Importing an SNA Custom Handshake Class

Part No: 821–2599
March 2011

ORACLE

120126@25097

Contents

Importing an SNA Custom Handshake Class

This document provides links to conceptual information on how to import an SNA Custom Handshake Class. It includes the following steps:

Adding a Custom Handshake Class to a Project

To use a custom handshake class with the SNA Adapter, you need to configure the Adapter by specifying the class name. You also need to import the file into the SNA Project, and if you are using it in a Java Collaboration Definition, you need to import it into the Collaboration as well.

Perform the following steps to use a custom handshake class:

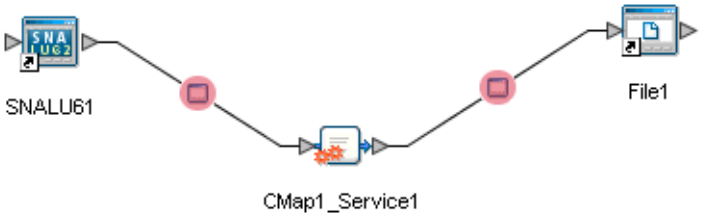
Before you perform these steps, create the custom class and make it available to the NetBeans IDE.

Configuring the Adapter for a Custom Handshake Class

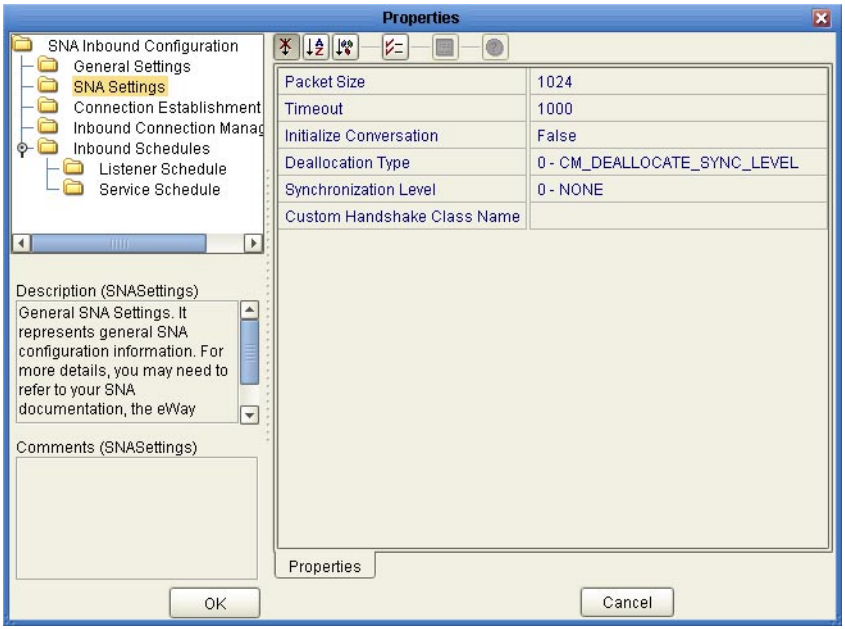
In order for the SNA Adapter to know which custom handshake class to use, you need to specify the class name in the Adapter properties of the Connectivity map.

To Configure the Adapter

1. In the NetBeans IDE, expand the Project containing the SNA Adapter until the Connectivity Map is visible.
2. Open the Connectivity Map in the Connectivity Map Editor.
3. To import the custom handshake class for the inbound Adapter, double-click the inbound SNA Adapter icon.



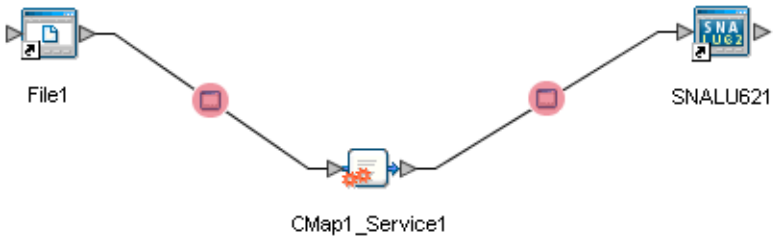
The SNA Adapter Properties Editor appears, displaying the default properties for the inbound Adapter.



