MQSeries Adapter Offering – Baan IV Template
Version 1.0

9th January 2002

Neal Mulvenna
IBM
8501 IBM Drive
Charlotte, NC
28262

mulvenna@us.ibm.com

Property of IBM

Take Note!

Before using this report be sure to read the general information under "Notices".

First Edition, January 2002

This edition applies to Version 1.0 of *MQSeries Adapter Offering – Baan IV Template* and to all subsequent releases and modifications unless otherwise indicated in new editions.

Table of Contents

Notices

The following paragraph does not apply in any country where such provisions are inconsistent with local law.

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates.

Any reference to an IBM licensed program or other IBM product in this publication is not intended to state or imply that only IBM's program or other product may be used. Any functionally equivalent program that does not infringe any of the intellectual property rights may be used instead of the IBM product.

Evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, is the user's responsibility.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, New York 10594, USA.

The information contained in this document has not be submitted to any formal IBM test and is distributed AS-IS. The use of the information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While IBM has reviewed each item for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

Trademarks and service marks

The following terms, used in this publication, are trademarks of the IBM Corporation in the United States or other countries or both:

- AIX
- IBM
- MQSeries
- MQSeries Adapter Offering
- MQSI

The following terms are trademarks of Microsoft Corporation:

- Windows NT
- Windows 2000

The following terms are trademarks of Invensys Corporation:

- Baan

The following terms are trademarks of Sun Microsystems Inc:

- Java
- JDK (Java Development Kit)

Summary of Amendments

Date	Changes
January 2002	Initial release

Preface

MQSeries Adapter Offering provides a “drag and drop” facility that simplifies the development of mappings from one data format to another. This SupportPac is a simple example of a resulting adapter set that maps from Baan IV objects to Open Applications Group’s Business Object Documents (BODs).

The MQSeries Adapter Builder (MQAB) “repository” that was created and used to generate the adapter set is also included, and may be used via the MQAB tool to extend the adapter set.

Baan IV does not include a “push” mechanism for externalizing event-triggered data changes. This SupportPac includes a custom Baan program and sample database triggers that, utilized together, provide the push mechanism.

This SupportPac includes:

- Four adapters created by the MQAB and generated in Java code.
    - One that maps from the BaanIV Customer object (externalized as XML) to an Open Applications Group’s SyncCustomer BOD (XML).
    - One that maps from the BaanIV Item object (XML) to an Open Applications Group’s SyncItem BOD (XML).
    - One that maps from the BaanIV Inventory object (XML) to an Open Applications Group’s SyncInventory BOD (XML).
    - One that maps from the BaanIV SalesOrder object (XML) to an Open Applications Group’s SyncSalesOrder BOD (XML).
- Four sample database triggers for the above BaanIV objects.
- Instructions for adding a new “triggered action” table to the Baan database.
- A custom Baan 4GL program and instructions for its installation and use. This program is a daemon that runs as a Baan batch “job”. It inspects the triggered action table for new events, retrieves necessary data from the Baan database tables, and externalizes the data as XML documents in a flatfile.
- A Java utility daemon which reads XML documents from a flatfile and utilizes the MQSeries Adapter Offering Runtime Kernel (MQAK) to place the XML on an MQSeries queue along with some routing header data.
- The BaanIV XML DTDs that correspond to the output XML of the custom Baan program, and are used in the input data context mappings within the adapters.
- A subset of the Open Applications Group DTDs (version 7.1). They are used in the output data context mappings within the adapters.
- The MQAK setup and configuration files modified for running the above adapters and the utility daemon.
- Sample Baan XML data to allow trying the adapters without a Baan IV system.
- Data mapping and Baan table documents.

Note: Baan does not provide XML versions of their Baan IV objects. The Baan objects DTDs were created by IBM.

Chapter 1. Overview

Function Overview

IBM's MQSeries Adapter Offering (MQAO) provides a powerful solution to the challenge of formatting and reformatting data. In its simplest form, MQAO takes a description of a message format (layout) and, when presented with messages in this format, can break apart that message into its constituent fields and assemble a message that matches the outbound format.

Many transformations involve input and/or output messages in an XML format. In many cases, a definition of the fields in the XML message is available in a Document Type Definition (DTD) file. MQAO provides a GUI interface that allows DTD files to be imported into the metadata repository. Once this is done it is possible to use the MQAO GUI to build mappings from one format to another. Through the use of Input/Output Terminals the messages can be read from or written to files or queues in the case of message-oriented-middleware, or any other form of middleware.

