

**How-to Guide
SAP NetWeaver '04**



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

Version 1.00 – September 2005

**Applicable Releases:
SAP NetWeaver '04
Exchange Infrastructure 3.0**

© Copyright 2005 SAP AG. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP AG. The information contained herein may be changed without prior notice.

Some software products marketed by SAP AG and its distributors contain proprietary software components of other software vendors.

Microsoft, Windows, Outlook, and PowerPoint are registered trademarks of Microsoft Corporation.

IBM, DB2, DB2 Universal Database, OS/2, Parallel Sysplex, MVS/ESA, AIX, S/390, AS/400, OS/390, OS/400, iSeries, pSeries, xSeries, zSeries, z/OS, AFP, Intelligent Miner, WebSphere, Netfinity, Tivoli, and Informix are trademarks or registered trademarks of IBM Corporation in the United States and/or other countries.

Oracle is a registered trademark of Oracle Corporation.

UNIX, X/Open, OSF/1, and Motif are registered trademarks of the Open Group.

Citrix, ICA, Program Neighborhood, MetaFrame, WinFrame, VideoFrame, and MultiWin are trademarks or registered trademarks of Citrix Systems, Inc.

HTML, XML, XHTML and W3C are trademarks or registered trademarks of W3C[®], World Wide Web Consortium, Massachusetts Institute of Technology.

Java is a registered trademark of Sun Microsystems, Inc.

JavaScript is a registered trademark of Sun Microsystems, Inc., used under license for technology invented and implemented by Netscape.

MaxDB is a trademark of MySQL AB, Sweden.

SAP, R/3, mySAP, mySAP.com, xApps, xApp, SAP NetWeaver, and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP AG in Germany and in several other countries all over the world. All other product and service names mentioned are the trademarks of their respective companies. Data

contained in this document serves informational purposes only. National product specifications may vary.

These materials are subject to change without notice. These materials are provided by SAP AG and its affiliated companies ("SAP Group") for informational purposes only, without representation or warranty of any kind, and SAP Group shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP Group products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

These materials are provided "as is" without a warranty of any kind, either express or implied, including but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or non-infringement. SAP shall not be liable for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials.

SAP does not warrant the accuracy or completeness of the information, text, graphics, links or other items contained within these materials. SAP has no control over the information that you may access through the use of hot links contained in these materials and does not endorse your use of third party web pages nor provide any warranty whatsoever relating to third party web pages.

SAP NetWeaver "How-to" Guides are intended to simplify the product implementation. While specific product features and procedures typically are explained in a practical business context, it is not implied that those features and procedures are the only approach in solving a specific business problem using SAP NetWeaver. Should you wish to receive additional information, clarification or support, please refer to SAP Consulting.

Any software coding and/or code lines / strings ("Code") included in this documentation are only examples and are not intended to be used in a productive system environment. The Code is only intended better explain and visualize the syntax and phrasing rules of certain coding. SAP does not warrant the correctness and completeness of the Code given herein, and SAP shall not be liable for errors or damages caused by the usage of the Code, except if such damages were caused by SAP intentionally or grossly negligent.

Content

1	Scenario	2
2	Introduction.....	2
3	The Step-By-Step Solution.....	3
3.1	Setting Up the Developer Studio	3
3.2	Creating the Java Class	5
3.3	Providing the EJB Environment.....	6
3.4	Adding Code to the Module	8
3.5	Creating a Java Archive	8
3.6	Creating an Enterprise Application Archive (EAR)	9
3.7	Using the Module	11
4	Appendix	12
4.1	Documentation for Module Development	12
4.2	Documentation of the Resource Adapter API.....	12
4.3	Sample Module	12
4.4	Code Samples.....	12
4.4.1	Reading File Names	12
4.4.2	Writing Audit Log Entries	13

1 Scenario

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

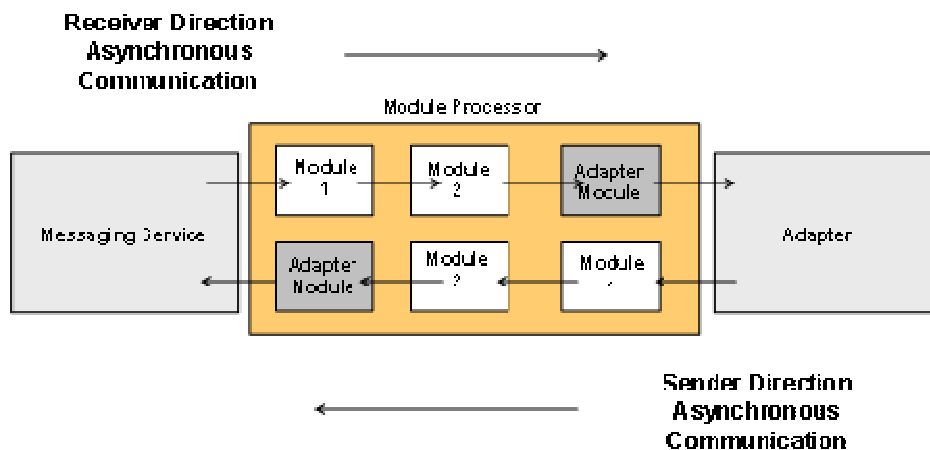
2 Introduction

A message from the Integration Server arrives in the Adapter Framework and is processed 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. In this case, 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. In the case of the latter, the conversion to an XI message must then take place in an adapter-specific module.