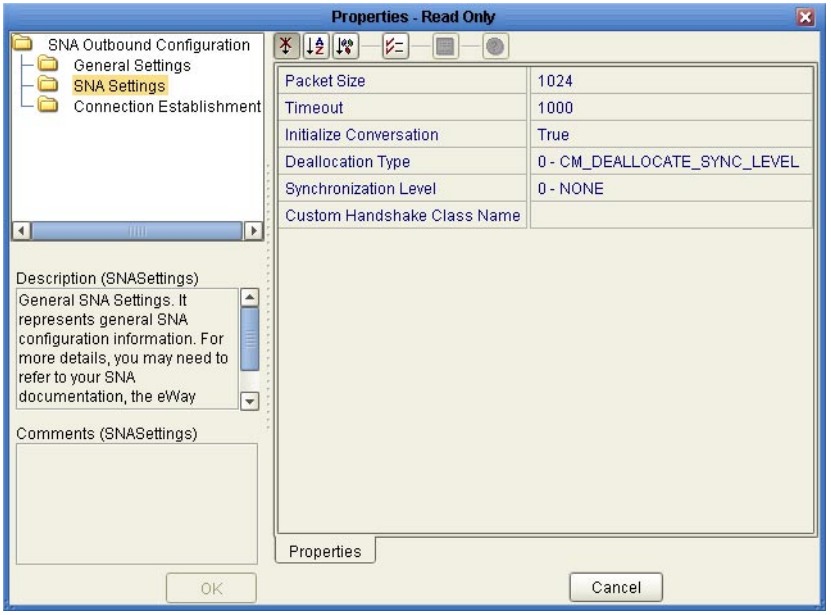
4 Edit the Custom Handshake Class Name property in the Properties Editor and click OK.

For the sample code provided with the Adapter, enter **`com.stc.connector.snalu62.api.SNACustomerHandshakeImplSampleAccept`**.

5 **To import the custom handshake class for the outbound Adapter, double-click the outbound SNA Adapter icon.**



The SNA Adapter Properties Editor appears, displaying the default properties for the outbound Adapter.



6 **Edit the Custom Handshake Class Name property in the Properties Editor and click OK.**

For the sample code provided with the Adapter, enter **`com.stc.connector.snalu62.api.SNACustomerHandshakeImplSampleInitialize`**.

7 **Redeploy your project.**

Importing a Custom Handshake Class into a Project

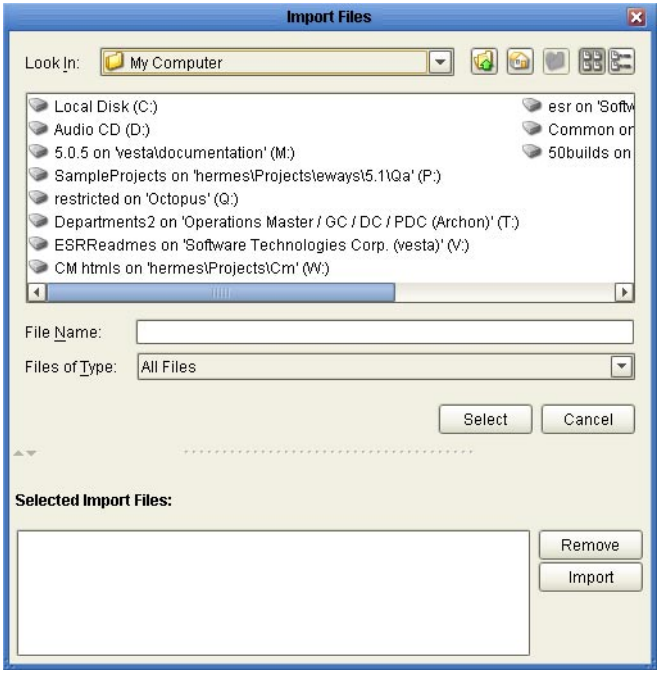
In order to use a custom class in a Project and a Java Collaboration Definition (JCD), you need to package the class into a JAR file and then import the file into both the Project and the JCD.

▼ To Import a Custom Handshake Class

1 **Prepare a JAR file that includes your built class.**

2 **From the NetBeans Projects window, right-click the sample Project and select Import > File.**

The Import Files window appears.



3 **Navigate to the JAR file and click Select.**

The selected JAR file appears in the Selected Import Files pane at the bottom of the Import Files window.

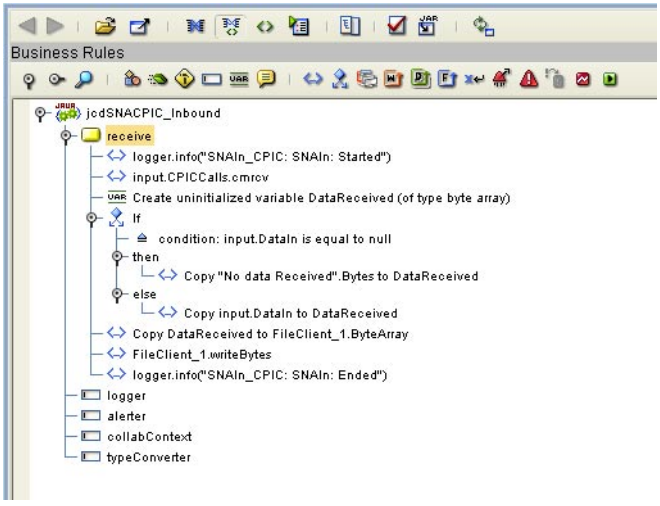
4 **Click Import.**

The selected JAR file appears in the Project tree under the project you selected.

