options Reference Manual.

Kafka Capture Point Sample Configuration

The following is a sample configuration for the Kafka Capture Point application:

1. Install and start Zookeeper and Kafka Server as described on the Kafka website.
2. Create the following topics in Kafka, with 32 partitions each:
   - inbound
   - notification
   - error
   - processed
3. Configure the Kafka Capture Point application. On the `application` options tab, edit the `settings` section:
   - capture-point-type = kafka
   - kafka-server = host:port of kafka broker
   - inbound-topic-name = inbound
   - outbound-topic-name = notification
   - error-topic-name = error
   - processed-topic-name = processed

Kafka Capture Point - Interaction Server JVM

This page provides an example of configuring Interaction Server options to load Java virtual machine (JVM) and all of the required libraries when using Kafka. See also the general description of the configuration requirements for Java.

> **Important**
>
> Kafka Capture Point requires minimum JRE 8 to function. Java 11 is recommended.

Configuring Interaction Server to load JVM and required libraries

In the Interaction Server Application object, on the `Applications Options` tab, add the following jar files to the `-Djava.class.path` option in the **jvm-options** section:

- `<path to IXN dir>/lib/ixn-java-aux.jar`
- `<path to IXN dir>/lib/KafkaEventLogger/kafka-clients-3.0.0.jar`
- `<path to IXN dir>/lib/KafkaEventLogger/slf4j-api-1.7.32.jar`

If transformation is used in this capture point, then the following jars should be added to the option as well:

- `<path to IXN dir>/lib/XmlTransformer/xercesImpl.jar`
- `<path to IXN dir>/lib/XmlTransformer/xsltc.jar`
- `<path to IXN dir>/lib/groovy-all-2.4.21.jar`

All required jar files are provided in the IP.

> **Important**
>
> - On Windows, libraries are separated with a semi colon (;) and on Linux, with a colon (:).
> - Check your IXN Server installation for exact versions of third-party libraries provided because they may differ from ones mentioned in this documentation.

Kafka Capture Point - Topic Partitioning

Each Kafka topic contains one or more partitions. In a practical sense, a partition is a minimum processing unit. So, when working with Kafka, special attention should be paid to the topic partitioning because it influences message handling strategies.

Inbound topic partitioning

This section provides information on inbound topic partitioning.

Number of partitions

For each partition of the inbound topic, Kafka Capture Point has only the offset, index of the processed messages. This is the standard behavior for Kafka consumers. This is a simple and reliable way of working, but it limits the number of messages that can be processed in parallel. The parallel processing of messages is the straightforward way to speed up processing (up to some level). So, the inbound topic partitions number is the effective hard limit for the Kafka Capture Point receiving threads number and for performance of Kafka Capture Point. Each receiving thread is capable of processing messages from several partitions.

The number of the inbound topic partitions should be chosen with some reserve. The recommended number is 32 or 64.

Message partitioning

Another aspect of the partitioning is the way how the messages are spread across partitions. The usual requirement is to process requests related to an interaction sequentially, one at a time. Essentially, this means all such requests should be placed in one partition. This can be achieved by specifying the same partition key for these requests. Kafka uses partition keys to select partitions to put message to. Thus, specifying the same partition keys guarantees messages are placed in the same partition. Combining this with the processing of one request at a time by receiving thread of the Kafka Capture Point allows to satisfy the given requirement.

The natural candidates for the partition key are the value of `Interaction ID` generated by Interaction Server and the value of `External ID` supplied to Interaction Server from external system.

If the requirement is not needed or satisfied by other means, the partition key of the inbound messages can be left empty.

Notification topic partitioning

This section provides information on notification topic partitioning.

Number of partitions

The number of the notification topic partitions does not have any effect on Kafka Capture Point. So, it is not limited from the Kafka Capture Point side. This number should be chosen in accordance with the requirements and peculiarities of the consumers of this topic. But most probably, the recommendation for inbound topic partitions is applicable here as well.

The same applies to partitioning of error and processed topics.

Message partitioning

There are two kinds of messages that are sent to the notification topic:

- Replies to inbound requests
- Unsolicited notifications

The value of `Interaction ID` of the reply or the unsolicited notification is used as the partition key for the notification topic. If the value is not present in the reply, then the partition key supplied in the inbound request is reused for the reply.

Kafka Capture Point - Matching Requests and Replies

Unlike JMS, which supports `MessageID` and `CorrelationID` message properties for request and reply matching, Kafka does not provide any built-in means for this goal. Thus, Kafka Capture Point uses its own convention to allow matching outbound replies for the inbound requests.

Kafka Capture Point adds header `MessageID` to each outbound reply. Its value is the string representation of index of partition and offset of the inbound message separated with a dot. That is, if an inbound message is received from partition 6 with offset 20451, then the value of the `MessageID` header of the reply is `6.20451`.

Though `MessageID` uniquely identifies an inbound request, it is not always convenient. Thus, Kafka Capture Point reads the value of the first `CorrelationID` header of the inbound request and sends it back in the `CorrelationID` header of the reply. The name of the header used for reading correlation ID can be configured in the Kafka Capture Point option `correlation-id-header-key`. By default, it is `CorrelationID`.

> **Important**
>
> If the Kafka Capture Point option `copy-original-headers-in-reply` is set to `true`, the order of the `CorrelationID` header in the list of all headers copied from the inbound request to the outbound reply is preserved.

This approach effectively emulates the JMS message properties `MessageID` and `CorrelationID`.

This behavior applies to all outbound messages sent in reply to inbound requests including messages sent to the `notification`, `processed`, and `error` topics.

Kafka Capture Point - Debugging

The **ixn-java-aux.jar** file provides the means to debug Kafka Capture Point without Interaction Server, thus providing a simple and rapid sanity check of the Kafka environment.

You can use the Java class `com.genesyslab.eservices.interactionserver.capturepoints.CheckCPApp` to run Kafka Capture Point as a console Java application in the same way it is done in Interaction Server. It can produce/consume messages to/from Kafka. The only command-line argument is the path to the application settings file. For example:

On Windows:

```
java -cp <path to IXN dir>/lib/ixn-java-aux.jar;<path to IXN dir>/groovy_event_logger/lib/
KafkaEventLogger/kafka-clients-2.3.0.jar;<path to
IXNdir>/groovy_event_logger/lib/KafkaEventLogger/slf4j-api-1.7.26.jar;<path to IXN
dir>/transformation/groovy-all-2.4.15.jar;
com.genesyslab.eservices.interactionserver.capturepoints.CheckCPApp <path to application
settings>
```

On Linux:

```
java -cp <path to IXN dir>/lib/ixn-java-aux.jar:<path to IXN dir>/groovy_event_logger/lib/
KafkaEventLogger/kafka-clients-2.3.0.jar:<path to
IXNdir>/groovy_event_logger/lib/KafkaEventLogger/slf4j-api-1.7.26.jar:<path to IXN
dir>/transformation/groovy-all-2.4.15.jar:
com.genesyslab.eservices.interactionserver.capturepoints.CheckCPApp <path to application
settings>
```

The application settings file must be in JSON format. It follows the generic Genesys format: sections are on the first level and options are on the second level. For example:

```
{
   "check-cp-app": {
       "cp-type": "kafka",
       "cp-options-file": "<path to Kafka Capture Point settings file>",
       "received-messages-dir": "<path to a directory with inbound message files>",
       "commit-inbound": "true"
       "notifications-dir": "<path to a directory with notification files>",
   }
}
```

The following options are available:

- `cp-type` - (Mandatory) Must have `kafka` as the value always.
- `received-messages-dir` - A path to a directory where all the messages read from the inbound topic are stored in. Each message is stored in a separate file with an ordered number as a name. The default value is an empty string, which means messages won't be stored.
- `notifications-dir` - A path to a directory where unsolicited notifications are read from the 'to be sent to the notification' topic. Each file is considered to have one notification. All the characters of a file name, up to the last period symbol, are used as a partition key. The default value is an empty string, which means unsolicited notifications won't be sent.
- `commit-inbound` - If the value is set to `false`, Capture Point will never commit an inbound message;

instead, it will keep processing this message and continuously send out notifications, resulting in an infinite loop. The default value is `true`.

- `cp-options-file` - (Mandatory) A path to the Kafka Capture Point settings file. It follows the generic Genesys format: sections are on the first level and options are on the second level. For example:

```
{
   "settings": {
       "inbound-topic-name": "inbound",
       "processed-topic-name": "processed",
       "error-topic-name": "error",
       "notification-topic-name": "notification",
       "copy-original-properties-in-reply": "false",
       "consumer-receive-timeout": "10000",
       "kafka-server": "10.10.19.160:9092,10.10.19.161:9092,10.10.19.162:9092",
   },

   "consumer-options": {
      "max.poll.interval.ms": 1000,
      "max.poll.records": 20,
      "auto.offset.reset": "earliest",
   },

   "producer-options": {
      "retries": 10,
   }
}
```

File Capture Point

The integrated File Capture Point provides the ability to capture interactions from XML files that are found in a specified directory, and also provides compatibility with iWD file capture points.

- Configuring
  - Start with the general procedure for creating a Capture Point Application object.
  - Set configuration options for your particular environment.
- If you are using Groovy transformation scripts, configure Interaction Server to load the Java Virtual Machine.
- XML Representation: File Capture Point is capable of capturing interactions in the form of XML documents from a local or network directory. Consult the description of inbound and outbound XML messages.
- Read about File Capture Point's two modes of operation.
- Consult the rules for file naming.

File Capture Point Modes of Operation

The File Capture Point supports two modes of operation: Normal mode and iWD compatibility mode. The mode is specified by the configuration option `iwd-compatibility-mode`. See the eServices Reference Manual for complete information about this, and other, Capture Point configuration options.

Normal mode

In normal mode, the following four directories are defined and can be used:

- Inbound directory—The directory from which the interactions or tasks are captured.
- Error directory—If a file from the inbound directory is impossible to parse or otherwise process, and no corresponding interaction has been created, the original file is copied to this directory.
- Processed directory—If a file from the inbound directory has been successfully processed and its corresponding interaction has been created, the original file is copied into this directory.
- Notification directory—All solicited and unsolicited notifications, resulting from processing of interactions captured by this capture point will be written in the form of `.xml` files into this directory, subject to the notification filtering settings.

iWD Compatibility Mode

In iWD compatibility mode, the following directories, which extend the functions of the Notification directory, are added to the set of normal mode directories:

- Completed directory—If an interaction is placed into one of the Interaction Server queues belonging to the set of "completed" queues, as specified by the parameter `CompleteQueues` of the outbound transformer, an iWD notification `TaskInfo` produced by the outbound transformation will be saved in the form of an `.xml` file into this directory. The name of the `.xml` will follow the filename rules. See File Naming Rules.
- Rejected directory—If an interaction is placed into one of the Interaction Server queues belonging to the set of "rejected" queues, as specified by the parameter `RejectQueues` of the outbound transformer, an iWD notification `TaskInfo` produced by the outbound transformation will be saved in the form of an `.xml` file into this directory. The name of the `.xml` file will follow the filename rules.
- ErrorHeld directory—If an interaction is placed into one of the Interaction Server queues belonging to the set of "error held" queues, as specified by the parameter `ErrorHeldQueues` of the outbound transformer, an iWD notification `TaskInfo` produced by the outbound transformation will be saved in the form of an `.xml` file into this directory. The name of the `.xml` file will follow the filename rules.
- Canceled directory—If an interaction is placed into one of the Interaction Server queues belonging to the set of "canceled" queues, as specified by the parameter `CancelQueues` of the outbound transformer, an iWD notification `TaskInfo` produced by the outbound transformation will be saved in the form of an `.xml` file into this directory. The name of the `.xml` file will follow the filename rules.

In iWD compatibility mode, the error directory will contain a notification`.txt` file with the error description, along with the copy of the original `.xml` file that failed to be processed. The notification file will contain the error description and will be named consistently with the file that failed to be processed. Therefore, if the `.xml` file `FileName_1.xml` failed to be processed, the file name for the

error notification would be FileName_1.txt.

In iWD compatibility mode, the processed directory serves as a "captured" directory, as defined in the iWD XML file capture adapter. In other words, if an `.xml` file from the inbound directory has been successfully captured and submitted to a queue of the business process, the interaction contained in the original file is considered to be "captured" and the copy of the file is placed into the processed directory. iWD compatibility mode should always be used together with the supplied iWD compatibility groovy scripts. When iWD compatibility scripts (or any other groovy transformation scripts) are used by the File Capture Point, the Interaction Server must be configured to load Java Virtual Machine as described in Java Configuration, with the following JAR files correctly configured to be present in the class path: `groovy-all-2.4.21.jar, xercesImpl.jar, and xsltc.jar`.

> **Important**
>
> Check your IXN Server installation for exact versions of third-party libraries provided because they may differ from ones mentioned in this documentation.

File Capture Point File Naming Rules

This page describes the file naming rules that are followed for various directories that are used by the File Capture Point.

Error or Processed directory

When an interaction has been successfully captured from a file (for example, with the name `FileName.xml`), in the inbound directory, the file is copied, with its name preserved, into the processed directory. If Interaction Server cannot process a captured file, this file is copied into the error directory, with its name preserved. If a file with a desired filename exists in the destination directory, the filename resolution rule is used to find out the suffix to be appended to the desired filename. If the File Capture Point is operating in iWD compatibility mode, the name of the error notification file must match the name of the file that was written into the error directory (and include the same suffix if necessary).

All Other Directories

All notifications (messages written as files into directories other than the error and processed directories), both in normal mode and in iWD compatibility mode, can be named according to the notifications naming mode selected. The two modes available are `sequential` and `by-id`.

sequential naming

In this mode, the files in each destination directory are named `<counter>.xml`, where the `<counter>` is an integer, which is incremented for each new notification written. At startup, and when switching over, the File Capture Point checks all configured notification directories (except for inbound, error, and processed), finds out the current value of the `<counter>`, and increments it before each notification is written.

by-id naming

All notifications, both in normal mode and iWD compatibility mode, are written into their corresponding directories with the file names set to `<InteractionID>.xml`. If a file with a desired filename already exists in the destination directory, the File Capture Point finds the next available name for a notification for this Interaction ID, by sequentially checking the names matching the form of `<InteractionID>_<counter>.xml`, while the `<counter>` is incremented starting from 1.

File Capture Point Configuration Options

For File Capture Point configuration options, refer to eServices Options Reference Manual.

Important

- The Interaction Server `jvm-path` option (`java-config` section) is required for message transformation.
- The `jvm-options` section must be properly configured; most notably the `-Djava.class.path` option.
- If using iWD Compatibility Mode mode, Genesys suggests disabling the `schemaDocumentPath` option (`inbound-transformer-parameters` section) because the `iwd_messages.xsd` schema is extremely restrictive. To disable the option, you can either remove it or alter its name, e.g. to `//SchemaDocumentPath`. This applies to both the XML File Capture Point and the JMS Capture Point.

Database Capture Point

The integrated Database Capture Point provides the ability to capture interactions from databases, and also provides compatibility with the iWD Database Capture Adapter. The Database Capture Point provides the ability to create interactions based on a database query, and to update database records to propagate changes in interaction states or parameters.

