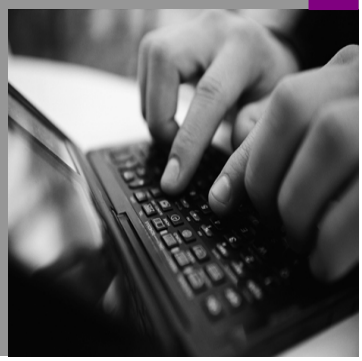


How-to Guide
SAP NetWeaver '04

How To... Create Modules for the J2EE Adapter Engine

Version 1.00 – July 2005

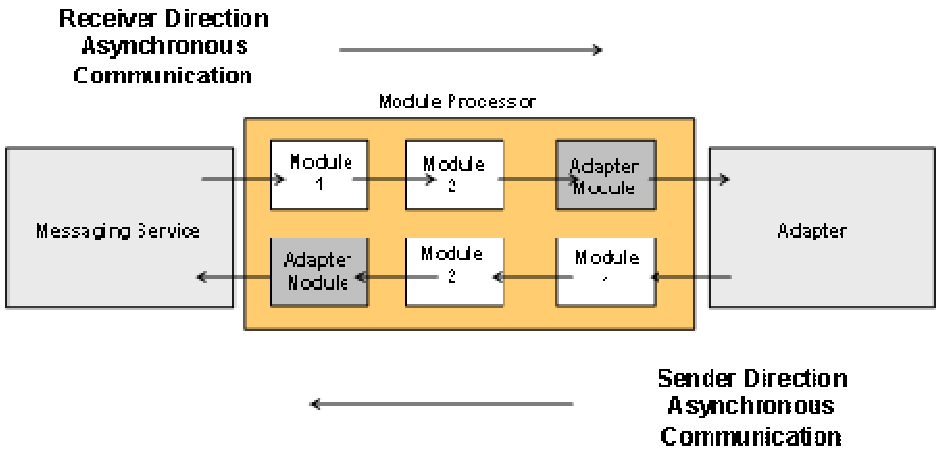
Applicable Releases:
SAP NetWeaver '04
Exchange Infrastructure 3.0

1 Scenario

You want to modify the message payload in an Adapter Engine module, but there is no standard module that meets your requirements.

2 Introduction

A message from the Integration Server is received in the Adapter Framework by the messaging service. Based on the receiver information, the corresponding module chain is selected for further processing. The Adapter Framework contains two default module chains: one for the sender/inbound direction and one for the receiver/outbound direction. You can use these default module chains for the adapter if the entire message processing is executed within the adapter. You can enhance the default module chains with customer-specific modules. The module processor controls the steps in the module chain by calling generic and, if defined, adapter-specific modules. The last module in the module chain forwards the message to the adapter. The adapter transfers the message to the connected system.
Message processing in the sender/inbound direction proceeds in a similar way. The adapter calls the module processor in the form of an EJB 2.0 local session bean and transfers the message object either as an XI message, or in its own format. If the message object is transferred in its own format, it must be converted to an XI message in an adapter-specific module.



The configuration of adapter modules is part of the communication channel in the Integration Directory or Partner Connectivity Kit. If you want to add your own modules to adapters, you integrate these modules at the correct place in the module processor.

3 The Step-By-Step Solution

This manual guides you though the development of a simple adapter module with the help of SAP NetWeaver Developer Studio. Make sure that you have the newest version of the developer studio.
As an example, you will develop an adapter module that creates an additional attachment. This could be useful in the mail adapter, for example.

3.1 Setting up the Developer Studio

You need several libraries to compile your module. The easiest way to obtain these libraries is by downloading the newest patch of *XI ADAPTER FRAMEWORK CORE 3.0* from SAP Service Marketplace. Open the file with WinZip and extract the following SDAs:

- aii_af_lib.sda, aii_af_svc.sda
- aii_af_cpa_svc.sda

Open the SDA aii_af_lib.sda with WinZip and extract the following library files:

- aii_af_cci.jar
- aii_af_mp.jar
- aii_af_ms_api.jar
- aii_af_ms_spi.jar
- aii_af_trace.jar

Open the SDA aii_af_svc.sda and extract the following library file:

- aii_af_svc.jar

Open the SDA aii_af_cpa_svc.sda and extract the following library file:

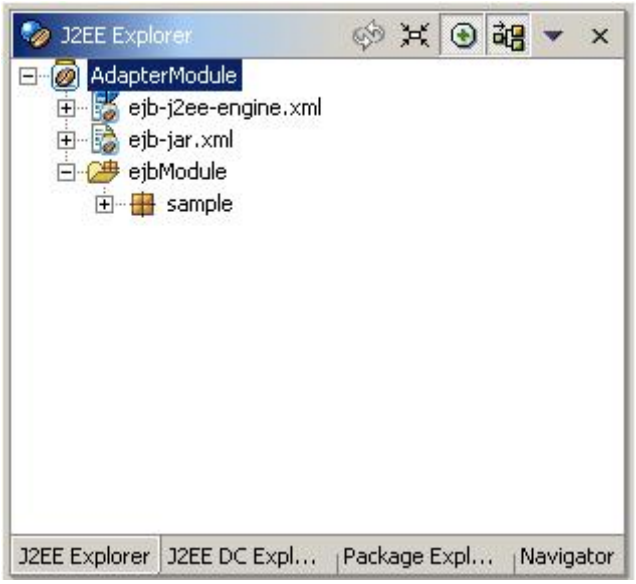
- aii_af_cpa.jar

(You can also find the SDA files on your Adapter Framework installation.)

Enter SAP NetWeaver Developer Studio and carry out the following steps:

1. Create a new Project as *J2EE → EJB Module*.
2. Assign the libraries to your project by choosing *Project → Properties → Java Build Path → Libraries*.
3. Create a package for your Java classes.

The J2EE Explorer of your developer studio should now look as follows:



3.2 Creating the Java Class

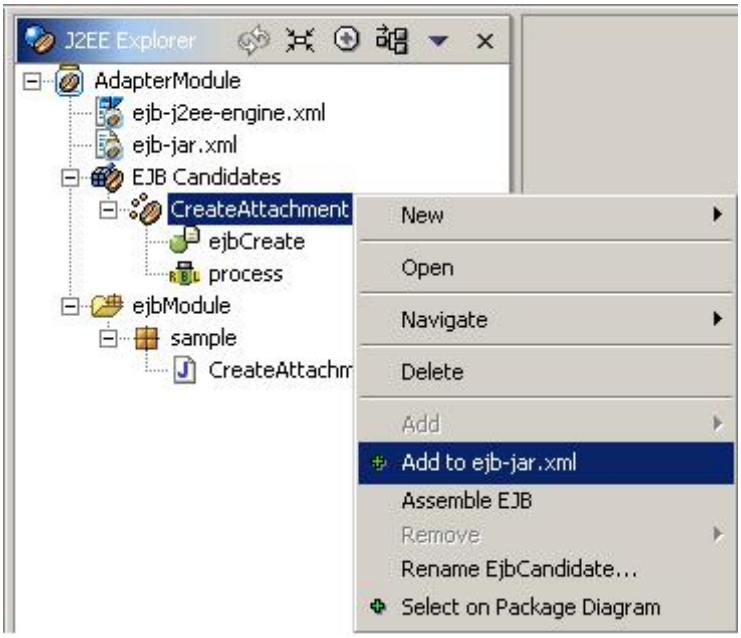
Even if you need a stateless session bean as the adapter module, it is useful not to let the developer studio create it on its own. Instead, create a new Java class and attach the following code template to the class. (Check that the package and class names are correct):

```
package sample;

import javax.ejb.CreateException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;

import com.sap.aii.af.mp.module.*;
import com.sap.aii.af.ra.ms.api.*;

/**
 * @ejbHome <{com.sap.aii.af.mp.module.ModuleHome}>
 * @ejbLocal <{com.sap.aii.af.mp.module.ModuleLocal}>
 * @ejbLocalHome <{com.sap.aii.af.mp.module.ModuleLocalHome}>
 * @ejbRemote <{com.sap.aii.af.mp.module.ModuleRemote}>
 * @stateless
 */
public class CreateAttachment implements SessionBean, Module{

      private SessionContext myContext;

      public void ejbRemove() {
      }

      public void ejbActivate() {
      }

      public void ejbPassivate() {
      }

      public void setSessionContext(SessionContext context) {
            myContext = context;
      }

      public void ejbCreate() throws CreateException {
      }

      public ModuleData process(ModuleContext moduleContext,
                                ModuleData inputModuleData)
                                throws ModuleException {

         return inputModuleData;
      }
}
```

The comment lines starting with .*.@ are needed to enable the developer studio to create the ebj-jar.xml automatically.

3.3 Providing the EJB Environment

Your class should appear under *EJB Candidates*. Choose *Add to ejb-jar.xml* in the context menu.