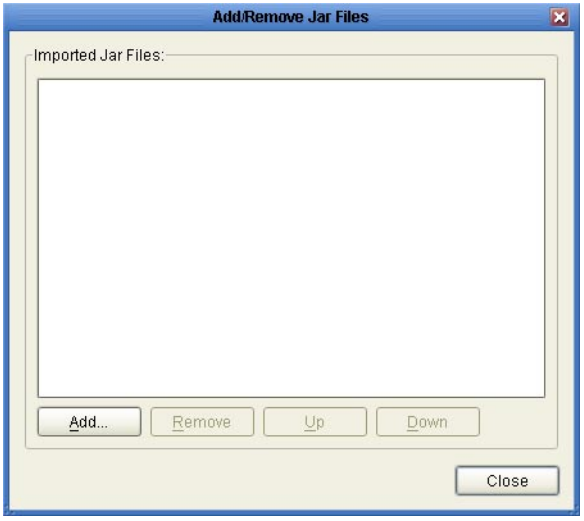
5 Do the following to import the file into a JCD:

a. Open the JCD in the Collaboration Editor.

b. In the Collaboration Editor toolbar, click the Import JAR file icon.



The Add/Remove JAR Files window appears.



c. Click Add, navigate to and select the JAR file, and then click Import.

The selected JAR file appears in the Imported JAR Files pane.

d. Click Close on the Add/Remove JAR Files window.

The selected JAR file appears under the JCD in the Projects window.

Note – If you make any changes to the class, repeat the previous steps.

Sample Handshake Class

The Java Collaboration can handle the SNA connection completely using the default class that is provided with the Adapter, `com.stc.connector.snalu62.api.SNACustomerHandshakeImplSampleDummy`. This class has been implemented in the SNA Adapter. The sample code for this custom class is shown below:

```
package com.stc.connector.snalu62.api;

import com.stc.connector.logging.LogFactory;
import com.stc.connector.logging.Logger;
import com.stc.connector.snalu62.exception.SNAApplicationException;

/**
 * This is a sample class to implement the interface SNACustomerHandshake.
 * It implements a dummy handshake. That is, the method startConversation()
 does not perform a function.
 * No SNA conversation is established inside this implementation class.
You should establish the SNA conversation manually
(e.g. in the java Collaboration).
*/

public class SNACustomerHandshakeImplSampleDummy implements SNACustomerHandshake {
    public static final String version = "cvs $Revision: 1.1.2.2 $
$Date: 2005/11/10 21:40:15 $";
    private Logger logger = LogFactory.getLogger
("STC.eWay.SNALU62." + getClass().getName());

    /**
     * Constructor
     *
     */
    public SNACustomerHandshakeImplSampleDummy() {
        super();
    }

    /**
     * @see com.stc.connector.snalu62.api.SNACustomerHandshake#startConversation
(com.stc.connector.snalu62.api.SNACPICCalls)
     */
    public void startConversation(SNACPICCalls cpic) throws SNAApplicationException {
        logger.info("SNACustomerHandshakeImplSampleDummy.startConversation():
Done nothing here.");

    }

}
```

Implementing an SNA Custom Handshake Class

To further utilize the capabilities of the SNA Adapter, you can implement a custom handshake class in a deployed Project. After the default Collaboration is generated, you can then modify the Collaboration to suit your application's needs. While you need to write your own code for both Inbound and Outbound SNA modes, the following code is also provided as the source for the class that is implemented in the SNA adapter.