The integrated Database Capture Point picks up updates for the interactions in the source database and applies these updates to the corresponding interactions. All relevant queries for selection and updates in the source database are configurable in the integrated Database Capture Point application settings.

Outline of Deployment

1. Configure the Capture Point
    1. Start with the general procedure for creating a Capture Point Application object and a Capture Point Service.
    2. Set configuration options for your particular environment.
2. Install and configure the required ODBC driver.
3. Read about the Configurable Queries available with this Capture Point.
4. Read about the Query Language.
5. Read about Error Handling.

Configurable Queries

The set of possible configurable queries in Database Capture Point includes the queries of iWD 8.0 Database Capture Adapter and introduces a number of new queries, corresponding to existing interaction events. iWD compatibility is achieved by configuring corresponding iWD-related queries and parameters.

The queries are written in SQL language, observing the semantics of the DBMS that you are using. When performing select queries, the columns should be named as standard interaction properties or user data keys (both case-sensitive). In update queries (using the interaction parameters or special keys) the interaction parameters and user data are case-sensitive as well.

When using parameters (such as "`externalid=<external id of the interaction>`"), write a question mark followed by the name of the parameter known to the interaction server in single quotes (such as "` externalid=?'ExternalId' `"). The question mark must be followed by the parameter name in single quotes, with no spaces. Do not use the curly apostrophe/single quote symbol (’ face="">); use the straight single quote (').

- Inbound Queries

- Notification Queries
- Source Update Queries
- There is also the following query that does not fit in the three previous categories:

Query parameter	Description
`startupQuerySql`	This optional query runs once, upon the Database Capture Point point establishing a connection to the database. It cannot take any parameters from Interaction Server.

Query Language

The Database Capture Point uses a particular query language, for which a reference listing is provided.

Error Handling

In situations where a capture or update query results in an error and cannot be executed, the values `ErrorCode` and `ErrorDescription` are provided to the corresponding error queries.

A returned `ErrorCode` can be equal to `0` for different `ErrorDescriptions`. This means that the error is not a protocol error and might not have a separate error code.

If an inbound (or source update) query results in an ODBC exception, the exception is reported in the logs, and the inbound (or source update) cycle pauses for the duration of the `inbound-exception-sleep-interval` parameter (for inbound queries) or the `updates-exception-sleep-interval` parameter (for source update queries). Both of these parameters are configuration options for the Database Capture Point.

Database Capture Point Configuration Options

For Database Capture Point configuration options, refer to eServices Options Reference Manual.

ODBC Drivers

> **Important**
> If you upgrade from older version and previously used DB Server to connect to database, refer to Configuring Interaction Server DAP.

For the Database Capture Point to work correctly, you must install and configure drivers. For all platforms and DBMS, you must:

- Obtain a client driver for the desired database.
- Using an ODBC manager, configure and test the connection to the database.

The procedures are similar for the same DBMS on different operating systems. In the procedures that follow, examples are used for User IDs, passwords, and names (such as `MY_ORAQ`).

ODBC Manager

The type of ODBC Manager to use depends on the operating system that you are using. Environments with alternative third-party drivers, database accelerators and ODBC Managers are not supported on compatibility issues. For more information, see Configuring Interaction Server DAP and Installing ODBC on Linux.

Notification Queries for Database Capture Point

Read the general description of configurable queries for the Database Capture Point. Notification queries are invoked upon the corresponding reporting events being generated. All notification queries are optional. If no query exists in the configuration, then no action is performed when the corresponding event occurs. Notification queries are queued (up to a `batch-size`, or up to `storing-timeout`, both configurable options) and executed in one transaction.

Notification Queries

Query parameter	Description	Reporting event (and condition)
`assignedUpdateSql`	The database query that updates the database to reflect that the associated interaction has been assigned to an agent. Values of all interaction properties and user data (except binary and kv-lists) of the corresponding interaction are available to this query. For example: `update inbound set status = 'assigned', assignedto=?'AssignedTo', assignedat=?'AssignedAt' where interactionid=?'InteractionId'`	`EventPartyAdded` (party is not `strategy`)
`completedUpdateSql`	The database query that updates the database to reflect that the associated interaction has been placed into a queue belonging to the `CompleteQueues` set specified in the `iwd-parameters` section of the configuration options (if the section and property are configured). Values of all interaction properties and user data (except binary and kv-lists) of the corresponding interaction are available to this query. For example: `update inbound set status = 'completed' where interactionid=?'InteractionId'`	`EventPlacedInQueue` (queue in `CompleteQueues`)
`canceledUpdateSql`	The database query that updates the database to reflect that the associated interaction has been placed into a queue belonging to the `CancelQueues` set specified	`EventPlacedInQueue` (queue in `CancelQueues`)

Query parameter	Description	Reporting event (and condition)
	in `iwd-parameters` section of the configuration options (if the section and property are configured). Values of all interaction properties and user data (except binary and kv-lists) of the corresponding interaction are available to this query. For example: `update inbound set status = 'canceled' where interactionid=?'InteractionId'`	
`heldUpdateSql`	The database query that updates the corresponding database record to reflect that the associated interaction has been put on hold. Values of all interaction properties and user data (except binary and kv-lists) of the corresponding interaction are available to this query. For example: `update inbound set status = 'held' where interactionid=?'InteractionId'`	`EventHeld`
`queuedUpdateSql`	The database query that updates the database to reflect that the associated interaction has been placed into any queue not belonging to the sets of iWD queues specified in the `iwd-parameters` section of the configuration options (such as `CancelQueues`, `CompleteQueues`, and so on). Values of all interaction properties and user data (except binary and kv-lists) of the corresponding interaction are available to this query. For example: `update inbound set status = 'queued', queue=?'Queue' where interactionid=?'InteractionId'`	`EventPlacedInQueue` (queue not in any iWD queues)
`errorHeldUpdateSql`	The database query that updates the database to reflect that the associated interaction has been placed into a queue belonging to the `ErrorHeldQueues` set specified in the `iwd-parameters` section of the configuration options (if the section and property are configured). Values of all interaction properties and	`EventPlacedInQueue` (queue in `ErrorHeldQueues`)

Query parameter	Description	Reporting event (and condition)
	user data (except binary and kv-lists) of the corresponding interaction are available to this query. For example: `update inbound set status = 'errorheld' where interactionid=?'InteractionId'`	
`rejectedUpdateSql`	The database query that updates the database to reflect that the associated interaction has been placed into a queue belonging to the `RejectQueues` set specified in the `iwd-parameters` section of the configuration options (if the section and property are configured). Values of all interaction properties and user data (except binary and kv-lists) of the corresponding interaction are available to this query. For example: `update inbound set status = 'rejected' where interactionid=?'InteractionId'`	`EventPlacedInQueue` (queue in `RejectQueues`)
`restartedUpdateSql`	The database query that updates the database to reflect that the associated interaction has been placed in the `RestartQueues` set specified in the `iwd-parameters` section of the settings (if the section and property are configured). Values of all interaction properties and user data (except binary and kv-lists) of the corresponding interaction are available to this query. For example: `update inbound set status = 'restarted' where interactionid=?'InteractionId'`	`EventPlacedInQueue` (queue in `RestartQueues`)
`stoppedUpdateSql`	The database query that updates the database to reflect that the associated interaction has been stopped in Interaction Server. Values of all interaction properties and user data (except binary and kv-lists) of the corresponding interaction are available to this query. For example: `update inbound set status = 'stopped' where interactionid=?'InteractionId'`	`EventProcessingStopped`

Query parameter	Description	Reporting event (and condition)
routeRequestedUpdateSql	The query statement that updates the database to reflect that the associated interaction has been sent to a router. Values of all interaction properties and user data (except binary and kv-lists) of the corresponding interaction are available to this query. For example: `update inbound set status = 'routing' where interactionid=?'InteractionId'`	EventPartyAdded (party is strategy)
updatedUpdateSql	The query statement that updates the database to reflect that the associated interaction has been updated in Interaction Server by some other entity (not this Database Capture Point). Values of all interaction properties and user data (except binary and kv-lists) of the corresponding interaction are available to this query. For example: `update inbound set priority=?'Priority' where interactionid=?'InteractionId'`	EventPropertiesChanged
resumedUpdateSql	The query statement that updates the corresponding database record to reflect that the associated interaction has been resumed from a hold. Values of all interaction properties and user data (except binary and kv-lists) of the corresponding interaction are available to this query. For example: `update inbound set status = 'resumed' where interactionid=?'InteractionId'`	EventResumed

Inbound Queries for Database Capture Point

Read the general description of configurable queries for the Database Capture Point. The three inbound queries are listed below. All three are required.

Inbound Queries

Query parameter	Description
`captureQuerySql`	The database query that returns the result set in which each row will be captured as an interaction by Interaction Server. If a column name does not belong to the predefined interaction properties' names, its value will be attached to the user data of the interaction with a key corresponding to the column name. For example: `select externalid "ExternalId", stamp "ReceivedAt", tenantid "TenantId", priority "Priority", status "Status" from inbound where status='new'`
`capturedUpdateSql`	The database query that updates the corresponding database record to reflect that certain data has been successfully captured as an interaction by Interaction Server. Besides the values available from the corresponding capture query, the `InteractionId` value is available to this query if it has not been provided in the result set of the corresponding capture query. For example: `update inbound set interactionid=?'InteractionId', status='submitted' where externalid=?'ExternalId'`
`errorUpdateSql`	The database query that updates the corresponding database record to reflect that the associated interaction has not been captured by Interaction Server. Besides the values available from the corresponding capture query, additional values `ErrorCode` (integer) and `ErrorDescription` (string up to 256 characters) are available to this query. For example: `update inbound set status='error', errorcode=?'ErrorCode', errordescr=?'ErrorDescription' where externalid=?'ExternalId'`

Source Update Queries for Database Capture Point

Read the general description of configurable queries for the Database Capture Point. If `sourceUpdateQuerySql` is specified, the other two queries are required to be configured and correct. If no query exists in the configuration, then no action is performed when the corresponding event occurs.

Source Update Queries

Query parameter	Description
Query parameter	Description
`sourceUpdateQuerySql`	The database query that fetches a set of rows, where each row represents an update request. Each such update request may contain one or more columns that represent interaction properties. The name of the column represents the name of the interaction property and the value is the new value of that interaction property. Each row of the result set must contain either `"InteractionId'` or `"ExternalId'`. If both `"InteractionId'` and `"ExternalId'` are contained in a row, the value of `"InteractionId'` will be used to access the interaction, and the value of `"ExternalId'` will be treated as one of the interaction properties to update. For example: `select interactionid "InteractionId", stamp "SomeTime", priority "Priority" from updates where status='new'`
`sourceUpdatedUpdateSql`	The database update (or delete) query that will execute against a special table in the source database to mark a particular update as having been processed. For example: `update updates set status='applied' where interactionid=?'InteractionId'`
`sourceErrorUpdateSql`	This update is executed when there is an error executing an update request (the one that is fetched by `sourceUpdateQuerySql`). Besides the values available from the corresponding capture query, additional values `"ErrorCode'` (integer) and `"ErrorDescription'` (string up to 256 characters) are available to this query. For example: `update updates set status='error', errorcode=?'ErrorCode', errordescr=?'ErrorDescription' where interactionid=?'InteractionId'`

Query Language for Database Capture Point

Interaction Properties

Setting and Getting Interaction Properties and their Data

Interaction property	Can be provided in submit	Can be updated by source update query	Input data type	Output data type
InteractionId	Y	N	Varchar	Varchar(256)
ExternalId	Y	Y	Varchar	Varchar(256)
ParentID	Y	Y	Varchar	Varchar(256)
MediaType	Y	N	Varchar	Varchar(256)
InteractionType	Y	N	Varchar	Varchar(256)
InteractionSubtype	Y	N	Varchar	Varchar(256)
TenantId	Y	N	Varchar or Int	Int
Queue	Y	Y	Varchar	Varchar(256)
Workbin	Y	N	Varchar	Varchar(256)
WorkbinAgentId	Y	N	Varchar	Varchar(256)
WorkbinPlaceId	Y	N	Varchar	Varchar(256)
WorkbinAgentGroupId	Y	N	Varchar	Varchar(256)
WorkbinPlaceGroupId	Y	N	Varchar	Varchar(256)
IsOnline	Y	N	Varchar or Int	Int
ReceivedAt	Y	N	Datetime	Varchar(256)
SubmittedBy	N	N	Not applicable	Varchar(256)
State	N	N	Not applicable	Int
IsLocked	N	N	Not applicable	Int
SubmittedAt	N	N	Not applicable	Varchar(256)
DeliveredAt	N	N	Not applicable	Varchar(256)
SubmittedToRouterAt	N	N	Not applicable	Varchar(256)
PlacedInQueueAt	N	N	Not applicable	Varchar(256)
MovedToQueueAt	N	N	Not applicable	Varchar(256)
AbandonedAt	N	N	Not applicable	Varchar(256)
IsHeld	N	N	Not applicable	Int
HeldAt	N	N	Not applicable	Varchar(256)

Interaction property	Can be provided in submit	Can be updated by source update query	Input data type	Output data type
AssignedAt	N	N	Not applicable	Varchar(256)
AssignedTo	N	N	Not applicable	Varchar(256)
CompletedAt	N	N	Not applicable	Varchar(256)

Special Column Names and Data Keys

Refer to the following table for special column names or data keys.

Special Column Names & Data Keys

Special column names or data keys	Can be provided in submit	Can be updated by source update query	Input data type	Output data type
Hold	Y	Y (but should not)	Int or Varchar	
ErrorCode	Y (but should not)	Y (but should not)	Not applicable	Int
ErrorDescription	Y (but should not)	Y (but should not)	Not applicable	Varchar(256)
EventTime	N	N	Not applicable	Varchar(256) available to notification queries only
ActorType	N	N	Not applicable	Int
ActorMediaServerId	N	N	Not applicable	Varchar(256)
ActorStrategyId	N	N	Not applicable	Varchar(256)
ActorRouterId	N	N	Not applicable	Varchar(256)
ActorTenantId	N	N	Not applicable	Int
ActorPlaceId	N	N	Not applicable	Varchar(256)
ActorAgentId	N	N	Not applicable	Varchar(256)
ReasonSystemName	N	N	Not applicable	Varchar(256)
ReasonDescription	N	N	Not applicable	Varchar(256)
Operation	N	N	Not applicable	Int
ItxServerName	N	N	Not applicable	Varchar(256)
ItxServerDBID	N	N	Not applicable	Int
TenantsNames	N	N	Not applicable	Varchar(256)
TenantsDBIDs	N	N	Not applicable	Varchar(256)
ReportingEventSequenceNumber	N	N	Not applicable	Varchar(256), available to notification queries only

User Data

All other column names not corresponding to interaction properties, special column names, or data keys are interpreted as user data keys.

Data Types

The tables above refer to data types Datetime, Int, and Varchar. More formally, these data types are defined for each DBMS as follows:

Data Types Defined Per DBMS

DBMS	Int Types	Datetime Types	Varchar Types
Oracle	int, integer, smallint	date, timestamp	varchar2
DB2	numeric, decimal, smallint	timestamp	varchar, char
MSSQL	numeric, decimal, smallint, money, smallmoney	datetime, datetime2	varchar

Important

The values in columns of Datetime type are converted and attached to their corresponding keys as strings, therefore their values are available as Varchar type for output parameters. If they need to be inserted into actual datetime columns, either casting or conversion should be performed.