MQAO also supports MQSI compute node transformations that can be built using drag and drop operations and where needed, it is possible to script additional logic by making use of the ESQL scripting language.

MQAO includes support for database nodes for storing and retrieving lookup tables and message logging.

For the MQSeries Adapter Offering – Baan IV Template the input terminals are the custom Baan XML and output terminals are the Open Applications Group's BOD XML. The mappings have been set to a set of fields that allow visibility to the different mapping capabilities of the MQSeries Adapter Offering.

Also, while MQAO can be fully integrated into the rest of the MQSeries Family of products this SupportPac does not take advantage of that integration in order to show the capabilities of MQAO.

Chapter 2. Prerequisites

System Requirements

In order to use this SupportPac the following must be installed on a system:

- Windows 2000 or AIX V4.3
- The adapters and adapter utilities have been generated in Java and require a minimum of JRE 1.1.8 to run, preferably JRE 1.2.2 or SDK 1.2.2.
- MQSeries V5.1 or V5.2 Server with the latest CSD, and the MA88 SupportPac (MQSeries classes for Java and MQSeries classes for Java Message Service).
- The MQAOJFramework.jar, shipped with the MQAB product.
- MQSeries Adapter Kernel with its latest CSD.
- Any modification of the adapters will require the full MQSeries Adapter Builder product with its latest CSD.
- A BaanIV ERP system (although sample output Baan data is included to demonstrate the adapters and utility)

Skill Level Required

1. Setup and Demonstration of the MQAB adapters and MQAO utilities using the supplied sample data alone, requires the following skills:
    - MQSeries administration
    - MQAK administration
    - Some Java knowledge
    - Ability to set up environment variables
2. Installation of the Baan custom program and the related Baan database table reguires a BaanIV programmer with training in Baan Tools.
3. Installation of the sample database triggers requires a DBA.

Supported Platforms

This SupportPac should be unzipped on a Windows NT or 2000 platform. The included MQAB repository should only be used there.

The executables may be run on the above platforms or any platforms that MQAK can be run on.

However, they have only been tested on a Windows 2000 and an AIX 4.3 platform.

Chapter 3. SupportPac Contents

Contents of the zip file

This SupportPac is delivered as a single file in a standard zip format. When the file is unzipped, several folders and files should be produced, as follows:

• aa01.pdf	This document
• license2.txt	License file
• MQBaanAdapters.jar	Jar file containing all the adapter executables
• MqJGet.class	Simple java utility described in chapter 4
• Interfaces (folder)	Contains the Java utility daemon in a subfolder
• MQAB (folder)	Contains the MQAB repository and generated adapters
• runtimedata (folder)	Contains MQAO configuration and command files
• Baan programs (folder)	Contains the custom Baan 4GL program and action table info
• DTDs (folder)	Contains all related DTDs in two subfolders
• Mappings (folder)	Mapping documents and Baan database object reports
• Sample data (folder)	Contains sample BaanIV output XML to feed the adapters
• Sample triggers (folder)	Contains sample database triggers in a subfolder
• unix (folder)	Contains several unix/AIX files/utilities/scripts.

Content Details

The MQBaanAdapters.jar should be placed in the user’s CLASSPATH, along with the MQAB and MQAK necessary executables (not included in this SupportPac).

The runtimedata folder contains several .cmd files, which can be used on a windows platform to prepare and start the adapters.

The MQAB folder contains the export form of the MQAB repository for the Baan4 adapters. This is file Baan4.zip. DO NOT UNZIP BAAN4.ZIP! It should be imported as is into MQAB. All other data and subfolders in the MQAB folder were created by MQAB during adapter generation.

Using the Daemons

There are three daemon programs involved in this solution:

1. The Baan 4GL custom program. This program inspects the (new) triggered action table in the Baan database on a periodic basis (time interval set by the user) and externalizes all the related data as a flatfile. This program must be installed in Baan, the appropriate environment variables set, and the daemon started from Baan. The source code is supplied so that the user is free to make modifications as desired.

2. The Java utility. This program looks for the flatfile created by the Baan program, and processes the XML message(s) to an MQSeries queue. The program uses values in the (extended and supplied) MQAK aqmsetup file to locate the flatfile. A .cmd file is supplied to start this daemon.