Configuration of the 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 relevant location in the module processor.

3 The Step-By-Step Solution

This manual guides you through the development of a simple adapter module with the help of the SAP NetWeaver Developer Studio. Make sure that you use the latest version of the Developer Studio.

By way of example, we explain how to develop an adapter module that creates an additional attachment. This is something that might be useful in the mail adapter.

3.1 Setting Up the Developer Studio

You need several libraries to compile your module.

The easiest way of getting these libraries is by downloading the latest patch for *XI ADAPTER FRAMEWORK CORE 3.0* from *SAP Service Marketplace*.

Open the file using *WinZip* and extract `aai_af_lib.sda`, `aai_af_svc.sda` and `aai_af_cpa_svc.sda`.

Open the file `aai_af_lib.sda` using *WinZip* and extract the following library files:

- `aai_af_cci.jar`
- `aai_af_mp.jar`
- `aai_af_ms_api.jar`
- `aai_af_ms_spi.jar`
- `aai_af_trace.jar`

Extract the following library file from `aai_af_svc.sda`:

- `aai_af_svc.jar`

Extract the following library file from `aai_af_cpa_svc.sda`:

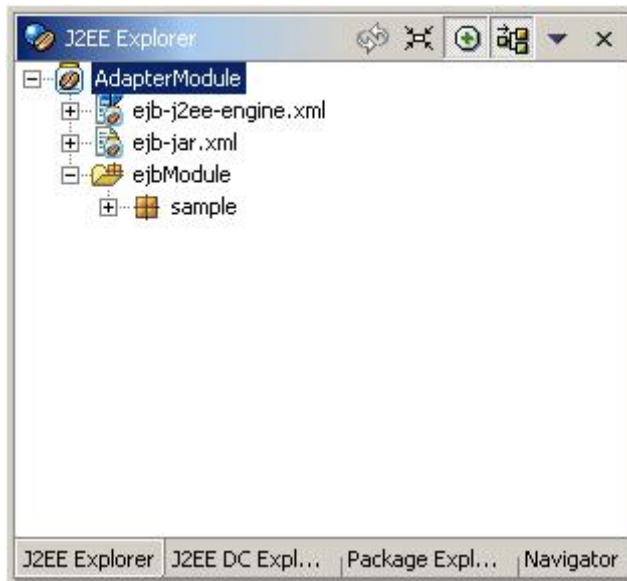
- `aai_af_cpa.jar`

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

Steps in SAP NetWeaver Developer Studio

- Open *SAP NetWeaver Developer Studio*.
- Create a new *Project* as **J2EE** -> **EJB Module**.
- Choose *Project* -> *Properties* -> *Java Build Path* -> *Libraries* to assign the libraries to your project.
- Create a package for your Java classes.

Your *J2EE Explorer* view in Developer Studio should look as follows:



3.2 Creating the Java Class

Even if you need a stateless session bean as an adapter module, it is useful not to let the Developer Studio create it. Instead, create a new java class and attach the following code template to the class (check that the package and class name 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 {
    }

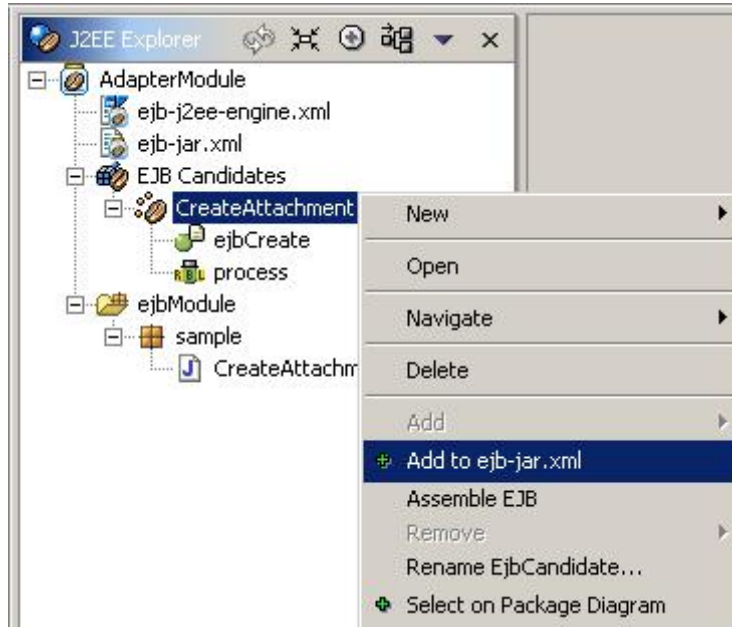
    public ModuleData process(ModuleContext moduleContext,
                              ModuleData inputModuleData)
        throws ModuleException {

        return inputModuleData;
    }
}
```

Comment lines such as `@ejbHome` are required to enable the Developer Studio to create the `ejb-jar.xml` file automatically.

3.3 Providing the EJB Environment

Your class should appear in the navigation tree under *EJB Candidates*. In the context menu, choose *Add to ejb-jar.xml*.



ejb-jar.xml should look as follows:

```
<?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>
```


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

```
<?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*. This occurs because there is no Java source available for the EJB interface classes.