Web Service Capture Point

The web service Capture Point provides a web service interface for interaction-related requests such as `submit`, `stop`, `update`, `hold`, `resume`, and `get info`, as well as for ping requests. It can operate in either of the following two modes:

- iWD compatibility mode, exposing the functionality of the Web Service Capture Point that is described in the intelligent Workload Distribution Deployment Guide.
- Native mode, with a more general set of requests as compared to iWD compatibility mode

Common Aspects

Service URL

The Web Service Capture Point service URL can be easily obtained from the Interaction Server startup log. Look for the following message and simply copy the URL:

```
11:17:58.814 Trc 23323 Capture point 'WSCapturePoint' will set endpoint:
'http://zoolander.us.int.genesyslab.com:10080
/Genesys/Interaction/WSCP_812_zoo/WebServiceCapturePoint'
```

You can also construct the URL using the template provided by the Web Service Capture Point application option`soap-endpoint`, whose default value is: `<Protocol>://<ServerName>:<ServerPort>/Genesys/Interaction/<CapturePointName>/WebServiceCapturePoint` where

- Protocol is HTTP or HTTPS, as specified in the `protocol` option.
- Server Name is either specified in the `soap-hostname` option or is equal to the name of Interaction Server's host.
- Port is the port of the Web Service Capture Point Application object.
- CapturePointName is the name of the Application object.

This template can be changed, but generally it contains the four parts just listed. Note that none of the parameters are mandatory and the entire endpoint can be simply specified in its final form, which may be preferable in some cases.

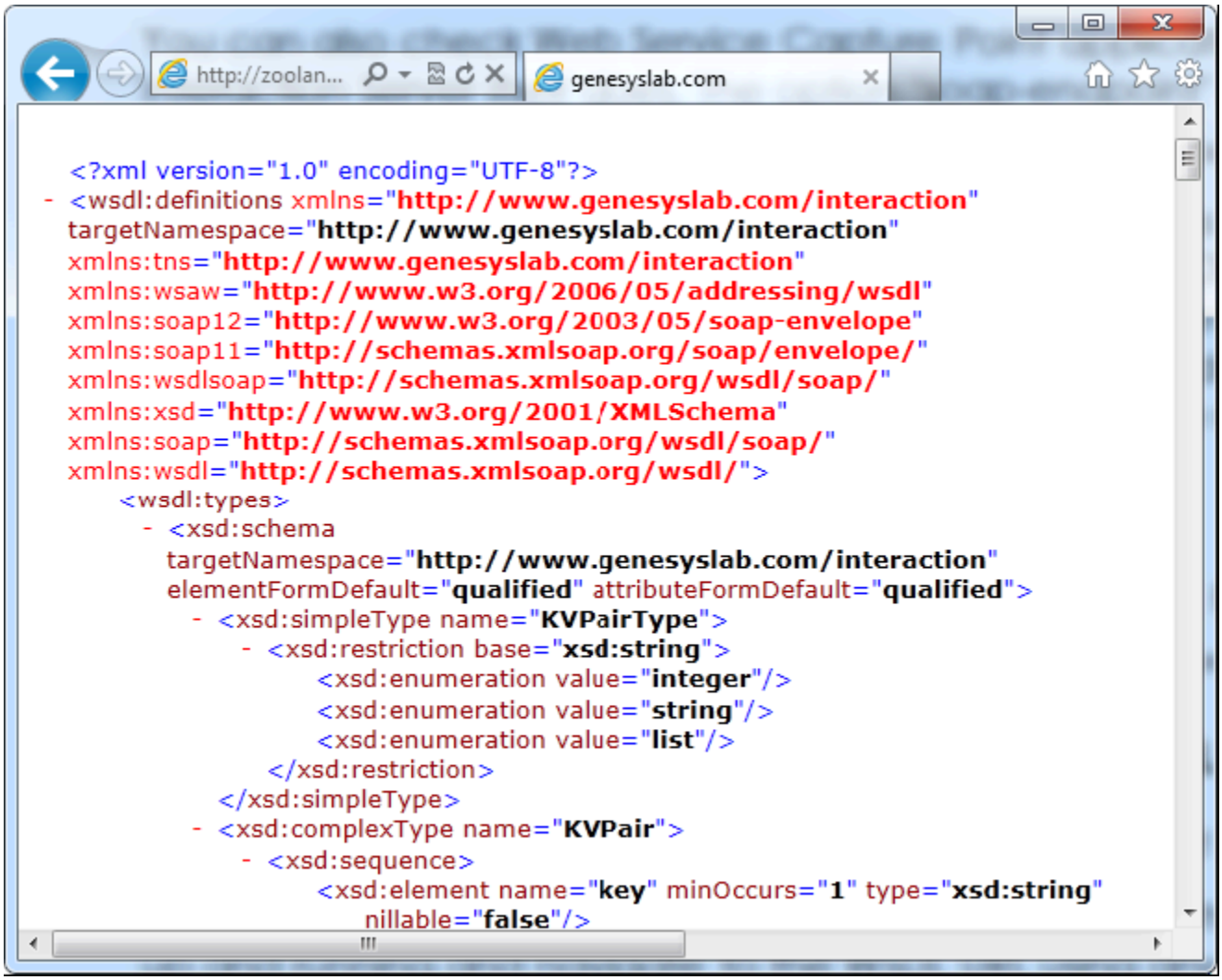
WSDL URL

The WSDL URL is the service URL with `?wsdl` appended; for example:

```
http://zoolander.us.int.genesyslab.com:10080/Genesys/Interaction/WSCP_812_zoo/
WebServiceCapturePoint?wsdl
```

Checking Connectivity and Inspecting WSDL

Once you get the service URL, you can use it in different tools to generate a Web Service client. To check that the service is up and running and to inspect the service WSDL, first ensure that Interaction Server is running, then navigate to the WSDL URL using any web browser. The following figure shows WSDL in Internet Explorer.



WSDL

This confirms that you have access to the WSCP service. You can inspect the WSDL or save it to a file to later use. Saving the WSDL is not required since most tools can simply access the WSDL URL directly, as long as Interaction Server is running.

Generating a Client

The following tools were used to generate WSCP clients for this document:

- Visual Studio 2010
- JAX-WS 2.2
- Apache CXF
- Apache Axis2

This document provides the following examples of generating a client:

- .NET client
- JAX-WS
- Apache CXF
  - Java client
  - Javascript client
- Apache Axis2/Java

Web Service Capture Point Client Over Secure HTTP

This section provides an example of configuring a Web Service Capture Point, generating and importing certificates, and using .NET and Java clients over Secure HTTP. OpenSSL version 1.0.0g or better is assumed to be installed. This example configuration assumes the presence of a server host (`zoolander.us.int.genesyslab.com` in the example) and a client host (`clienthost.us.int.genesyslab.com` in the example). The server host has an Interaction Server with a Web Service Capture Point named`WSCP_812_zoo` connected to it.

Server Certificate

The server certificate is used for server authentication (by the client) and ensures that server can be trusted. The Web Service Capture Point requires a server certificate to support SSL. You must generate a server certificate and put it into the client's trusted certificates store.

Client Certificate

A client certificate is required for mutual SSL authentication. If the Web Service Capture Point is configured for server authentication only, the client certificate is not required. This guide provides examples of

- Generating a client certificate for a .NET client.
- Generating a client certificate for a Java client.

Web Service Capture Point Configuration

In a Web Service Capture Point application, named, for example, `WSCP_812_zoo`, set the following options:

- `server-key-file=<Path to wscpserver.pem>\wscpserver.pem`
- `password=<'PEM pass phrase' for wscpserver.pem>`
- `protocol=https`
- `require-client-authentication=true`
- `cacert-file=<Path to wscp_clients.pem>\wscp_clients.pem`

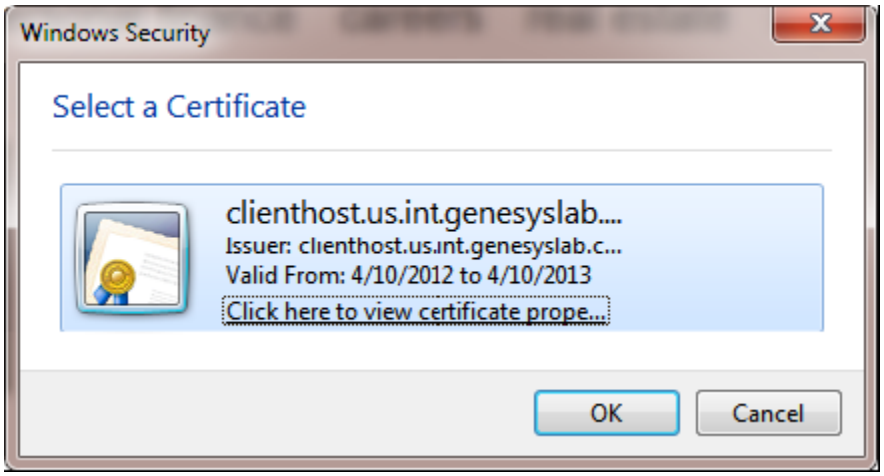
Do not change any other options. You must restart Interaction Server for these option values to take effect. If client authentication is not required, set the option `require-client-authentication` to `false` and omit all procedures relevant to generation and manipulation of client certificates (Client Certificate for Browser and .NET Client and Client Certificate for Java Client).

WSDL over HTTPS in the Browser

Assuming the client host has the server certificate in the trusted certificates, and the client certificate in personal certificate, you can request the WSDL from the client host by entering the URL that you obtained in Service URL:

```
https://zoolander.us.int.genesyslab.com:10080/Genesys/Interaction/WSCP_812_zoo
/WebServiceCapturePoint?wsdl
```

The browser then prompts the user to select a certificate, as shown below.



Select a Certificate

Select the imported certificate and click `OK`. The contents of the WSDL file should display in the browser.

Client Modifications

Once a client has been configured, certain modifications are required:

- For .NET clients
- For Java clients

Messaging

This section presents details of requests and responses, as follows:

- Native Mode
  - Requests
  - Sample Responses
- iWD-Compatible Mode
  - Requests
  - Sample Responses

> **Important**
> On Windows, libraries are separated with a semi colon (;) and on Linux, with a colon (:).

Web Service Capture Point Configuration Options

For Web Service Capture Point configuration options, refer to eServices Options Reference Manual.

Web Service Capture Point Native Mode

When operating in native mode, the Web Service Capture Point defines and supports the following capabilities:

- Submit an interaction to a queue or a workbin
- Hold an interaction, either by `InteractionId` or by `ExternalId`
- Resume an interaction, either by `InteractionId` or by `ExternalId`
- Stop an interaction, either by `InteractionId` or by `ExternalId`
- Update an interaction, either by `InteractionId` or by `ExternalId`; this includes:
  - Move to a different queue
  - Update interaction properties
  - Delete interaction properties
- Get Info on an interaction, either by `InteractionId` or by `ExternalId`
- Ping (for heartbeat monitoring and to obtain health monitoring information)

Web Service Capture Point iWD Compatibility Mode

When operating in iWD Compatibility Mode, the Web Service Capture Point is functionally equivalent to iWD Web Service Capture Point 8.0.

Supported Requests

iWD Compatibility Mode supports the following capabilities:

- Create a task
- Cancel a task, either by `InteractionId` or by `CaptureId`
- Hold a task, either by `InteractionId` or by `CaptureId`
- Resume a task, either by `InteractionId` or by `CaptureId`
- Update a task, either by `InteractionId` or by `CaptureId`
- Get task data, either by `InteractionId` or by `CaptureId`
- Restart a task, either by `InteractionId` or by `CaptureId`
- Complete a task, either by `InteractionId` or by `CaptureId`
- Ping (for heartbeat monitoring)

Ignored Elements

The following elements of iWD compatible requests have limited or no mapping in the corresponding Interaction Server requests and therefore are ignored by Interaction Server:

- The element `actor` is not mapped.
- The element `reason` is mapped to `ReasonSystemName` in the requests `holdTaskByCaptureId`, `resumeTaskByCaptureId`, `holdTaskByTaskId`, and `resumeTaskByTaskId`. This element is ignored in all other iWD requests.
- The value of the element `mediaType` in the request `createTask` is ignored. You can specify a Genesys-compatible media type in the element `data` using the key `MediaType`, as shown in this example.
- The element `hold` in `restartTaskByCaptureId` and `restartTaskByTaskId` is ignored.
- The element `completeDateTime` in `completeTaskByCaptureId` and `completeTaskByTaskId` is ignored, as Interaction Server has an interaction attribute `CompletedAt` which is set automatically when the interaction (task) is placed into a completed queue.

iWD WSDL Modification

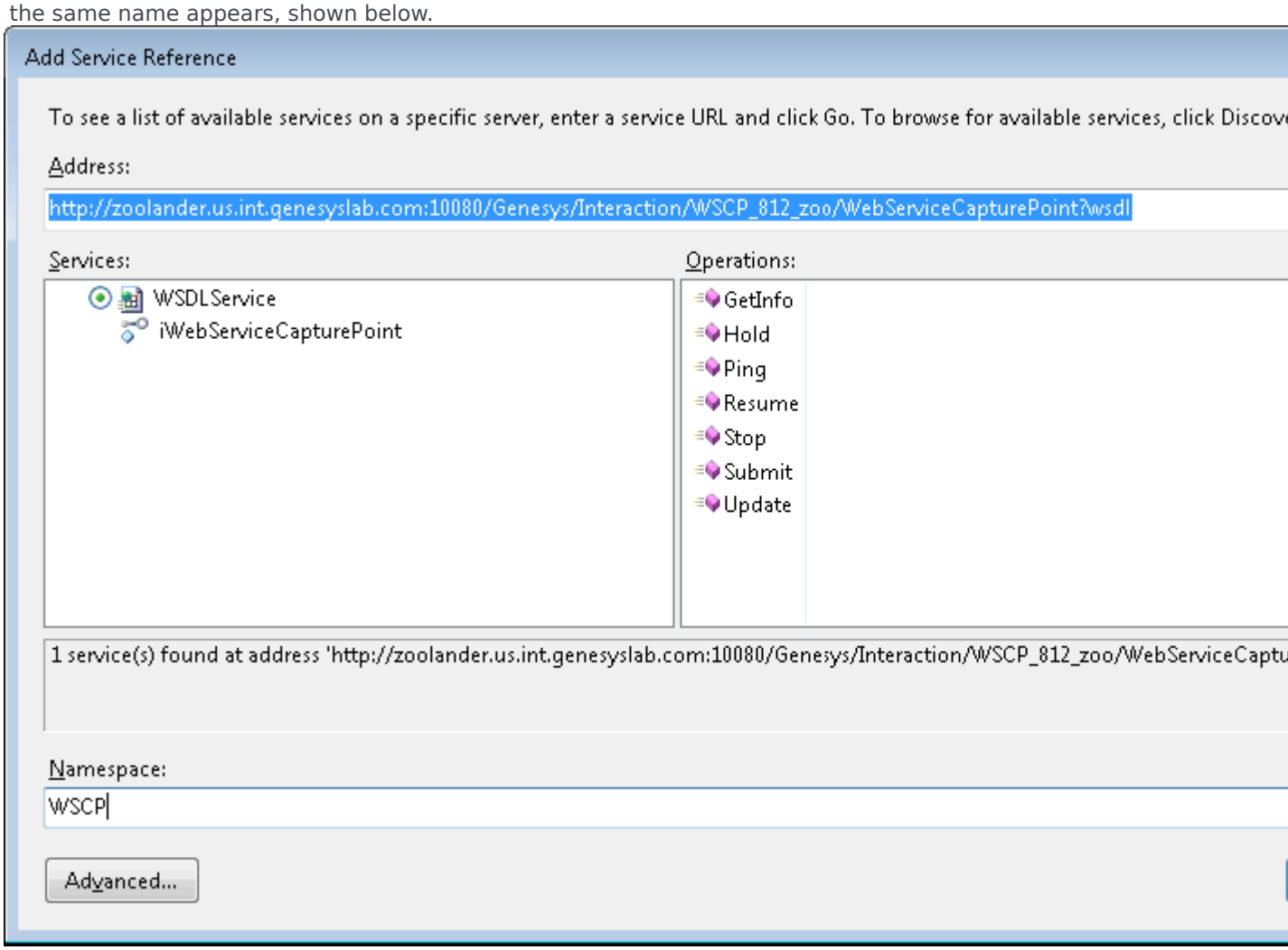
The original WSDL document from the Web Service Capture Adapter of iWD has been modified to permit all elements of the DateTime type be `nillable`, and to allow zero occurrence (`minOccurs="0"`). This change should not affect existing clients of iWD Web Service Capture Adapter.