3. The standard MQAK daemon, which reads, messages from a queue and calls the appropriate MQAO adapter to process a message. A .cmd file is supplied to start this daemon.

The MQAO Adapter daemon and supplied utility daemon are Java classes that can be used to execute the Java adapters used in the data transformation. They must be run on a system with Java JDK 1.2.2 or better.

OAGIS Business Object Definitions (BODs)

The open applications group has developed definitions of a number of common business objects. They have also defined a number of XML formats for these objects. These definitions are available as DTDs that can be downloaded from the open applications group website (http://www.openapplications.org). There are close to 200 definitions available as of this writing and more are being added.

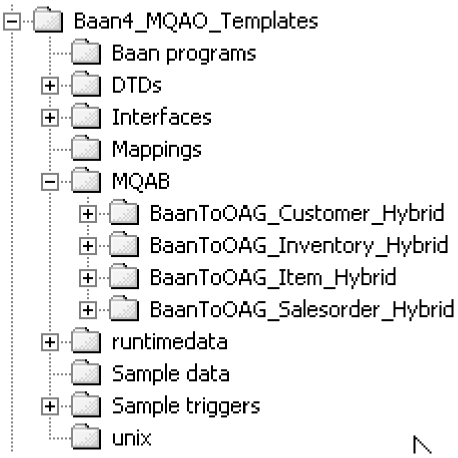
Using the MQAOBaan4Template zip file:

Important Note:

It is assumed that you have installed the Java Development Kit 1.2.2 and the IBM MQSeries Adapter Kernel for Multiplatforms Version 1.1

Unzip the file and extract to your **C:\** directory. Ensure that you extract *using folder names*. This will ensure that all directory structures are extracted properly. If you extract to any other drive, you must make changes to all the configuration files and .cmd files in the runtimedata subdirectory to reflect the new drive or base directory. *set_env_variables.bat* and *aqmsetup*. In each of these files you must change the drive letter **C:\** to your the drive letter where you extracted the zip file.

Verify the extraction. You directory structure should be like the figure below.



- Baan4_MQAO_Templates
  - Baan programs
  - DTDs
  - Interfaces
  - Mappings
  - MQAB
    - BaanToOAG_Customer_Hybrid
    - BaanToOAG_Inventory_Hybrid
    - BaanToOAG_Item_Hybrid
    - BaanToOAG_Salesorder_Hybrid
  - runtimedata
  - Sample data
  - Sample triggers
  - unix

Chapter 4. Setting up the pieces – Stand-alone adapters test

The adapters may be exercised without the Baan application by using the supplied sample Baan output XML data. The center of operations is the runtimedata folder. All the .cmd files and the config and setup files assume this zipfile was unzipped into the base C: device root directory. If not, then they all need to be updated to reflect the actual unzip location.

Setting up MQSeries

Inspect the aqmconfig.xml file in the runtimedata folder (or the one in the unix folder if setting up on an AIX machine) to note the MQSeries queue names that are used. (They are: HeaderQ, NoHeaderQ, OAGOutQ, GENERIC_AEQ, GENERIC_RPL, TraceServerAIQ). These queues must be created in MQSeries before proceeding. They can all be local queues for the exercise.

Starting the flatfile processor daemon

The daemon program is com.ibm.bpac.baan.util.AddEpicHdrToBaanDaemon. You specify the cycle time in milliseconds as a parameter. Use the Start_Baan_flatfile_processor.cmd to start it. Either double-click on it or start it from a separate DOS Command prompt window. Some print statements are active, so you can see when it processes flatfile data. Press <Enter> to end the daemon.

Starting the MKAK adapter daemon

The daemon is the aqmstrad utility. Use the Start_Baan_adapters.cmd to start it, as above. Press <Cntrl>C to end it.

Using the sample data

Use any of the “Copy…..cmd” commands to copy sample data to the location and filename that the flatfile daemon is looking for it. Repeat at will.

Inspecting the results

The flatfile daemon places messages on the MQSeries HeaderQ queue, and in the case of Item data only, also on the NoHeaderQ queue. If the MQAK daemon is not running, you can check these queues for content. A simple Java utility, MQJGet, is included in the base directory to get a message from the specified queue and place it in a flatfile, so the message can be inspected.