- Choose the *Package Explorer* view in the Developer Studio.
- Choose your project, and in the context menu, choose *Close Project*.
- Choose *Open Project* again.
- Finally, switch back to the *J2EE Explorer* view.

3.4 Adding Code to the Module

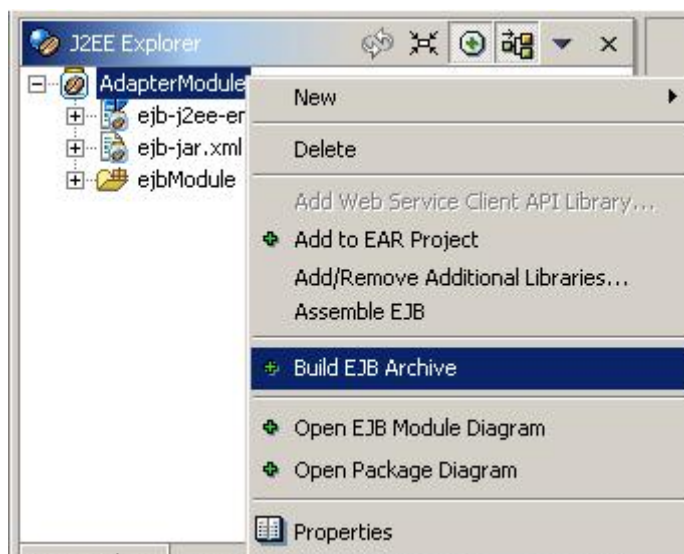
You can now add a short example code. You can change the *process* method for this purpose:

```
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 in the context menu, choose *Build EJB Archive*, as shown in the following figure:



3.6 Creating an Enterprise Application Archive (EAR)

In order to deploy your adapter module to the J2EE Adapter Framework, you have to create an EAR file:

- Create a new project as *J2EE -> Enterprise Application Project*.
- Select your EJB project(s) as reference.
- In this project, you need to assign references in *application-j2ee-engine.xml*.
- Open this file, choose the source tab page, and 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 may need to apply them here as well:

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

```
<?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 in the context menu, choose *Build Application Archive*. Make sure that the libraries are not inside the generated EAR file, or delete them using *WinZip*.
- Deploy the EAR file from the deploy service in the Visual Administrator tool, or with the Software Deployment Manager (SDM).

3.7 Using the Module

- If you want to use the module in an adapter, open the Integration Directory and create a new communication channel.
- Enter the adapter-specific parameters and choose the *Module* tab page. A default entry contains the required adapter module.
- Apply your module before the specific-adapter module:
 - Enter *localejbs/<JNDI-Name>* as module name.
 - Choose the type *Locale Enterprise Bean*.
 - Choose any module key. Using your module key, you can assign parameters to your module in the *Module Configuration* table.

The screenshot shows the 'Edit Communication Channel' window for 'PCK MailService MailReceiver'. The 'Module' tab is selected, displaying the following data:

Processing Sequence

No.	Module Name	Type	Module Key
1	localejbs/CreateAttachment	Call ModuleLocalInterface	create
2	localejbs/sap.com/com.sap.aii.adapter.ma...	Call ModuleLocalInterface	mail

Module Configuration

Module Key	Parameter Name	Parameter Value
create	parameter	value

4 Appendix

4.1 Documentation for Module Development

- In the SAP Help Portal, choose the *Documentation* tab page.
- Choose the *SAP NetWeaver* tab page, then *SAP NetWeaver '04* and *English* as the language.
- Navigate to *SAP NetWeaver -> Process Integration -> SAP Exchange Infrastructure -> Runtime -> Connectivity -> Partner Connectivity Kit -> Adapter and Module Development*.

4.2 Documentation of the Resource Adapter API

- When you download the latest patch for *XI ADAPTER FRAMEWORK 3.0* from *SAP Service Marketplace*, you receive a description of the classes of the resource adapter API.
- Using *WinZip*, recursively unpack `sample_ra.sda -> sample_ra.rar -> sample_ra.jar`. Extract the latter to your local PC. You can now access the JavaDoc by opening the `index.html` file in a Web browser.

4.3 Sample Module

You receive sample code for an adapter module (which is already deployed and ready to 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 file/FTP sender 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 Log Entries

To write to the audit log of the message monitor of the Adapter Framework, you 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 results in an entry in the audit log:

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 adapter for processing

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

www.sdn.sap.com/irj/sdn/howtoguides