The *ejb-jar.xml* file should now look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans
2.0//EN"
                         "http://java.sun.com/dtd/ejb-jar_2_0.dtd">
<ejb-jar>
    <description>EJB JAR description</description>
    <display-name>EJB JAR</display-name>
    <enterprise-beans>
        <session>
            <ejb-name>CreateAttachment</ejb-name>
            <home>com.sap.aii.af.mp.module.ModuleHome</home>
            <remote>com.sap.aii.af.mp.module.ModuleRemote</remote>
            <local-home>com.sap.aii.af.mp.module.ModuleLocalHome</local-home>
            <local>com.sap.aii.af.mp.module.ModuleLocal</local>
            <ejb-class>sample.CreateAttachment</ejb-class>
            <session-type>Stateless</session-type>
            <transaction-type>Container</transaction-type>
        </session>
    </enterprise-beans>
</ejb-jar>
```

In *ejb-j2ee-engine.xml*, you should assign a JNDI name. You need this name for the configuration of the module. An example of this is illustrated in the following code:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ejb-j2ee-engine SYSTEM "ejb-j2ee-engine.dtd">
<ejb-j2ee-engine>
	<enterprise-beans>
		<enterprise-bean>
			<ejb-name>CreateAttachment</ejb-name>
			<jndi-name>CreateAttachment</jndi-name>
			<session-props/>
		</enterprise-bean>
	</enterprise-beans>
</ejb-j2ee-engine>
```

After setting up the EJB environment, you receive the error message: *Bean problem: No interface classes found*. The reason for this is that no Java source is available for the EJB interface classes. Navigate to the developer studio's *Package Explorer* view and perform the following steps:

1. Select your project and choose *Close Project* in the context menu.
2. Choose *Open Project*.
3. Return to the *J2EE Explorer* view.

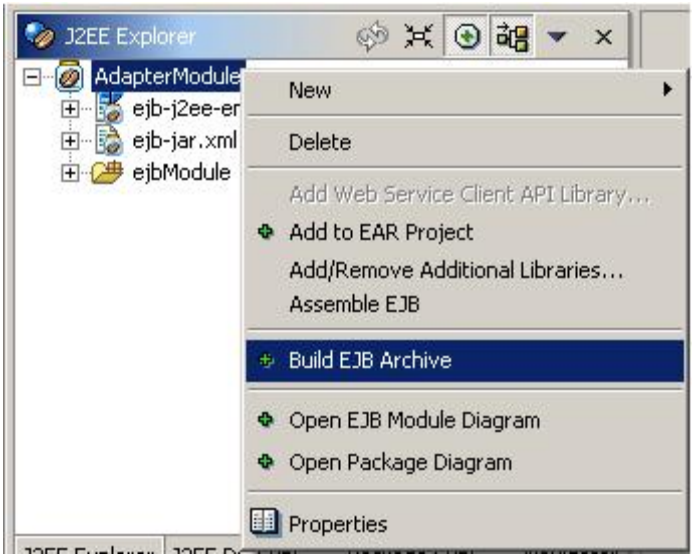
3.4 Adding Code to the Module

You can now add some short example code. You can change the *process* method for this purpose:

```
Public ModuleData process(ModuleContext moduleContext,
                          ModuleData inputModuleData)
                          throws ModuleException {

   // create a second attachment for the receiver mail adapter
   try {
            // get the XI message from the environment
            Message msg = (Message)
                    inputModuleData.getPrincipalData();
            // create a new payload
            TextPayload attachment = msg.createTextPayload();
            // provide attributes and content for the new payload
            attachment.setName("Attachment");
            attachment.setContentType("text/plain");
            attachment.setText("Hello World");
            // add the new payload as attachment to the message
            msg.addAttachment(attachment);
            // provide the XI message for returning
            inputModuleData.setPrincipalData(msg);
   } catch (Exception e) {
            // raise exception, when an error occurred
            ModuleException me = new ModuleException(e);
            throw me;
   }
   // return XI message
   return inputModuleData;
}
```

3.5 Creating a Java Archive

Select the package and choose *Build EJB Archive* in the context menu:



3.6 Creating an Enterprise Application Archive (EAR)

To deploy your adapter module to the Adapter Framework, you have to create an EAR file. Proceed as follows:

1. Create a new project as *J2EE → Enterprise Application Project*.
2. Select your EJB project(s) as reference.
   In this project, you need to assign references in *application-j2ee-engine.xml*. To do this, proceed as follows:
   1. Open the file.
   2. Select the source tab.
   3. Apply the following code.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE application-j2ee-engine SYSTEM "application-j2ee-engine.dtd">
<application-j2ee-engine>
        <reference
                reference-type="hard">
                <reference-target
                        provider-name="sap.com"
                        target-type="library">com.sap.aii.af.lib</reference-target>
        </reference>
        <reference
                reference-type="hard">
                <reference-target
                        provider-name="sap.com"
                        target-type="service">com.sap.aii.adapter.xi.svc</reference-target>
        </reference>
        <reference
                reference-type="hard">
                <reference-target
                        provider-name="sap.com"
                        target-type="service">com.sap.aii.af.svc</reference-target>
        </reference>
        <reference
                reference-type="hard">
                <reference-target
                        provider-name="sap.com"
                        target-type="service">com.sap.aii.af.cpa.svc</reference-target>
        </reference>
        <fail-over-enable
                mode="disable"/>
</application-j2ee-engine>
```

If you want to use additional libraries for your adapter module project, you can apply them here as well.

Select the project and choose *new* → *META-INF/log-configuration.xml*. This is useful if you want to add trace information to your module.
The following code is an example of the log-configuration.xml file:

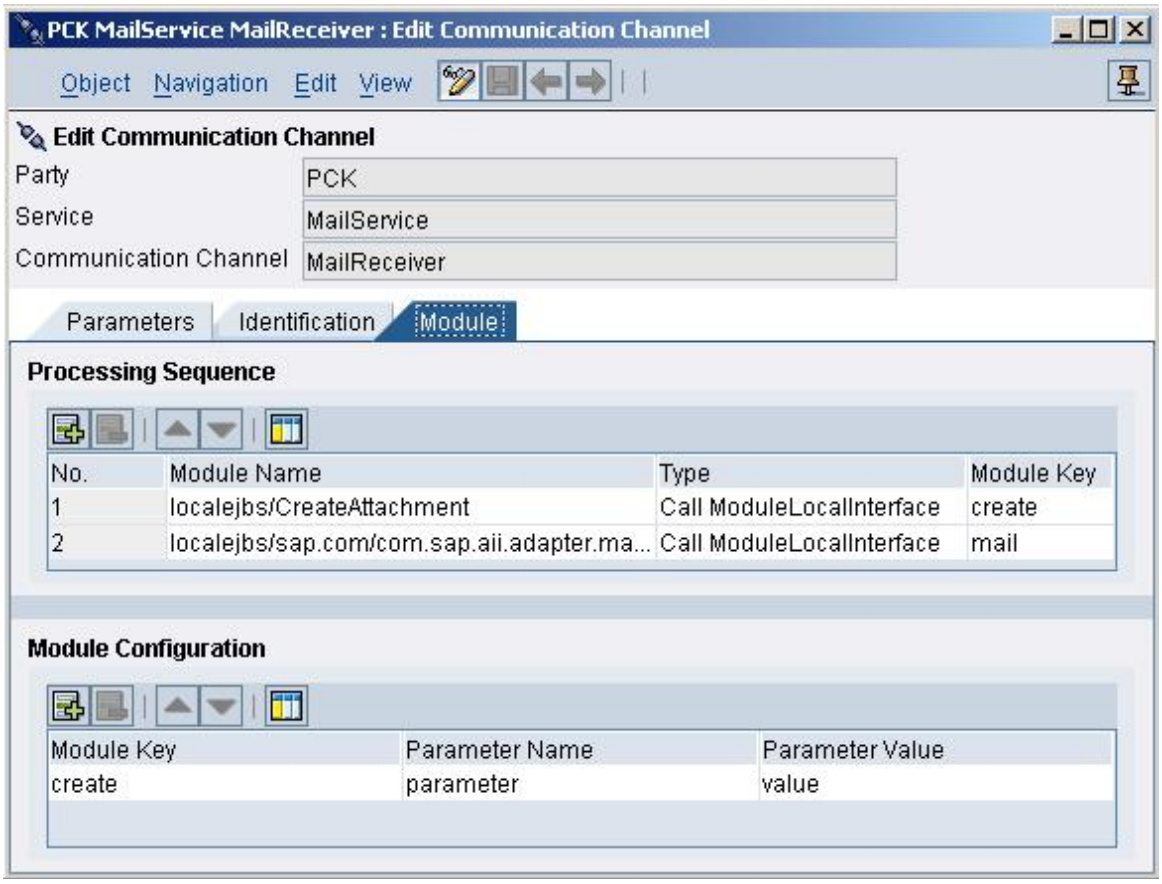
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log-configuration SYSTEM "log-configuration.dtd">
<log-configuration>
      <log-formatters>
            <!-- This formatter produces human readable messages. -->
            <log-formatter
                  name="trc"
                  pattern="%26d %150l [%t] %10s: %m"
                  type="TraceFormatter"/>
      </log-formatters>
      <log-destinations>
            <!-- Destination for Trace Information of this sample -->
            <log-destination
                  count="5"
                  effective-severity="DEBUG"
                  limit="2000000"
                  name="sample.trc"
                  pattern="./log/applications/sample/default.trc"
                  type="FileLog">
                  <formatter-ref
                        name="trc"/>
            </log-destination>
      </log-destinations>
      <log-controllers>
            <!-- Trace Location sample -->
            <log-controller
                  effective-severity="DEBUG"
                  name="sample">
                  <associated-destinations>
                        <destination-ref
                              association-type="LOG"
                              name="sample.trc"/>
                  </associated-destinations>
            </log-controller>
            <!-- Logging Category: none, use the default XILog -->
      </log-controllers>
</log-configuration>
```