The MQAK daemon reads messages from the HeaderQ, calls the appropriate adapter, and places the transformed message on the OAGOutQ queue. Inspect these messages also.

Chapter 5. Setting Up the Pieces – Baan Application

Creating the Baan database Action Table

The TQPACT001ccc table should be set up in the Baan database (ccc = Baan company to use). Note that only one table for one company needs to be built and maintained even if you are using multiple companies - The logic is set to handle that.

Use Baan Tools to create the table. See qpact001.rpt in the Baan programs folder for the exact structure to be created. (view with wordpad).

Installing the Custom Baan Daemon Program

Program overview:

The custom Baan daemon program inspects the TQPACT001ccc table for new table rows.

For each row found, additional data is retrieved from the normal Baan tables, the data is formatted into a valid XML string (DTDs supplied, but not used in the program), and the string is placed on a flatfile, so that the MQAO pre-adapter program can process it.

The table row is then deleted.

Installation:

A Baan programmer familiar with Baan Tools can install the Baan daemon program.

The program supplied, qpact2001, was set up after creation of a new Baan package of "QP". As above with the action table, this can be any valid package, new or existing. If something else is used, appropriate renaming must be done before proceeding.

When ready to install, first use Baan Tools to create a new GUI program, of type = 4. Specify the qpact001 table as its table. Once that is set up, locate the Baan generated source code for it in the proper Baan VRC directory and copy the supplied qpact2001 to that directory. Rename (replace) to match (if qpact2001 name is kept, the source code will be "pact20010"), then compile.

Setting the Environment Variables

Some of the following environment variables are unique to the custom Baan daemon program; A few are also used by the MQAO pre-adapter utility which is written in java. As such, it does not directly use the environment variables, rather a copy of them placed in the active MQAO aqmsetup file.

If on a unix platform, be sure to set these environment variables for the userID that will be used to start the Baan daemon.

```
#################################
# Env vars needed by custom Baan MQAO integration programs
# which should match values in aqmsetup (which Baan does not access)
# The BAANOUT_DIRECTORY value should end with a delimitter.  And if on a Windows platform,
# the backslashes should be doubled.
export BAANOUT_DIRECTORY=/home/mqao/runtimedata/outbound/
# windows platform example:
export BAANOUT_DIRECTORY= C:\\baan\\data\\outbound\\
# Note that in aqmsetup, the ending delimitter should NOT be present, and on the NT,
#      the backslashes should NOT be doubled.
# The BAAN_PASS_FILENAME value should match exactly with that in aqmsetup or the file will not
be found
export BAAN_PASS_FILENAME=BaanOutputXML
### Following also used by custom Baan programs but do not go into aqmsetup
export BAAN_BUILDING_FILENAME=BaanUseOnly
export BAAN_COMPANY=556
export BAAN_INVENTORY_LVL=CNTR
export BAANOUT_WAITTIME=1
export BAANOUT_CYCLES=10000
##################################
```

Installing the Baan Database Triggers

The database triggers should be installed by a DBA. Proper permissions should be set such that they will be fired whenever any Baan system user is updating Baan objects.

The sample database triggers supplied are for an Oracle database. They should be inspected and changed as appropriate to reflect the Baan company(s) used. As each Baan Company has its own set of tables on the database, additional triggers will need to be created and installed per company.

Note: Other databases (e.g., DB2) may have slight variations to the trigger logic in the area of date/timestamp, and in the prefix used on column names. On Oracle, Baan uses a “T$” prefix, for most other databases, a “T_” prefix is used instead. As an example, the Customer field in Baan is referred to as “cuno” within the Baan application. On the database, it’s actual column name is either t$cuno or t_cuno, depending on the RDBMS. Also note that every table in Baan has two hidden (from the user) columns, refcntd and refcntu, for referential integrity. Although not actually used on the triggered action table, they must be present.

Study the sample database triggers to understand what they do. Note that only the identifying object key data is inserted into the action table. The Baan custom program uses that data to retrieve all other data from the standard Baan tables. Except of course for a “delete” action, whence just the key data is externalized.

Starting the Baan custom program daemon

The following instructions are for starting the daemon from a Baan “green screen” session rather than the GUI interface. If Baan is running on a Windows platform, use a DOS Command Prompt window. If Baan is on a unix (AIX) platform, either open a telnet session or work from an X-station. The CLASSPATH and PATH environment variables must be properly set up for accessing Baan.