Sample Code for Inbound Mode

```
package com.stc.connector.snalu62.api;

import com.stc.connector.logging.LogFactory;
import com.stc.connector.logging.Logger;

import com.stc.connector.snalu62.exception.SNAApplicationException;

/*
 * This is a sample class to implement the interface SNACustomerHandshake.
 * It implements a simple Accept_Conversation scenario for windows platform.
*/
public class SNACustomerHandshakeImplSampleAccept implements SNACustomerHandshake {
    public static final String version = "cvs $Revision: 1.1.2.1.2.2 $
$Date: 2005/11/10 21:40:15 $";
    private Logger logger = LogFactory.getLogger("STC.eWay.SNALU62.
" + getClass().getName());
    private String logMsg;

    /**
     * Constructor
     *
     */
    public SNACustomerHandshakeImplSampleAccept() {
        super();
    }
    /**
     * @see com.stc.connector.snalu62.api.SNACustomerHandshake#startConversation
(com.stc.connector.snalu62.api.SNACPICCalls)
     */
    public void startConversation(SNACPICCalls cpic) throws SNAApplicationException {
        try {
            //do whatever checking logics before/after the following CPIC call
               on your desires
            cpic.cmsltp();

            //do whatever checking logics before/after the following CPIC call
               on your desires
            cpic.cmaccp();
            if (!cpic.getConversationAttributes().returnCodeIs(0) &&   // 0: CM_OK
                !cpic.getConversationAttributes().returnCodeIs(35))
            {  //35: CM_OPERATION_INCOMPLETE
                logMsg = "SNACustomerHandshakeImplSampleAccept.startConversation():
                   The return code is <"
                    + cpic.getConversationAttributes().getReturnCode()
```

```
                    + ">.";
                logger.error(logMsg);
                throw new SNAApplicationException(logMsg);
            }

            if (cpic.getConversationAttributes().returnCodeIs(35))
            {  //35: CM_OPERATION_INCOMPLETE
                logger.info("SNACustomerHandshakeImplSampleAccept.startConversation():
                      About to call cmwait ...");
                //do whatever checking logics before/after the following CPIC call
                  on your desires
                cpic.cmwait();
            }

            if (!cpic.getConversationAttributes().returnCodeIs(0) ||
                !cpic.getConversationAttributes().convReturnCodeIs(0)) { // 0: CM_OK
                logMsg = "SNACustomerHandshakeImplSampleAccept.startConversation():
                    The return_Code is <"
                    + cpic.getConversationAttributes().getReturnCode()
                    + "> and the conversation_Return_Code is <"
                    + cpic.getConversationAttributes().getConvReturnCode()
                    + ">. SNA conversation is not established.";
                logger.error(logMsg);
                throw new SNAApplicationException(logMsg);
            }

            //do whatever other logics on your desires here
            //...
        } catch (Exception e) {
            logMsg = "SNACustomerHandshakeImplSampleAccept.startConversation():
            Failed. Got exception ["
                + e.toString()
                + "].";
            logger.error(logMsg, e);
            throw new SNAApplicationException(logMsg, e);
        }

    }

}
```

Note – The above code has been wrapped for display purposes.

Sample Code for Outbound Mode

```
package com.stc.connector.snalu62.api;

import com.stc.connector.logging.LogFactory;
import com.stc.connector.logging.Logger;

import com.stc.connector.snalu62.exception.SNAApplicationException;

/**
 * This is a sample class to implement the interface SNACustomerHandshake.
 * It implements a simple Initialize_Conversation scenario for windows platform.
```

```
*/
public class SNACustomerHandshakeImplSampleInitialize implements SNACustomerHandshake {
    public static final String version = "cvs $Revision:
 1.1.2.1.2.2 $   $Date: 2005/11/10 21:40:15 $";
    private Logger logger = LogFactory.getLogger("STC.eWay.SNALU62." + getClass().
getName());
    private String logMsg;

    /**
     * Constructor
     *
     */
    public SNACustomerHandshakeImplSampleInitialize() {
        super();
    }

    /**
     * @see com.stc.connector.snalu62.api.SNACustomerHandshake#startConversation
(com.stc.connector.snalu62.api.SNACPICCalls)
     */
    public void startConversation(SNACPICCalls cpic) throws SNAApplicationException {
        try {
            //do whatever checking logics before/after the following CPIC call on your
                desires
            cpic.cminit();

            //do whatever checking logics before/after the following CPIC call
              on your desires
            cpic.cmssl();

            //do whatever checking logics before/after the following CPIC call on your
               desires
            cpic.cmallc();
            if (!cpic.getConversationAttributes().returnCodeIs(0)) { // 0: CM_OK
                logMsg = "SNACustomerHandshakeImplSampleInitialize.
                 startConversation(): The return_Code is <"
                    + cpic.getConversationAttributes().getReturnCode()
                    + ">. SNA conversation is not established.";
                logger.error(logMsg);
                throw new SNAApplicationException(logMsg);
            }

            //do whatever other logics on your desires here
            //...
        } catch (Exception e) {
            logMsg = "SNACustomerHandshakeImplSampleInitialize.startConversation():
               Failed. Got exception ["
                + e.toString()
                + "].";
            logger.error(logMsg, e);
            throw new SNAApplicationException(logMsg, e);
        }

    }

}
```

Note – The above code has been wrapped for display purposes.