Select the project and choose *Build Application Archive* in the context menu. Make sure that the libraries are not inside the generated EAR file, or delete them with WinZip. You can now deploy the EAR file by using the Visual Administrator deploy service, or by using the Software Deployment Manager (SDM).

3.7 Using the Module

If you want to use the module in an adapter, you need to open the Integration Directory and create a new communication channel.

1. Specify the adapter-specific data.
2. Choose the *Module* tab.
   Here you find a default entry with the required adapter module.
3. Apply your module before the specific adapter module.
4. Enter *localebjs/<JNDI-Name>* as the module name.
5. Choose *Locale Enterprise Bean* as the type.
6. Specify any module key.
   You can assign parameters to your module by using the module key.



4 Appendix

4.1 Documentation for Module Development

In the SAP Library for SAP NetWeaver 04, navigate to *SAP Exchange Infrastructure* → *Runtime* → *Connectivity* → *Partner Connectivity Kit* → *Adapter and Module Development*.

4.2 Documentation of the Resource Adapter API

To obtain a description of the classes of the Resource Adapter API, proceed as follows:

1. Download the newest patch for *XI ADAPTER FRAMEWORK 3.0* from SAP Service Marketplace.
2. Use WinZip to recursively unpack *sample_ra.sda* → *sample_ra.rar* → *sample_ra.jar*.
3. Extract the latter to your local PC.

You can now access the JavaDoc by opening the file *index.html* in a browser window.

4.3 Sample Module

You can obtain sample code for an adapter module (already deployed and ready for use) from the same resource mentioned above.
Recursively extract *sample_module.sda* → *sample_module.jar* → *ConvertCRLFfromToLF.java*.

4.4 Code Samples

4.4.1 Reading File Names

In the sender file adapter, you can access the name of the processed file.
Apply the following code to your module:

```
...
import java.util.Hashtable;
...

public ModuleData process(ModuleContext moduleContext,
                          ModuleData inputModuleData)
                          throws ModuleException {

   Message msg = (Message) inputModuleData.getPrincipalData();
   Hashtable mp = (Hashtable)
             inputModuleData.getSupplementalData("module.parameters");
   String fx = (String) mp.get("FileName");

   ...

}
```

4.4.2 Writing Audit Logs

To write to the audit log of the message monitor of the Adapter Framework, apply the following code:

```
...
import com.sap.aii.af.service.auditlog.*;
...

public ModuleData process(ModuleContext moduleContext,
                          ModuleData inputModuleData)
                          throws ModuleException {

   Message msg = (Message) inputModuleData.getPrincipalData();
   AuditMessageKey amk = new AuditMessageKey(msg.getMessageId(),
                                             AuditDirection.INBOUND);
   Audit.addAuditLogEntry(amk, AuditLogStatus.SUCCESS,
                          "CreateAttachment: Module called");


   ...

}
```

This adds an entry to the audit log as illustrated in the following example:

Audit Log for Message: 59e5c730-ee22-11d9-9a45-000bcd73d971

Time Stamp	Status	Description
2005-07-06 15:32:09	Success	The message was successfully received by the messaging system. Profile: XI URL: http://p120939:50000/MessagingSystem/receive/AFW/XI
2005-07-06 15:32:09	Success	Using connection AFW. Trying to put the message into the receive queue.
2005-07-06 15:32:09	Success	Message successfully put into the queue.
2005-07-06 15:32:09	Success	The message was successfully retrieved from the receive queue.
2005-07-06 15:32:09	Success	The message status set to DLNG.
2005-07-06 15:32:09	Success	Delivering to channel: MailReceiver
2005-07-06 15:32:09	Success	CreateAttachment: Module called
2005-07-06 15:32:09	Success	Mail: message entering the adapter
2005-07-06 15:32:09	Success	Mail: Receiver adapter entered with qos ExactlyOnce
2005-07-06 15:32:09	Success	Mail: calling the adpter for processing

Total: 16 Entries 10 messages displayed per page; this is page 1 of 2 page(s)

www.sap.com/netweaver