Web Services Capture Point—Generate a .NET Client

This page provides an example of generating a .NET Client. See the list of tools used to generate clients in this document.

Start

1. Open Visual Studio 2010 and create a C# Win32 console application.
2. In Solution Explorer right-click `References` and choose `Add Service Reference.` The dialog box of the same name appears, shown below.



Add Service Reference

3. Enter the WSDL URL of the Web Service Capture Point.

4. Enter the service namespace (for example, `WSCP`):
5. Click Go.

   Provided Interaction Server is running and the WSDL URL is specified correctly, `WebServiceCapturePoint` should appear in the `Services` list.
6. Click `OK` to generate the service client.
7. To test the service, open the `Program.cs` file and insert the following code in the `main` method:

```
WSCP.iWebServiceCapturePointClient client = new WSCP.iWebServiceCapturePointClient();
// This is an optional step to reconfigure the client to use different endpoint.
// It's usually done using configuration setting for the application
//client.Endpoint.Address = new System.ServiceModel.EndpointAddress(
//    "http://localhost/Genesys/Interaction/MyCP/WebServiceCapturePoint");
// Create a key-value list of extensions and specify the signature,
// so we can recognize the request in Interaction Server log
var extension = new WSCP.KVList();
extension.Add(new WSCP.KVPair() { key = "signature",
value = new WSCP.KVPairValue() { ValueString = ".Net WSCP test client" } });
// We expect ping info back in Ping response
WSCP.KVList userdata = null;
WSCP.KVList pinginfo = null;
try
{
    // Ping the server and get some statistics back
    client.Ping(out userdata, out pinginfo, ref extension);
    Console.Out.WriteLine(trace_list(pinginfo));
}
catch (FaultException<FaultMessage> ex)
{
    // process WSCP specific error code
  Console.Out.WriteLine("Error {0}: {1}",
ex.Detail.ErrorCode, ex.Detail.ErrorDescription);
}
catch (Exception ex)
{
    Console.Out.WriteLine(ex.ToString());
}
```

8. Add the method `trace_list` to your program to output the server response:

```
static string trace_list(WSCP.KVList list, string indent = "")
{
    StringBuilder result = new StringBuilder();
    list.ForEach((item) =>
    {
        result.Append(indent);
        result.Append(item.key);
        if(null != item.value.ValueString)
        {
            result.Append(" [string] = ");
            result.Append(item.value.ValueString);
            result.Append('\n');
        }
        else if (null != item.value.ValueList)
        {
            result.Append(" [list] = \n");
            result.Append(trace_list(item.value.ValueList, indent + "    "));
        }
        else
        {
```

```
                result.Append(" [int] = ");
                result.Append(item.value.ValueInt.ToString());
                result.Append('\n');
            }
        });
        return result.ToString();
    }
```

This simple test prints Interaction Server statistics into a console window. You can then discover the service methods using autocompletion and the object browser.

End

Generate Service Proxy with wsimport

Start

To generate a Web Service Capture Point service proxy for Java, use the `wsimport` utility, which is included in JDK:

```
wsimport -d <output directory> <WSDL URL>
```

For example:

```
wsimport -d c:\Temp\MyJSClient
http://zoolander.us.int.genesyslab.com:10080/Genesys/Interaction/WSCP_812_zoo/
WebServiceCapturePoint?wsdl
```

The tool generates a set of files for the proxy.

Create a simple Java console application to ping the service:

```
import java.net.URL;
import javax.xml.namespace.QName;
import javax.xml.ws.Holder;
import com.genesyslab.interaction.*;
public class Test {

  public static void main(String[] args) throws Exception {

    WebServiceCapturePoint service = new WebServiceCapturePoint(new
URL("http://zoolander.us.int.genesyslab.com:10080/Genesys/Interaction/WSCP_812_zoo/
WebServiceCapturePoint/?WSDL"),new
QName("http://www.genesyslab.com/interaction", "WebServiceCapturePoint"));
    IWebServiceCapturePoint cp = service.getIWebServiceCapturePointHttpBinding();

    KVPairValue val = new KVPairValue();

    val.setValueString("I am coming from JAXWS client");

    KVPair pair = new KVPair();

    pair.setKey("Source");
    pair.setValue(val);

    KVList extList = new KVList();

    extList.getKvitem().add(pair);

    Holder<KVList> extension = new Holder<KVList>(extList);

    Holder<String> eventTime = new Holder<String>();
    Holder<KVList> userData = new Holder<KVList>();
    Holder<KVList> pingInfo = new Holder<KVList>();

    cp.ping(extension, eventTime, userData, pingInfo);

    System.out.println("Ping response time:" + eventTime.value);
    printKVList("PingInfo", pingInfo.value);
    printKVList("UserData", userData.value);
```

```
    printKVList("Extension", extension.value);
  }

  public static void printKVList(String name, KVList kvList) {
    printKVList(name, kvList, "");
  }

  private static void printKVList(String name, KVList kvList, String shift) {
    if (null == kvList) {
      System.out.println(shift + name + "[KVList]=null");

    } else {

      System.out.println(shift + name + "[KVList]=");

      for (KVPair pair : kvList.getKvitem()) {
        KVPairValue value = pair.getValue();

      if (value.getValueInt() != null) {
         System.out.println(shift + "\t" + pair.getKey() + "[int]="
                + value.getValueInt());
      } else if (null != value.getValueList()) {
         printKVList(pair.getKey(), value.getValueList(), shift
               + "\t");
      } else {
         System.out.println(shift + "\t" + pair.getKey()
               + "[string]=" + value.getValueString());
        }
      }
    }
  }
}
```

End

Apache CXF—Java Client

To generate a Web Service Capture Point service proxy for Java use the `wsdl2java` tool:

```
wsdl2java -frontend jaxws21 -d <output directory> <WSDL URL>
```

For example:

```
wsdl2java -d c:\Temp\MyJSClient
http://zoolander.us.int.genesyslab.com:10080/Genesys/Interaction/WSCP_812_zoo/
WebServiceCapturePoint?wsdl
```

The tool generates a set of files for the proxy.

Create a simple Java console application to ping the service:

```
import java.net.URL;
import javax.xml.ws.Holder;
import com.genesyslab.interaction.*;

public class Test {
    public static void main(String[] args) throws Exception {


      WebServiceCapturePoint service = new WebServiceCapturePoint(new
URL("http://zoolander.us.int.genesyslab.com:10080/Genesys/Interaction/WSCP_812_zoo/
WebServiceCapturePoint/?WSDL"));

      IWebServiceCapturePoint cp = service.getIWebServiceCapturePointHttpBinding();

      KVPairValue val = new KVPairValue();

      val.setValueString("I am coming from CXF client");

      KVPair pair = new KVPair();

      pair.setKey("Source");
      pair.setValue(val);

      KVList extList = new KVList();

      extList.getKvitem().add(pair);

      Holder<KVList> extension = new Holder<KVList>(extList);

      Holder<String> eventTime = new Holder<String>();
      Holder<KVList> userData = new Holder<KVList>();
      Holder<KVList> pingInfo = new Holder<KVList>();

      cp.ping(extension, eventTime, userData, pingInfo);

      System.out.println("Ping response time:" + eventTime.value);
      printKVList("PingInfo", pingInfo.value);
      printKVList("UserData", userData.value);
      printKVList("Extension", extension.value);
    }
    public static void printKVList(String name, KVList kvList) {
      printKVList(name, kvList, "");
```

```
    }
    private static void printKVList(String name, KVList kvList, String shift) {
      if (null == kvList) {
         System.out.println(shift + name + "[KVList]=null");
      } else {
         System.out.println(shift + name + "[KVList]=");

         for (KVPair pair : kvList.getKvitem()) {
            KVPairValue value = pair.getValue();

            if (value.getValueInt() != null) {
               System.out.println(shift + "\t" + pair.getKey() + "[int]="
                     + value.getValueInt());
            } else if (null != value.getValueList()) {
               printKVList(pair.getKey(), value.getValueList(), shift
                     + "\t");
            } else {
               System.out.println(shift + "\t" + pair.getKey()
                     + "[string]=" + value.getValueString());
            }
         }
      }
   }
}
```

Apache CXF—Javascript Client

This page provides an example of generating a Javascript client using the Apache CXF`wsdl2js` tool. See the list of tools used to generate clients in this document. You can also generate a Java client in Apache CXF.

To generate a Javascript client using the Apache CXF`wsdl2js` tool:

```
wsdl2js -d <output directory> <WSDL URL>
```

For example:

```
wsdl2js -d c:\Temp\MyJSClient
http://zoolander.us.int.genesyslab.com:10080/Genesys/Interaction/WSCP_812_zoo/
WebServiceCapturePoint?wsdl
```

The tool generates a single file that contains a proxy that can send requests to the service and receive replies asynchronously. You must also include the `cxf-util.js` file, which is part of Apache CXF.

The sample below does not require anything beyond HTML and Javascript (`wscp.js` is the file generated by `wsdl2js`):

```
<html>
<head>
<script type="text/javascript" src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.1/
jquery.min.js"></script>
<script type="text/javascript" src="cxf-utils.js"></script>
<script type="text/javascript" src="wscp.js"></script>

<script language="JavaScript" type="text/javascript">

var gCounter = 0;

function print_list(list, indent)
{
    var r = '';

    for(var i=0; i < list._kvitem.length; ++i)
    {
        r += indent;

        var pair = list._kvitem[i];
        r += pair._key;
        if( pair._value._ValueString )
        {
            r += " [str] = '";
            r += pair._value._ValueString;
            r += "'";
            r += "<br>";
        }
        else if( pair._value._ValueInt )
        {
            r += " [int] = ";
            r += pair._value._ValueInt;
            r += "<br>";
        }
```

```
        else
        {
            r += " [list] = ";
            if( pair._value._ValueList )
            {
                r += "<br>";
                r += print_list(pair._value._ValueList, indent + "....");
            }
            else
            {
                r += " EMPTY";
                r += "<br>";
            }
        }
    }
    return r;
}
function test()
{
    var svc = new _iWebServiceCapturePoint();
    svc.url = 'http://zoolander.us.int.genesyslab.com:10080/Genesys/Interaction/WSCP_812_zoo/
WebServiceCapturePoint';

    var extension = new _KVList();
    var items = new Array();

    var signature = new _KVPair();
    signature.setKey("signature");
    var signature_value = new _KVPairValue();
    signature_value.setValueString("JavaScript client generated with CXF");
    signature.setValue(signature_value);

    var counter = new _KVPair();
    counter.setKey("Request count");
    var counter_value = new _KVPairValue();
    counter_value.setValueInt(++gCounter);
    counter.setValue(counter_value);
    items.push(signature);
    items.push(counter);
    extension.setKvitem(items);

    svc.Ping(
        function(response)
        {
            var r = "Response timestamp: " + response.getEventTime() + ", ping info:<br>";
            r += print_list(response.getPingInfo(), "");
            $("#response_text").html(r);
        },
        function(status, statusText)
        {
            $("#response_text").html("Response failed: (" + status + ") " + statusText);
        },
        extension
    );
}

</script>
</head>
<body>
<p>Press the button to call the service...</p>
<p><input value="Ping the service" type="button" onclick="test()"/></p>
<p><div id="response_text"></div></p>
</body>
```

```
</html>
```

Generate Service Proxy with Axis2

This page provides an example of generating a Web Service Capture Point service proxy using the Axis2 plug-in. See the list of tools used to generate clients in this document.

The following sample demonstrates how to send a Ping request and to print out the contents of the PingResponse.

```
import com.genesyslab.www.interaction.WebServiceCapturePointStub.*;
import com.genesyslab.www.interaction.*;

public class TestWSCP {
    public static void main(String[] args) {
        try {
            WebServiceCapturePointStub serviceStub = new WebServiceCapturePointStub(

        "http://zoolander.us.int.genesyslab.com:10080/Genesys/Interaction/WSCP_812_zoo/
WebServiceCapturePoint");
            Ping ping = new Ping();

            KVList ext = new KVList();

            // create a string kv pair
            KVPair strPair = new KVPair();
            KVPairValue value = new KVPairValue();
            strPair.setKey("Source");
            value.setValueString("I am coming from axis2 client");
            strPair.setValue(value);

            // add this pair to the extension
            ext.addKvitem(strPair);

            // set extension
            ping.setExtension(ext);

            PingResponse response = serviceStub.Ping(ping);

            System.out.println("Ping response time:" + response.getEventTime());
            printKVList("PingInfo", response.getPingInfo());
            printKVList("UserData", response.getUserData());
            printKVList("Extension", response.getExtension());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public static void printKVList(String name, KVList kvList) {
        printKVList(name, kvList, "");
    }
        private static void printKVList(String name, KVList kvList, String shift) {
            if (null == kvList) {
            System.out.println(shift + name + "[KVList]=null");
        } else {
            System.out.println(shift + name + "[KVList]=");
            for (KVPair pair : kvList.getKvitem()) {
                KVPairValue value = pair.getValue();
            if (null != value.getValueList()) {
                printKVList(pair.getKey(), value.getValueList(), shift
```

```
                    + "\t");
            } else if (null != value.getValueString()) {
                System.out.println(shift + "\t" + pair.getKey()
                    + "[string]=" + value.getValueString());
            } else {
            System.out.println(shift + "\t" + pair.getKey() + "[int]="
                + value.getValueInt());
}
            }
        }
    }
}
```

Web Service Capture Point Client Over Secure HTTP

These pages provide an example of configuring a Web Service Capture Point, generating and importing certificates, and using .NET and Java clients over Secure HTTP. OpenSSL version 1.0.0g or better is assumed to be installed.

This example configuration assumes the presence of a server host (`zoolander.us.int.genesyslab.com` in the example) and a client host (`clienthost.us.int.genesyslab.com` in the example). The server host has an Interaction Server with a Web Service Capture Point named `WSCP_812_zoo` connected to it.

Server Certificate

The server certificate is used for server authentication (by the client) and ensures that server can be trusted. The Web Service Capture Point requires a server certificate to support SSL.

This page provides an example of generating a server certificate and putting it in the client's trusted certificates store.

Generate a server certificate

First generate a server certificate, along with a private key:

```
openssl req -x509 -days 365 -subj "/C=US/ST=California/L=Daly City/CN
=zoolander.us.int.genesyslab.com" -newkey rsa:2048 -keyout wscpserver.pem -out wscpserver.pem
```

The output file `wscpserver.pem` contains a private key along with a certificate. During the private key generation, the user is prompted for a password, which will be required later. The user will be asked to come up with a *PEM pass phrase,* which will be later used in the WSCP configuration, along with the generated `.pem` file. The server certificate can also be a self-signed certificate or a certificate signed by any Certificate Authority (CA). The certificate generated for the server must be imported or copied into the client's trusted certificates store. Use the procedure and tools appropriate for your platform.

The private key should **never** be copied or given to anyone. It should be password protected (encoded) and should be accessible to the server only. The client is given only the certificate (public key) to put into the trusted certificates store.

The following is a procedure for putting server certificates into client's trusted certificates store for Windows, using the `openssl` utility.

Put server certificate in client's store

Start

1. Convert the generated certificate to DER format:

   ```
   openssl x509 -outform der -in wscpserver.pem -out wscpserver.cer
   ```

   The output file `wscpserver.cer` contains a public server certificate, which will be added to the trusted certificates of the client using the Web Service Capture Point.

2. Import the generated .CER server certificate into the trusted certificates store (for browser and .NET client):

   a. Start Microsoft Management Console.

   b. On the `File` menu, select `Add or Remove Snap-ins`.

   c. Choose `Certificates`, then click `Add`.

d. When prompted, choose `Computer account` and `Local Computer`.

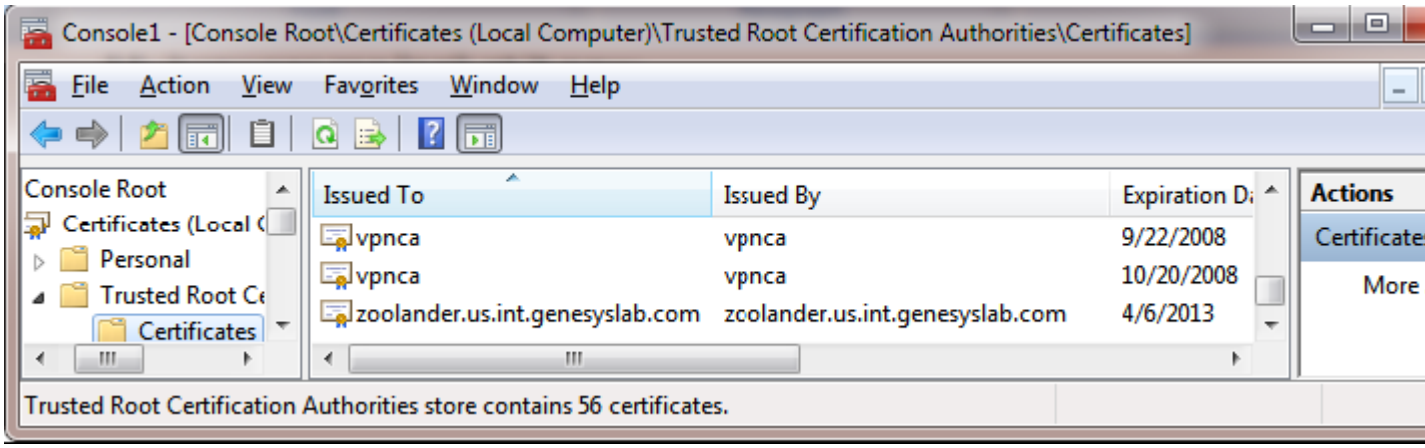
e. Click `Finish`, then `OK`.

f. Right-click `Certificates > Trusted Root Certification Authorities > Certificates`.

g. Choose `All tasks > Import"`

h. Choose `wscpserver.cer` for import.

The certificate is added to the trusted certificates, as shown below.



Certificate Added to Trusted Certificates

3. For Java clients only, import the generated .CER server certificate into a Java keystore. Assuming that a standard JDK is present on the client host, add the server certificate to a trust store on the client host:

```
keytool -import -keystore truststore.jks -file wscpserver.cer -alias wscpserver
```

End

Client Certificate for Browser and .NET Client

A client certificate is required for mutual SSL authentication. If the Web Service Capture Point is configured for server authentication only, the client certificate is not required.

Examples are available of generating the certificate for .NET and for Windows.

Configure Web Service Capture Point for HTTPS

In a Web Service Capture Point application, named, for example, `WSCP_812_zoo`, set the following options:

- server-key-file=<Path to wscpserver.pem>\wscpserver.pem
- password=<'PEM pass phrase' for wscpserver.pem>
- protocol=https
- require-client-authentication=true
- cacert-file=<Path to wscp_clients.pem>\wscp_clients.pem

Do not change any other options. You must restart Interaction Server for these option values to take effect.

If client authentication is not required, set the option `require-clientauthentication` to `false` and omit all procedures relevant to generation and manipulation of client certificates.

HTTPS for WS CP .NET Client

> [1]

Assuming that a .NET client has been previously configured without secure HTTPS, and all of the procedures of generating, exporting, and importing certificates have been completed, you must edit the existing .NET client's `app.config` to make it work over HTTPS. The following example shows the required changes in italics:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <system.serviceModel>
     <behaviors>
       <endpointBehaviors>
         <behavior name="ProvideClientCertificate">
           <clientCredentials>
             <clientCertificate storeLocation="CurrentUser" x509FindType="FindByIssuerName"
findValue="clienthost.us.int.genesyslab.com"/>
           </clientCredentials>
         </behavior>
       </endpointBehaviors>
     </behaviors>
       <bindings>
           <basicHttpBinding>
               <binding name="iWebServiceCapturePointHttpBinding" closeTimeout="00:01:00"
                   openTimeout="00:01:00" receiveTimeout="00:10:00" sendTimeout="00:01:00"
                   allowCookies="false" bypassProxyOnLocal="false"
hostNameComparisonMode="StrongWildcard"
                   maxBufferSize="65536" maxBufferPoolSize="524288"
maxReceivedMessageSize="65536"
                   messageEncoding="Text" textEncoding="utf-8" transferMode="Buffered"
                   useDefaultWebProxy="true">
                   <readerQuotas maxDepth="32" maxStringContentLength="8192"
maxArrayLength="16384"
                       maxBytesPerRead="4096" maxNameTableCharCount="16384" />
                   <security mode="Transport">
                       <transport clientCredentialType="Certificate"/>
                   </security>
               </binding>
           </basicHttpBinding>
       </bindings>
       <client>
           <endpoint address="https://zoolander.us.int.genesyslab.com:10080/Genesys/
Interaction/WSCP_812_zoo/WebServiceCapturePoint"
               binding="basicHttpBinding"
bindingConfiguration="iWebServiceCapturePointHttpBinding"
behaviorConfiguration="ProvideClientCertificate"
               contract="WSCP.iWebServiceCapturePoint"
name="iWebServiceCapturePointHttpBinding" />
       </client>
   </system.serviceModel>
</configuration>
```

HTTPS for WS CP Java Client

Assuming that all of the procedures of generating, exporting, and importing certificates have been completed, the following modifications are required for a Java client to run over HTTPS:

1. Update the URL of `WebService` or `WebService Stub` by replacing `http` with `https`.
2. Start your client with the following JVM options:
   - `-Djavax.net.ssl.keyStore="<Path to keystore.jks>/keystore.jks"`
   - `-Djavax.net.ssl.keyStorePassword="<Key store password, set when creating the keystore>"`
   - `-Djavax.net.ssl.keyStoreType=jks`
   - `-Djavax.net.ssl.trustStore="<Path to keystore.jks>/truststore.jks"`
   - `-Djavax.net.ssl.trustStorePassword=<Trust store password, set when creating the truststore>"`
   - `-Djavax.net.ssl.trustStoreType=jks`

Generate Client Certificate (.NET)

This page provides an example of generating a client certificate on Windows using the openssl utility.

Deploy a client certificate for a .NET Client

Start

1. Generate a client certificate:

   ```
   openssl req -x509 -days 365 -subj "/C=US/ST=California/L=Daly City/
   CN=clienthost.us.int.genesyslab.com" -newkey rsa:2048 -keyout
   wscpclientkey.pem -out wscpclient.pem
   ```

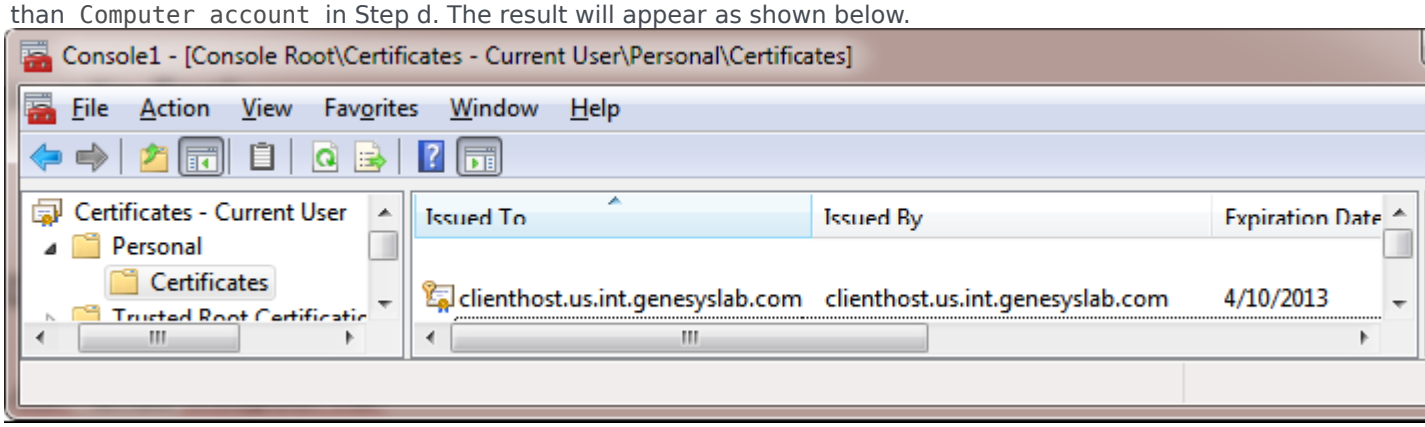
   The output certificate without a private key, `wscpclient.pem`, will be given to the WSCP so that it can authenticate the client. The user will be asked to provide a PEM pass phrase, which is later used to export the certificate, along with the key, `wscpclientkey.pem`, to another format.

2. Export the generated client certificate and the private key into PFX format:

   ```
   openssl pkcs12 -export -out wscpclient.pfx -inkey wscpclientkey.pem -in wscpclient.pem
   ```

   When exporting to PFX format, the user will be asked to provide a pass phrase (the same as the PEM pass phrase referred to in Step 1) and to set an Export Password, which will be used later.

3. Import the PFX certificate to Personal Certificates for Current User: Import the `wscpclient.pfx` with Microsoft Management Console and follow the same procedure as used to import the sever certificate (Step 2 of the server certificate procedure), except that you must choose `My user account` rather than `Computer account` in Step d. The result will appear as shown below.



Importing PFX Certificate

4. Copy the client certificate to the server host: host: Copy the contents of `wscpclient.pem` into a file named `wscp_clients.pem` on the server host.

End

Generate a Client Certificate (Java)

The following procedure provides an example of generating a client certificate using keytool.

Deploy a client certificate for a Java client

Start

1. Generate a Java client key:

   ```
   keytool -genkey -alias javawscpclient -keyalg RSA -keystore
    keystore.jks -keysize 2048
   ```

   This command generates a client key and places it in the local keystore.

2. Export the generated certificate from the keystore:

   ```
   keytool -export -alias javawscpclient -keystore keystore.jks -file
    javawscpclient.cer
   ```

3. Convert the exported certificate to .PEM format:

   ```
   openssl x509 -inform der -in javawscpclient.cer -out
    javawscpclient.pem
   ```

4. Copy the Java client certificate: Append the contents of `javawscplient.pem` to the contents of `wscp_clients.pem` on the server host.

End

Web Service Capture Point Requests (Native)

This page presents details of requests used by the Web Service Capture Point operating in Interaction Server native mode.

Request Submit

This request is used for creating a new interaction. It assumes that `Queue`, `TenantId`, `InteractionType`, `InteractionSubType`,and `MediaType` are either specified in the `default-values` section of the Web Service Capture Point or provided in the request parameters. Example `Submit` request:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/
encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/
2001
/XMLSchema" xmlns:ixn="http://www.genesyslab.com/interaction">
<SOAP-ENV:Body>
<ixn:Submit xmlns="http://www.genesyslab.com/interaction">
<TenantId>101</TenantId>
<Queue>Queue1</Queue>
<ExternalId>Test00001</ExternalId>
<UserData>
<kvitem><key>StringKey</key><value><ValueString>StringValue</ValueString></value></kvitem>
<kvitem><key>IntKey</key><value><ValueInt>812</ValueInt></value></kvitem>
<kvitem><key>List1Key</key><value><ValueList>

<kvitem><key>StringKeyL1</key><value><ValueString>StringValueL1</ValueString></value></kvitem>
    <kvitem><key>IntKeyL1</key><value><ValueInt>1812</ValueInt></value></kvitem>
    <kvitem><key>List2Key</key><value><ValueList>

<kvitem><key>StringKeyL2</key><value><ValueString>StringValueL2</ValueString></value></kvitem>
        <kvitem><key>IntKeyL11</key><value><ValueInt>11812</ValueInt></value></kvitem>
</ValueList></value></kvitem>
</ValueList></value></kvitem>
</UserData>
</ixn:Submit>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Request Hold

This request is used for putting an interaction on hold. It must have either an `InteractionId` or an `ExternalId` argument. Example `Hold` request:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:
xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ixn="http://www.genesyslab.com/interaction">
<SOAP-ENV:Body>
<ixn:Hold xmlns="http://www.genesyslab.com/interaction">
```

```
<ExternalId>Test00001</ExternalId>
</ixn:Hold>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Request Stop

This request is used for stopping a running interaction. It is very similar to request Hold. It must have either an `InteractionId` or an `ExternalId` argument. Only existing, running, or held interactions can be stopped. Example `Stop` request:

```
<tt><?xml version="1.0" encoding="UTF-8"?></tt>
<tt><SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ixn="http://www.genesyslab.com/
interaction"></tt>
<tt><SOAP-ENV:Body></tt>
<tt><ixn:Stop  xmlns="http://www.genesyslab.com/interaction"></tt>
<tt><ExternalId>Test00001</ExternalId></tt>
<tt></ixn:Stop></tt>
<tt></SOAP-ENV:Body></tt>
<tt></SOAP-ENV:Envelope></tt>
```

Request Resume

This request is used for resuming a held interaction. It is very similar to request Hold. It must have either an `InteractionId` or an `ExternalId` argument. Example `Resume` request:

```
<tt><?xml version="1.0" encoding="UTF-8"?></tt>
<tt><SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ixn="http://www.genesyslab.com/
interaction"></tt>
<tt><SOAP-ENV:Body></tt>
<tt><ixn:Resume xmlns="http://www.genesyslab.com/interaction"></tt>
<tt><ExternalId>Test00001</ExternalId></tt>
<tt></ixn:Resume></tt>
<tt></SOAP-ENV:Body></tt>
<tt></SOAP-ENV:Envelope></tt>
```

Request Update

This request is used for changing interaction properties. It must have either an `InteractionId` or an `ExternalId` argument. For changing properties there are the following two structures:

- Changed—For changing existing fields or creating new ones
- Deleted—For removing fields from the interaction

Example `Update` request:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ixn="http://www.genesyslab.com/
interaction">
<SOAP-ENV:Body>
<ixn:Update xmlns="http://www.genesyslab.com/interaction">
<ExternalId>Test00001</ExternalId>
```

```
<Changed>
<kvitem><key>StringKey</key><value><ValueString>StringValueAfterChange</ValueString></value></kvitem>
<kvitem><key>IntKey</key><value><ValueInt>8120</ValueInt></value></kvitem>
</Changed>
<Deleted>
<kvitem><key>List1Key</key><value></value></kvitem>
</Deleted>
</ixn:Update>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Request GetInfo

This request is used for getting interaction properties. It must have either an `InteractionId` or an `ExternalId` argument. Example `Getinfo` request:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ixn="http://www.genesyslab.com/
interaction">
<SOAP-ENV:Body>
<ixn:GetInfo xmlns="http://www.genesyslab.com/interaction">
<ExternalId>Test00001</ExternalId>
</ixn:GetInfo>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Request Ping

This request is used for heartbeat monitoring. It has no required parameters. Example `Ping` request:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ixn="http://www.genesyslab.com/
interaction">
<SOAP-ENV:Body>
<ixn:Ping>
</ixn:Ping>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Web Service Capture Point Responses (Native)

This page presents examples of responses used by the Web Service Capture Point when operating in Interaction Server native mode. All requests except `GetInfo` return a structure called `RequestResponse`. For a successful request, this structure has the following characteristics:

- `Hold, Stop, Resume, Update`—The response is empty.
- `Submit`—The response's `Extension` field contains the Interaction ID returned by Interaction Server.
- `Ping`—The response contains Interaction Server and Capture Points statistics.

The `GetInfo` request returns a structure called `GetInfoResponse`, which contains various fields holding interaction properties.

Example Error Response

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ixn="http://www.genesyslab.com/
interaction">
<SOAP-ENV:Body>
<SOAP-ENV:Fault>
<faultcode>SOAP-ENV:Client</faultcode><faultstring>Required value is missing</faultstring>
<detail>
<ixn:FaultMessage>< ixn:ErrorCode>2</ ixn:ErrorCode>
< ixn:ErrorDescription>Missing InteractionId or ExternalId</ ixn:ErrorDescription>
</ixn:FaultMessage>
</detail>
</SOAP-ENV:Fault>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Example of a Response to a Successful `Submit` Request

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ixn="http://www.genesyslab.com/
interaction">
<SOAP-ENV:Body>
<ixn:RequestResponse>
    <ixn:Extension>
        <ixn:kvitem>
            <ixn:key>InteractionId</ixn:key>
            <ixn:value><ixn:ValueString>02JH8H2FE3Q3T00E</ixn:ValueString></ixn:value>
        </ixn:kvitem>
    </ixn:Extension>
</ixn:RequestResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Web Service Capture Point Requests (iWD-Compatible)

This section presents details of requests used by the Web Service Capture Point when operating in iWD Compatibility Mode.

Request ping

Example of a `ping` request:

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:iwd="http://webservice.capture.gtl.evo">
 <SOAP-ENV:Body>
<iwd:ping>
</iwd:ping>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Sample Request createTask

This sample shows how to specify two k-v pairs and a Genesys-compatible media type in the `data` part of the message, and how to specify a `customerId` Task Extension in the `ext` part of the message.

```
<?xml version="1.0" encoding="UTF-8"
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns2="http://webservices.evo"
xmlns:ns4="http://taskinfo.gtl.evo"
xmlns:ns3="http://broker.gtl.evo"
xmlns:ns1="http://evo"
 xmlns:iwd="http://webservice.capture.gtl.evo">
 <SOAP-ENV:Body>
<iwd:createTask>
<iwd:captureId>TestiWD_0002</iwd:captureId>
<iwd:data xsi:type="iwd:string2stringMap">
<iwd:entry><iwd:key xsi:type="xsd:string">Key1</iwd:key>
<iwd:value xsi:type="xsd:string">Value1</iwd:value></iwd:entry>
<iwd:entry><iwd:key xsi:type="xsd:string">Key2</iwd:key>
<iwd:value xsi:type="xsd:string">Value2</iwd:value></iwd:entry>
<iwd:entry><iwd:key xsi:type="xsd:string">MediaType</iwd:key>
<iwd:value xsi:type="xsd:string">workitem</iwd:value></iwd:entry>
</iwd:data>
<iwd:ext xsi:type="ns3:TaskExt">
<ns3:customerId>My Best Customer</ns3:customerId>
</iwd:ext>
```

```
</iwd:createTask>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Sample Request getTaskByTaskId

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:iwd="http://webservice.capture.gtl.evo">
 <SOAP-ENV:Body>
<iwd:getTaskByTaskId>
<iwd:taskId>02JHNT2FEDRTR005</iwd:taskId>
</iwd:getTaskByTaskId>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Sample Request getTaskByCaptureId

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:iwd="http://webservice.capture.gtl.evo">
 <SOAP-ENV:Body>
<iwd:getTaskByCaptureId>
<iwd:captureId>TestiWD_0002</iwd:captureId>
</iwd:getTaskByCaptureId>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Sample request updateTaskByTaskId

This sample demonstrates how to update various interaction properties.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:ns2="http://webservices.evo" xmlns:ns4="http://taskinfo.gtl.evo"
xmlns:ns3="http://broker.gtl.evo" xmlns:ns1="http://evo"
xmlns:iwd="http://webservice.capture.gtl.evo">
<SOAP-ENV:Body>
<iwd:updateTaskByTaskId>
<iwd:taskId>02JHNT2FEDRTR00B</iwd:taskId>
<iwd:priority>123</iwd:priority>
<iwd:dueDateTime>2012-03-28T20:20:18Z</iwd:dueDateTime>
<iwd:data xsi:type="iwd:string2stringMap">
<iwd:entry><iwd:key xsi:type="xsd:string">Key1</iwd:key>
<iwd:value xsi:type="xsd:string">NewValue1</iwd:value></iwd:entry>
<iwd:entry><iwd:key xsi:type="xsd:string">Key3</iwd:key>
<iwd:value xsi:type="xsd:string">NewKeyNewValue</iwd:value></iwd:entry>
</iwd:data>
<iwd:ext xsi:type="ns3:TaskExt">
```

```
<ns3:customerId>The same customer</ns3:customerId>
</iwd:ext>
</iwd:updateTaskByTaskId>
</SOAP-ENV:Body></SOAP-ENV:Envelope>
```

Web Service Capture Point Responses (iWD-Compatible)

This page presents examples of responses used by the Web Service Capture Point when operating in iWD compatibility mode.

WebserviceFault Error Response

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ns2="http://webservices.evo"
xmlns:ns4="http://taskinfo.gtl.evo" xmlns:ns3="http://broker.gtl.evo" xmlns:ns1="http://evo"
xmlns:iwd="http://webservice.capture.gtl.evo">
<SOAP-ENV:Body>
<SOAP-ENV:Fault>
<faultcode>SOAP-ENV:Client</faultcode>
<faultstring>Interaction Server protocol error</faultstring>
<detail>
<fault xsi:type="ns2:WebserviceFault">
<code>43</code>
<message>Unknown interaction identifier specified</message>
<severity>ERROR</severity>
</fault>
</detail>
</SOAP-ENV:Fault></SOAP-ENV:Body></SOAP-ENV:Envelope>
```

createTaskResponse

The only parameter returned in the `createTaskResponse` is the `out` string, which contains the interaction ID of the new interaction.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:ns2="http://webservices.evo"
xmlns:ns4="http://taskinfo.gtl.evo" xmlns:ns3="http://broker.gtl.evo" xmlns:ns1="http://evo"
xmlns:iwd="http://webservice.capture.gtl.evo"><SOAP-ENV:Body>
<iwd:createTaskResponse>
<iwd:out>02JGQY2FEEP9P000</iwd:out>
</iwd:createTaskResponse>
</SOAP-ENV:Body></SOAP-ENV:Envelope>
```

Java Configuration

For JMS Capture Points and Groovy transformation scripts, as well as for File Capture Points if Groovy transformation scripts are used (for example, for iWD compatibility mode), Java 8 is the minimum required version. Java 11 is recommended. Here is a general description of the configuration requirements for Java:

- Configure the `jvm-path` option in Interaction Server. In the `java-config` section, the `jvm-path` option must specify the path to the `jvm.dll` file (for Windows) or `libjvm.so` file (for UNIX platforms). Interaction Server requires this to start JVM by means of JNI. This option is required for JMS Capture Points and Groovy transformation scripts.
- Configure the `jvm-options` section in Interaction Server. This section lists JVM option pairs, for example `["-Xmx256m",""]` or `["-Djava.class.path",".;C:\myjars\my-jar.jar;C:\myotherjars\my-other-jar.jar"]`. If JMS Capture Points or Groovy transformations are present, the option `-Djava.class.path` must contain a path to the Genesys-provided JAR files, as well as the Message Queue provider-specific JAR files, which are required in order for JMS and Groovy scripts to run.

These options are explained in more detail below. For OpenMQ, the provider-specific jar files are: `jms.jar`, `imq.jar`, and `fscontext.jar`. For TIBCO, the provider-specific jar files are `jms.jar` and `tibjms.jar`.

For more information about these and other Capture Point-related Interaction Server options, refer to the eServices Reference Manual.

Configuring Interaction Server to Load the Java Virtual Machine (JVM)

To enable JMS capture point functionality or Groovy transformation functionality, Interaction Server must be configured to load the Java Virtual Machine. Java 8 is the minimum required version. Java 11 is recommended.

Take care to specify the correct virtual machine with regard to the architecture; that is, for 64-bit Interaction Server, 64-bit JVM must be used and for 32-bit Interaction Server, 32-bit JVM must be used.

Interaction Server java-config Section

The section should contain only one option: `jvm-path`. This option specifies the full path to the `jvm.dll` (on the Windows platform) or to `libjvm.so` (on UNIX platforms). If this option is not present, Interaction Server does not attempt to load JVM. The following is an example of this option for the Windows platform:

```
jvm-path=C:\Program Files\ojdkbuild\java-11-openjdk-11.0.5-1\bin\server\jvm.dll
```

The following is an example of this option for Linux:

```
jvm-path=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.181-7.b13.el7.x86_64/jre/lib/amd64/
server/libjvm.so
```

Interaction Server jvm-options Section

This section specifies options that are used to run the JVM. Interaction Server composes the startup string for the JVM containing all of the options specified in this section.

-Xss1m

This option, with empty value, is required for all platforms. It specifies that the Java stack size should be 1 megabyte.

-Xoss1m

This option, with empty value, is required for all platforms. It specifies that the Native code stack size should be 1 megabyte.

It is important to note that Interaction Server creates many working threads to perform its tasks. If the stack size is set to be large, the multiplicity of threads will consume an unnecessarily large amount of memory. Many UNIX systems have unreasonably large default setting for stack size; the recommended stack size for Interaction Server is 1 megabyte.

-Djava.class.path

This option specifies all of the required JAR files. There are a few JAR files provided with the Interaction Server IP that implement Java wrappers to access JMS or Groovy transformation functionality. Additional JAR files are required to use different JMS providers or to use some specific features in customized transformation scripts. The following is the minimal class path that contains all the standard JAR files provided with Interaction Server:

```
-Djava.class.path=lib/ixn-java-aux.jar:lib/groovy-all-2.4.21.jar:lib/XmlTransformer/
xercesImpl.jar:lib/XmlTransformer/xsltc.jar
```

> **Important**
>
> Check your IXN Server installation for exact versions of third-party libraries provided because they may differ from ones mentioned in this documentation.

To make it work, for example, with Open MQ JMS provider on Windows platform, the following class path is required (considering the default installation path for Open MQ):

```
-Djava.class.path=lib/ixn-java-aux.jar;lib/groovy-all-2.4.21.jar;lib/XmlTransformer/
xercesImpl.jar;lib/XmlTransformer/xsltc.jar;
 C:\Program Files\Sun\MessageQueue\mq\lib\fscontext.jar; C:\Program Files\Sun\MessageQueue\mq\
lib\jms.jar;
 C:\Program Files\Sun\MessageQueue\mq\lib\imq.jar
```

For different platforms or different installation paths of the JMS provider, it must be adjusted accordingly.

> **Important**
>
> Staring from version 9.0.009.03, **slf4j-api-1.7.36.jar** must be explicitly specified in the `-Djava.class.path` option within the `java-options` section if Groovy Event Logger, JMS Event Logger/Capture Point, or Kafka Event Logger/Capture Point is used.

-Djava.library.path

This option specifies the path to native libraries that might be required by JVM or specific JMS providers. On the Windows platform it is usually not necessary to specify this option. On UNIX platforms this option must specify the path to the JRE libraries and in certain cases the path to `libjvm.so` itself.

Take extreme care to specify the library path to the same JRE directory from which `libjvm.so` is loaded (the `jvm-path` option). If these do not match, it is often hard to find the reason why the solution is not working.

Operating System Environment

Interaction Server itself does not make use of any environment variables and should not require Java to be in the path or `JAVA_HOME` environment variable to be set. But if these are set, they **must** refer to the same JRE that is configured in the Interaction Server configuration options.

Different operating systems have different default settings for maximum number of threads a process can create. Interaction Server can and will create a few dozen threads. It is important that limits set for the operating system allow creating a few hundred threads. The default value of 1024 should be sufficient for almost all purposes. Consult with your system administrator to check the operating system limits and ensure that these are adequate for Interaction Server.

Another important operating system parameter is the stack size for the thread. As previously mentioned, Interaction Server creates many threads and requires reasonable stack size for the threads. Some systems might have a default in the vicinity of 256 MB or more, which will definitely lead to problems when a process tries to create a few dozen threads. The stack size should be set to 2 MB for Interaction Server. The following command changes the thread stack size for most UNIX operating systems:

```
ulimit -s 2048
```

Again, consult your system administrator to check and ensure the correct operating system limits are in place before running Interaction Server.

XML Representation

The integrated JMS Capture Point is capable of capturing interactions in the form of XML documents from JMS-compliant message queue providers. The File Capture Point also captures XML documents, but from a local or network directory. This section describes both inbound and outbound XML messages for these two types of capture points.

Inbound Messages

A correctly generated XML document can use different encodings and will contain encoding specification in the document header. For that reason, XML should be always treated as binary data, not text. An XML document should always be put in a message queue as a binary message or written to a file as binary data. Message queue capture points, such as the JMS capture point, can accept binary messages and text messages (for backward compatibility). To avoid incorrect or unnecessary transcoding, ensure that the XML document uses the same encoding that a specific message queue provider uses to encode the text messages. The following encodings of *inbound* XML documents are supported:

- UTF-8
- UTF-16
- ISO-8859-1
- US-ASCII

All *outbound* XML documents are encoded using UTF-8. Inbound XML documents should follow the element structure outlined in this guide.

> **Important**
>
> Timestamps in inbound messages for Capture Points that process XML requests (JMS and File Capture Points) are treated as UTC.

Processed and Error Queues in JMS Capture Point

For message queue capture points, a copy of the original message is put either into the `processed` or `error` queues specified by the options `processed-queue-name` and `error-queue-name`, respectively. These options are configured in the `settings` section of the Capture Point Application object. No reformatting of the message takes place and no XML parsing or transformation is involved. This is an exact copy of the original message.

Outbound Notifications

The outbound XML encoding is UTF-8. For the JMS capture point, the message type of the outbound notification messages is controlled by the option `outbound-message-type`, and can be either `binary` (the default) or `text`. The messages placed in the notifications queue (JMS Capture Point) or

folder (File Capture Point) consist of outbound notifications and responses to capture point requests. The correlation identifiers in notification messages are not set because these are unsolicited notifications and not the replies. The correlation identifier is set for reply messages to correlate responses with requests. Outbound notifications are generated as separate XML documents with the root element `interaction`. The `operation` attribute specifies the type of notification and can be one of the following:

- `changed`—The interaction properties have changed.
- `stopped`—The interaction has been stopped/deleted.
- `held`—The interaction has been put on hold.
- `resumed`—The interaction has been resumed from hold.
- `moved`—The submitted interaction has been moved from one queue.
- `assigned`—The interaction has been delivered to an agent or pushed to a strategy.

Timestamps for outbound notifications and responses sent by the integrated capture points are in UTC (Coordinated Universal Time). This is inconsistent with iWD capture points. Outbound notifications and responses sent by iWD capture points are in local time. Outbound notifications are generated in a specific XML format.

Responses to Capture Point Requests

The responses are formatted the same way as notifications. Everything that is applicable to notification messages also applies to response messages, except for:

- Correlation Id (JMS Capture Point)
- Response Types (JMS and File Capture Points)
- Error Notification (JMS and File Capture Points)

These three areas are unique to the responses to capture point requests.

Inbound Messages

Operation Elements and Root Element

An inbound XML document can contain multiple operations, but only a single root element. For maximum flexibility the name of the root element can be anything, and it is not taken into account. The transformation scripts use `messages` as the name of the root element. If the document contains a single operation, this operation can be a root element. Any operation item is specified by an `interaction` element. The following are sample XML messages:

Sample 1

```
<?xml version="1.0" encoding="UTF-8"?>
<interaction operation="submit" ExternalId="SomeExternalId"/>
```

Sample 2

```
<?xml version="1."? encoding="UTF-8"?>
<messages>
<interaction operation="submit" ExternalId="ExternalId2"/>
<interaction operation="submit" ExternalId="ExternalId3"/>
</messages>
```

Sample 3

```
<?xml version="1."? encoding="UTF-8"?>
<myinteractions>
<interaction operation="submit" ExternalId="ExternalId4"/>
<interaction operation="submit" ExternalId="ExternalId5"/>
</myinteractions>
```

Operations

The operation type is specified by the `operation` attribute and can be one of the following:

- `submit`—Submit a new interaction
- `update`—Update or change interaction properties
- `hold`—Hold the interaction
- `resume`—Resume the interaction
- `stop`—Stop or delete the interaction
- `getinfo`—Request interaction properties

Properties Element

The `properties` element, which should be a direct child of the `interaction` element, specifies the interaction properties that are needed to perform the operation.

- For `submit`, the `properties` element specifies all of the interaction properties including any user data or custom properties. It also specifies standard attributes such as the tenant and queue to which interactions are submitted. Any attribute can have a default value specified in the capture point configuration.
- For `update`, the `properties` element specifies properties that need to be changed. This might include the `Queue` property, in which case the interaction will be moved into the specified interaction queue. For the `update` operation, configured default values are not used, and the attribute `InteractionId` or `ExternalId` must be specified.
- For `hold`, `resume`, and `stop`, the only attribute required or processed is `InteractionId` (or `ExternalId`), which specifies the interaction that is to be held, resumed, or stopped.

For simplicity, any child element of the `properties` element can be specified as an attribute of the `interaction` element. For example, to hold an interaction the following `interaction` element can be used:

```
<?xml version="1."? encoding="UTF-8"?>
<interaction operation="hold" InteractionId="itx00777"/>
```

Interaction Server supports key-value lists (of any depth) as interaction properties. To specify such attributes, natural XML structure is used. Note the `CustomerInfo` group of properties in the following example:

```
<?xml version="1."? encoding="UTF-8"?>
<interaction operation="submit">
  <properties>
    <ExternalId>SomeExternalId</ExternalId>
    <CustomerSegment>Gold</CustomerSegment>
    <CustomerInfo>
      <FirstName>William</FirstName>
      <LastName>Bell</LastName>
    </CustomerInfo>
  </properties>
</interaction>
```

Interaction Server supports spaces and some special characters in interaction property names. To allow for this in XML messages, any property can have a "real" name specified as a `name` attribute. For example:

```
<?xml version="1."? encoding="UTF-8"?>
<interaction operation="submit">
  <properties>
    <property name="First Name">William</property>
    <property name="Last Name">Bell</property>
  </properties>
</interaction>
```

The following is a list of Interaction Server's predefined properties and their meanings. Custom properties can also specified and attached to the interaction. <InteractionId>—Interaction Identifier. Can be omitted and generated by Interaction Server. <ParentId>—Parent interaction IDentifier. <ExternalId>—Identifier used by the external system. <TenantId>—Tenant IDentifier. <MediaType>—Interaction media type. <InteractionType>—Inbound, Outbound, or Internal.

<InteractionSubtype>—Interaction subtype, selected from the list defined for the tenant. <IsOnline>—The interaction is (1) or is not (0) online. <IsHeld>—The interaction is (1) or is not (0) on hold. <Queue>—Name of the queue in which the interaction is initially placed. <Workbin>—Initial workbin name; optional. <WorkbinAgentId>—Initial workbin agent ID. <WorkbinAgentGroupId>—Initial workbin agent group ID. <WorkbinPlaceId>—Initial workbin place ID. <WorkbinPlaceGroupId>—Initial workbin place group ID. <ReceivedAt>—Date and time received; format is YYYY-MM-DD HH:MM:SS. <Priority>—Initial interaction priority. <ServiceType>—Service type. <ServiceObjective>—Service objective in seconds.

Delete Element

The `delete` element, which must be a direct child of the `interaction` element, is used only for the `update` operation, and specifies the names of the properties that are to be deleted. For example:

```
<?xml version="1."? encoding="UTF-8"?>
<interaction operation="update" InteractionId="itx00777">
  <properties>
    <property name="Last Name">Ball</property>
  </properties>
  <delete>
    <property name="Middle Name"/>
  </delete>
</interaction>
```

Reason Element

The reason element, which must be a direct child of the `interaction` element, can specify the reason for the operation. This attribute is optional and can be used with the `hold`, `resume` and `stop` operations. The `reason` element has the attributes `name` and `description`. For example:

```
<?xml version="1."? encoding="UTF-8"?>
<interaction operation="hold" ExternalId="Loan1022011-02">
  <reason name="AwaitingInfo" description="Waiting for credit history report"/>
</interaction>
```

Responses to Capture Point Requests

Correlation Id

In response messages the JMSCorrelationID parameter is set to the JMSMessageID of the request by default. In Interaction Server 8.1.200 and later, you can change this default behavior using the JMS Capture Point's `use-correlation-id-in-reply` option: with a setting of `true`, the JMSCorrelationID parameter of the reply message is set to the value of JMSCorrelationID parameter of the request. A setting of `false` retains the default behavior.

Response types

The following operation types are used in responses to capture point requests:

- `submitted`—Only as a response to the `submit` operation, and never as an unsolicited notification
- `changed`—As a response to a capture point's `change` request or as a notification regarding changes to interactions submitted by this capture point
- `stopped`—As a response to the `stop` operation or as unsolicited notification if an interaction submitted by this capture point is stopped by another entity
- `held`—As a response to the `hold` operation or as unsolicited notification if an interaction submitted by this capture point is held by another entity
- `resumed`—As a response to the `resume` operation or as unsolicited notification if an interaction submitted by this capture point is resumed by another entity
- `info`—Only as a response to a `getinfo` request from a capture point
- `error`—In response to any failed request

Error Notification

The `error` element specifies the error code and (optionally) a description if an operation has failed. The following is an example of the `error` element:

```
<interaction operation="error" code="agent" description="107"/>
```

Outbound Notifications

Properties Element

The `properties` element specifies all current interaction properties. The following is the list of predefined Interaction Server properties. Note that any user data is also presented along the predefined properties. <InteractionId>—Interaction identifier.

<ParentId>—Parent interaction identifier.

<ExternalId>—Identifier used by external system.

<TenantId>—Tenant identifier.

<MediaType>—Interaction media type.

<InteractionType>—Inbound, Outbound, or Internal.

<InteractionSubtype>—Interaction subtype, selected from the list defined for the tenant.

<IsOnline>—The interaction is (1) or is not (0) online.

<IsHeld>—The interaction is (1) or is not (0) on hold.

<Queue>—Current queue name.

<Workbin>—Current workbin name, optional.

<WorkbinAgentId>—Workbin agent ID.

<WorkbinAgentGroupId>—Workbin agent group ID.

<WorkbinPlaceId>—Workbin place ID.

<WorkbinPlaceGroupId>—Workbin place group ID.

<SubmittedBy>—Capture point name.

<InQueues>—List of suggested destination queues.

<ReceivedAt>—Date and time received; format is YYYY-MM-DD HH:MM:SS.

<SubmittedAt>—Date and time submitted; format is YYYY-MM-DD HH:MM:SS.

<DeliveredAt>—Date and time delivered; format is YYYY-MM-DD HH:MM:SS.

<PlacedInQueueAt>—Date and time placed in queue; format is YYYY-MM-DD HH:MM:SS.

<MovedToQueueAt>—Date and time moved to queue; format is YYYY-MM-DD HH:MM:SS.

<AssignedTo>—Agent ID (Place ID if no Agent ID is present in the login).

<AssignedAt>—Date and time assigned; format is YYYY-MM-DD HH:MM:SS.

<Priority>—Current interaction priority.

<ServiceType>—Service type.

<ServiceObjective>—Service objective in seconds.

Changed and Deleted Elements

The changed and deleted elements are used only with the changed notification and specify changed and deleted interaction properties respectively. For example:

```
<?xml version="1."? encoding="UTF-8"?>
<interaction operation="changed" event_time="2010-10-22T07:34:05Z">
  <properties>
    <InteractionId>05512B2CQRPPR001</InteractionId>
    <InteractionType>Inbound</InteractionType>
    <InteractionSubtype>InboundNew</InteractionSubtype>
    <TenantId>107</TenantId>
    <Queue>Inbound</Queue>
    <MediaType>workitem</MediaType>
    <SubmittedBy>CapturePointMSMQPerceptron</SubmittedBy>
    <State>3</State>
    <ReceivedAt>2010-10-19T23:47:32Z</ReceivedAt>
    <SubmittedAt>2010-10-19T23:47:32Z</SubmittedAt>
    <DeliveredAt>2010-10-22T07:33:05Z</DeliveredAt>
    <PlacedInQueueAt>2010-10-19T23:47:32Z</PlacedInQueueAt>
    <MovedToQueueAt="MovedToQueueAt="">2010-10-19T23:47:32Z</MovedToQueueAt>
    <AssignedAt>2010-10-22T07:33:05Z</AssignedAt>
    <AssignedTo>a0001</AssignedTo>
    <ExternalId>MyExternalId</ExternalId>
    <LastName>Ball</LastName>
  </properties>
  <changed>
    <LastName>Ball</LastName>
  </changed>
  <deleted>
    <CustomerSegment/>
  </deleted>
  <actor type="agent" tenant="107" place="p0001" agent="a0001"/>
</interaction>
```

Reason and Actor Elements

The reason element specifies the reason for the operation, if it is provided by the server (and if, in turn, it was provided by the client in the request). The actor element specifies the actor of the operation and can be one of the following types:

- agent—The actor is an agent application and the tenant, place, and agent attributes specify the tenant identifier, place name and agent employee ID.
- strategy—The actor is a strategy and the tenant, strategy, and router attributes specify the strategy.
- mediaserver—The actor is a media server and the mediaserver attribute specifies the name of the media server.

The following is an example of the actor and reason elements (not all properties are included in

this example):

```
<?xml version="1."? encoding="UTF-8"?>
<interaction operation="held" event_time="2010-10-22T07:54:43Z">
  <properties>
    <InteractionId>05512B2CQRPPR001</InteractionId>
    <ExternalId>MyExternalId</ExternalId>
  </properties>
  <actor type="agent" tenant="107" place="p0001" agent="a0001"/>
  <reason name="AwaitingInfo" description="Waiting for credit report"/>
</interaction>
```

The following is an example of the `strategy` actor:

```
<actor type="strategy" tenant="107" strategy="InboundStrategy" router="URServer"/>
```

The following is an example of the `mediaserver` actor:

```
<actor type="mediaserver" server="CapturePointJMS"/>
```

Party Element

The `party` element is used in `assigned` notifications and specifies a party to which the interaction has been assigned. A party can be either an agent or a strategy. The `type` attribute specifies the party type and can be either `agent` or `strategy`. The `tenant` attribute specifies the identifier for the tenant to which the party belongs. The following is an example of an `agent` party (note the `place` and `agent` attributes):

```
<party type="agent" tenant="107" place="p0001" agent="a0001"/>
```

The following is an example of a `strategy` party (note the `strategy` and `router` attributes):

```
<party type="strategy" tenant="107" strategy="InboundStrategy" router="URServer"/>
```

Transformation

The integrated capture point functionality in Interaction Server supports optional message transformation for the File, JMS, and Kafka capture points. Internally, these capture points work with messages in XML format. XML message transformation can be applied to each incoming and outgoing message, allowing integration with custom interaction definitions and XML formats.

A sample iWD compatibility transformation script is included with the installation of Interaction Server and can be used as a basis for customization. The sample script is a transformation script that allows you to format an XML message according to the iWD 8.0 schema, and then have it transformed into Interaction Server's native message format.

Inbound vs. Outbound Transformations

The *inbound* transformation Groovy script (if specified by the `xsl-inbound-transform-path` option) is called when a capture point needs to transform an inbound XML message. The *outbound* transformation Groovy script (if specified by the `xsl-outbound-transform-path` option) is called when a capture point needs to transform outbound XML message.

Configuration Options

In order to enable transformation, the following options must be configured in the `settings` section of the Capture Point Application object:

For full descriptions of each of the following options, refer to the *eServices Reference Manual.*

- `xsl-inbound-transform-path`—String representation of a Uniform Resource Identifier (URI) that points to a shared Groovy script file containing the inbound transformation script.
- `xsl-outbound-transform-path`—String representation of a URI that points to a shared Groovy script file containing the transformation scripts for outbound notifications.

Transformers Interface

The interface for the transformation script is defined as follows:

```
package com.genesyslab.eservices.interactionserver.capturepoints.xmltransformer;
public interface XmlTransformer
{
        void init(java.util.Properties parameters);
        byte[] transform(byte[] inputXml, java.util.Properties parameters);
        void cleanup();
        void setLogger(Logger logger);
        void reconfigure(java.util.Properties parameters);
}
```

Any custom script should implement this interface in order to be usable by Interaction Server. For a good starting point for a custom script, refer to the sample scripts that are provided with Interaction Server.

When Interaction Server creates a transformer, it calls its `init` method and passes all of the parameters that are defined in the `inbound-transformer-parameters` section (for inbound transformation scripts) or in the `outbound-transformer-parameters` section (for outbound transformation scripts). The transformer object should store these properties for possible future use during transformation.

The main functional method `transform` transforms the `inputXML` XML message into the required form and returns the transformed XML message that will be either parsed by Interaction Server (for inbound messages) or, for outbound messages, put directly into notification message queue (for message queue capture points). Each call to the `transform` method by Interaction Server can be given a set of properties to account for during the individual transformation process of a single document. Interaction Server provides the following parameters to the transformation method:

Parameter	JMS capture point	File capture point	Kafka capture point
CapturePointName	The name of the capture point invoking the transformation	The name of the capture point invoking the transformation	The name of the capture point invoking the transformation
CurrentTime	The current UTC timestamp in the format `YYYY-MM-DDTHH:MM:SSZ`	The current UTC timestamp in the format `YYYY-MM-DDTHH:MM:SSZ`	The current UTC timestamp in the format `YYYY-MM-DDTHH:MM:SSZ`
`CapturePointType` (Only for outbound transformation)	The type of capture point - jms	The type of capture point - file	The type of capture point - kafka
`MessageId` (Only for inbound transformation)	The `JMSMessageId` property of the JMS message being transformed	The file name of the inbound file currently being transformed	The artificial unique identifier consisting of partition number the message was consumed from and its offset divided by dot (.)
`CorrelationId` (Only for outbound transformation)	The `JMSCorrelationId` property of the JMS message being transformed		The correlation ID as it was read from the inbound message header as specified in the `correlation-id-header-key` option
`OutboundMessage`	Contents of the original outbound JMS message		
`InboundMessage`	Contents of the original inbound JMS message		
`CopyOriginalHeaders`			`true` or `false` as specified in the `copy-original-headers-in-reply` option
`InboundRecord` (Only for inbound transformation)			The ConsumerRecord object which the message being transformed was received from.
`OutboundHeaders`			The List object in which Kafka message headers to be included

Parameter	JMS capture point	File capture point	Kafka capture point
(Only for outbound transformation)			into an outbound message can be added during transformation.

The `Logger` interface provided to the script allows for logging of any diagnostic or error messages into the Interaction Server log by the same means that Interaction Server logs messages. Logging configuration works the same as for any other Interaction Server messages, including logging to the console, a file, or network logging. The `Logger` interface is defined as follows:

```
package com.genesyslab.eservices.interactionserver.capturepoints.xmltransformer;
public interface Logger
{
    public static enum LogLevel { DEBUG, TRACE, STANDARD };
    public void log(LogLevel level, String logMessage);
}
```

XML Encoding Considerations

Interaction Server can parse XML messages in the following encodings:

- UTF-8
- UTF-16
- ISO-8859-1
- US-ASCII

Outbound notification messages are encoded in UTF-8. This requires the transformation scripts to provide output in one of the supported encodings (for inbound transformations) and to be capable of parsing the UTF-8 encoded XML messages (for outbound transformation scripts).

Because Groovy scripts use the `XmlParser` class to parse XML messages, they have no difficulty processing UTF-8 and can support any encoding supported by Java (depending on the installed packages). The outbound transformation scripts can also produce outbound XML messages in any encoding if the appropriate Java packages are installed. The outbound transformer provided with Interaction Server generates output in UTF-8 and is capable of generating output (without any additional Java packages) in the following encodings:

- US-ASCII
- ISO-8859-1
- UTF-8
- UTF-16BE
- UTF-16LE
- UTF-16

Care should be taken to correctly handle encoding in Groovy. The recommended place to look for an example is the transformation scripts that are provided with the Interaction Server installation. The following pattern shows how to correctly generate XML in the required encoding and with appropriate XML declaration:

```
def outputDoc = new StreamingMarkupBuilder()
outputDoc.encoding = "utf-8"
def outputStream = new ByteArrayOutputStream()
new OutputStreamWriter(outputStream, outputDoc.encoding) << outputDoc.bind {
        mkp.xmlDeclaration()
        somecontent {
    }
}
return outputStream.toByteArray()
```

iWD Compatibility Transformation Scripts

There are two Groovy scripts provided for transformation of inbound and outbound messages to and from iWD message format. These scripts provide backward compatibility with iWD 8.0 message format considering the structure of the iWD specific business process provided with iWD 8.0. The iWD 8.0 message format is described in detail in the iWD 8.0 Deployment Guide. Only general rules of the transformation process are described here. The provided transformation scripts below are only for standard iWD messages as specified in the document. For custom messages, customization of these scripts is necessary.

- Inbound Transformation Script
- Outbound Transformation Script

Inbound Transformation Script

The inbound transformation script path is `iwd_scripts\iWD2IxnServerTransformer.groovy`. The script produces output in UTF-8 for Interaction Server to parse.

Inbound Script Parameters

The script uses the following parameters:

- `CompleteQueues`—A comma-separated list of queue names for completed interactions (default `iWD_Completed`).
- `RestartQueues`—A comma-separated list of queue names for new interactions (default `iWD_New`).
- `CancelQueues`—A comma-separated list of queue names for canceled interactions (default `iWD_Canceled`).
- `ExtendedAttributes`—A comma-separated list of attributes that must be present under the `<Ext>` tag of the `CreateTask` iWD message.
- `AllowAnyAttributes`—If set to `true` or `yes`, the transformation script copies any unknown attributes to the transformed message.
- `CaseSensitiveAttributes`—If set to `false` or `no`, the transformation script ignores the case of known attribute names (including `Ext` and `Data` section names).
- `CaseSensitiveActions`—If set to `false` or `no`, the transformation script ignores letter case of action names.

  Interaction Server parser and interaction representation are case sensitive. The customized script must take care to produce the output in the correct case.

Root Element

The root element of iWD inbound message may be `GTLMessages` or `GTLMessage`. The script checks for the root element name and generates an error if the document root element is anything else. The root element of the transformed messages is always `messages` and the child elements describe the operations.

Transforming Actions

The iWD message action is the name of the tag of the child element of the root element. Possible actions and their translations are as follows:

- `CreateTask`—Translates to `<interaction operation='submit'>`
- `GetTaskInfo`—Translates to `<interaction operation='getinfo'>`
- `UpdateTask`—Translates to `<interaction operation='update'>`
- `CompleteTask`—Translates to `<interaction operation='update'>`

- HoldTask—Translates to <interaction operation='hold'>
- ResumeTask—Translates to <interaction operation='resume'>
- RestartTask—Translates to <interaction operation='update'>
- CancelTask—Translates to <interaction operation='update'>

CompleteTask, RestartTask, and CancelTask are transformed into the update operation, which allows changing the queue for the interaction. The queue name is then added based on transformer parameters. Specifically, for the CompleteTask action, the first queue name from the transformer parameter CompleteQueues is added as the Queue property of the translated message. For the rest of these actions, the first queue name from the appropriate parameter is taken.

Transforming Properties

The following transformation takes place for the inbound iWD message:

- All known direct children of the action element are translated according to the Translation Table for Known Attributes (Inbound) below and put into the properties tag of the transformed message.
- If any unknown tag is encountered, it is ignored if the AllowAnyAttributes option of the transformer is not set to true or yes. If the option is set to true or yes, the attribute is copied without any translation to the properties tag of the transformed message.
- If the Ext tag is encountered, all children of this tag are copied into the properties tag of the transformed message with the prefix IWD_ext_.
- If the Data tag is encountered, all children of this tag are copied into the properties tag of the transformed message without any changes.
- If the Reason tag is encountered, it is translated to the reason tag with the name attribute containing the value of the original Reason tag (for example <reason name="Original Reason"/>).

Attributes appear in the transformed message in the order described above.

Translation Table for Known Attributes (Inbound)

iWD Message Attribute	Attribute Name in Interaction Server	Notes
BrokerId	InteractionId	
CaptureId	ExternalId	
Actor		Ignored
ActionDateTime		Ignored
tenantId	IWD_tenantId	
solutionId	IWD_solutionId	
capturePointId	IWD_capturePointId	Same as SubmittedBy
priority	Priority	
businessValue	IWD_businessValue	
channel	iWD_channel	

iWD Message Attribute	Attribute Name in Interaction Server	Notes
category	IWD_category	
activationDateTime	IWD_activationDateTime	No default value
dueDateTime	IWD_dueDateTime	No default value
expirationDateTime	IWD_expirationDateTime	No default value
processId	IWD_processId	
departmentId	IWD_departmentId	
reprioritizeDateTime	IWD_reprioritizeDateTime	
Hold	IsHeld	Changed to 0 or 1 (from false or true)

Outbound Transformation Script

The outbound transformation script path is `iwd_scripts\IxnServer2iWDTransformer.groovy`. The script produces output in UTF-8 for Interaction Server to put into the notification queue (or to deliver to an external system by other means).

Outbound Script Parameters

The script uses the following parameters:

- `CompleteQueues`—A comma-separated list of queue names for completed interactions (default `iWD_Completed`)
- `RestartQueue`—A comma-separated list of queue names for new interactions (default `iWD_New`)
- `CancelQueues`—A comma-separated list of queue names for canceled interactions (default `iWD_Canceled`)
- `RejectQueues`—A comma-separated list of queue names for rejected interactions (default `iWD_Rejected`)
- `ExtendedAttributes`—A comma-separated list of interaction attributes that has to appear under the <Ext> tag of the iWD notification messages

The script uses the following internal parameters (hard-coded as static member variables) to maintain the compatibility with previous versions of iWD:

- `includeWorkbinQueueName`—If set to `true`, the default, the script includes the workbin queue name in `TaskDistributedQueue` messages, as is done in iWD. If set to `false`, the actual workbin name is included.
- `specificQueueNotifications`—If set to `true`, the script generates specific unsolicited notifications based on the queue name (for example, `TaskCompleted`, `TaskCanceled`, `TaskRestarted`, `TaskRejected`, `TaskErrorHeld` are generated instead of a generic `TaskDistributedQueue`). The default value is `false`, which generates the generic `TaskDistributedQueue` as is done in iWD.

The two internal parameters described above can be changed in the script file. Changes take effect after restart.

Root Element

The notification messages produced by Interaction Server always contain a single notification. This notification is a root element. The outbound transformation script expects the root element to be `messages` and if it is, treats all of the child elements as notifications (which always have the name `interaction`). If the root element is not `messages`, then it is expected to be `interaction` and is treated as a single notification element. In all other cases the transformation fails. In output XML, the root element is always `GTLMessages` and child elements are iWD notification elements.

Transforming Actions

The Interaction Server operation is specified by the `operation` attribute of the `interaction` tag. Possible actions and their translations are as follows:

- `<interaction operation='created'>`—Translates to `TaskCreated`
- `<interaction operation='changed'>`—Translates to `TaskUpdated`
- `<interaction operation='stopped'>`—Translates to nothing
- `<interaction operation='held'>`—Translates to `TaskHeld`
- `<interaction operation='resumed'>`—Translates to `TaskResumed`
- `<interaction operation='info'>`—Translates to `TaskInfo`
- `<interaction operation='moved'>`—Translates to one of `TaskCompleted`, `TaskRestarted`, `TaskCanceled`, `TaskRejected` or `TaskDistributedQueue`
- `<interaction operation='assigned'>`—Translates to `TaskAssigned`
- `<interaction operation='error'>`—Translates to `Error`

Note the transformation of the `moved` notification to different iWD notifications. The choice of the appropriate notification is made based on the `Queue` attribute of the original notification message as follows:

- If the value of the `Queue` attribute is included in the `CompleteQueues` parameter, then the `TaskCompleted` notification is generated.
- If the value of the `Queue` attribute is included in the `RestartQueue` parameter, then the `TaskRestarted` notification is generated.
- If the value of the `Queue` attribute is included in the `CancelQueues` parameter, then the `TaskCanceled` notification is generated.
- If the value of the `Queue` attribute is included in the `RejectQueues` parameter, then the `TaskRejected` notification is generated.
- Otherwise, the `TaskDistributedQueue` notification is generated.

Transforming Properties

The following transformation takes place for the outbound iWD message:

- All known direct children of the `properties` tag are translated according to the Translation Table for Known Attributes (Outbound) below and put into the transformed message as direct children of the notification message.
- All direct children of the `properties` tag that begin with prefix `IWD_ext_` are put into the `Ext` tag of the transformed message as child elements with the same name, but without the prefix `IWD_ext_`.
- All other children of the `properties` tag are put into the `Data` tag of the transformed message as child elements with exactly same names.

Translation Table for Known Attributes (Outbound)

Attribute Name in `properties`	iWD Message Attribute	Notes
InteractionId	BrokerId	
ExternalId	CaptureId	
SubmittedBy	CapturePointId	
IWD_CapturePointId		Ignored. `SubmittedBy` is used

Attribute Name in `properties`	iWD Message Attribute	Notes
		instead.
<actor>	Actor	Strategy, agent ID, or server name.
<reason>	Reason	Attribute `name of reason` tag.
event_time attribute of the notification	EventDateTime	If not present, it is set to the `CurrentTime` parameter of transformation.
IWD_tenantId	tenantID	
IWD_solutionId	solutionId	
IWD_departmentId	departmentId	
IWD_processId	processId	
IWD_channel	channel	
IWD_category	category	
State, Queue, IsHeld	status	Based on a set of attributes.
IWD_businessCalendarId	businessCalendarId	
SubmittedAt	createdDateTime	
HeldAt	heldDateTime	Only if held, no translation in iWD 8.0
AssignedAt	assignedDateTime	
CompletedAt	completedDateTime	
IWD_activationDateTime	activationDateTime	
IWD_dueDateTime	dueDateTime	
IWD_expirationDatetime	expirationDateTime	
Priority	priority	
IWD_reprioritizeDateTime	reprioritizeDateTime	
IWD_businessValue	businessValue	
AssignedTo	assignedToUser	
Queue	Queue	
Workbin, WorkbinAgentId, WorkbinAgentGroupId, WorkbinPlaceId, WorkbinPlaceGroupId	QueueType	• If Workbin is empty InteractionQueue • If WorkbinAgentId is set AgentWorkbin • If WorkbinAgentGroupId is set AgentGroupWorkbin • If WorkbinPlaceId is set PlaceWorkbin • If WorkbinPlaceGroupId is set PlaceGroupWorkbin

Attribute Name in `properties`	iWD Message Attribute	Notes
WorkbinAgentId, WorkbinAgentGroupId, WorkbinPlaceId, WorkbinPlaceGroupId	QueueTarget	First not empty