- `(if on a unix platform)Export TERM=vt100`
- `ba6.1` (to start a Baan ASCII session)
- `p` (on the main menu Choice: to allow entry of a session name)
- `qpact2001m000, return, esc,Y` (on the session Choice:_)

The daemon should now be running. Do not attempt to use the session for other purposes.

Inspecting the daemon output

If the database triggers are properly installed and have been fired by a user saving updates to the Baan objects in a Baan session (GUI or ASCII version), the daemon will create/append to a flatfile named in the BAAN_PASS_FILENAME environment variable, in the directory specified in the BAANOUT_DIRECTORY environment variable. Use any text editor to view the contents.

Ending the Baan custom program daemon

The daemon can end in a couple ways:

- It can run out of processing cycles. If the environment variables were set as above, this would occur approximately 10,000 seconds (27.8 hours) after it was started.

- It can be stopped by putting a file named “STOP“ in the directory pointed to by the BAANOUT_DIRECTORY environment variable. When the daemon finds a file with this name, it will delete that file, then stop.

When the daemon has stopped, the Baan session should also be closed (enter “A” in the “Choice: _” field. Repeat as necessary until totally out of Baan.

Chapter 6. Reviewing the MQAB Adapters

This chapter is not intended to be a tutorial on creating an adapter in MQAB. There are excellent MQAB tutorial materials for that on the IBM MQSeries website.

Setup

This chapter requires the MQAB tool to be installed and assumes the user has at least gone through the tutorial for creating Java adapters.

To load the Baan IV adapters into your MQAB:

1. Start MQAB

2. File -> Import Workspace, and import the …\Baan4_MQAO_Templates\MQAB\Baan4.zip file as-is.

3. File -> Open Workspace, and choose Baan4.xml.

4. Using the included mapping documents in the Mappings folder, inspect the message flows and the map block properties to see what the adapters are doing.

Looking at an MQAB Adapter

All four MQAB adapters in this support pac are called “hybrid” adapters rather than being labeled as “source” (called as an API by some application) or “target” (call some application API). These adapters all take an XML message off an MQSeries queue, transform it to another XML format, and place the transformed message on another MQSeries queue.

The following figures and text provide an overview of the most complex of the adapters, the SalesOrder.

The darker connector lines in the figures are the “control connection” lines, and will appear as blue lines in the Builder tool. The lighter lines (gray in the Builder) are “data connection” lines. The annotations by the small map block figures (e.g., *m1*) were added to these figures for ease of explanation and do not necessarily match to the block name actually used in the tool.

Message Sets

All the pertinent Baan and OAG DTDs (supplied) have been imported into MQAB as message sets. Click on the “+” signs to expand the Baan and OAG salesorders and note their structures. The Baan XML is straightforward and parallels the Baan Salesorder object as it exists in the Baan database. Inspection of the salesorder header and line database tables (tdsls040, tdsls041) will show that the XML tags match to the column names (except regarding capitalization and sequence – the author chose to place the tags in alphabetic sequence rather than table column order). The OAG message has a more complex structure,allowing for many possibilities in mapping.

For the salesorder adapter, the basic mapping is Baan salesorder header (table tdsls040 or XML tag SALESHDR) maps to the OAG salesorder header section (XML tag SOHEADER) as a singlet, and the multi-rows of the Baan salesorder lines (table tdsls041 or XML tag SALESLINE) map to the OAG salesorder line section (XML tag SOLINE) as repeating elements.

Two notes on importing DTDs:

1. The repeating elements (that will be used) should always be checked after import to ensure that they are flagged as repeating (check the Connection tab under Properties for the particular element, e.g., SALESLINE), and if not, manually change the setting.

2. For a DTD that is spread across multiple files (as is the case with OAG), only the file containing the root element needs to be imported. The included other files will be pulled in by the tool.

Salesorder_Java_SF

The Salesorder_Java_SF microflow shown in Figure 1 is the top-level connector flow for the SalesOrder adapter. Map *m1* in the figure (Map_Baan4_XML_To_OAG_XML in the tool) is where the sales order header data is mapped from Baan to OAG. Note also the literal values that are mapped there. Map *m2* in the figure (Map 1 in the tool – sorry about the numbering confusion) does little more than show the relationship of the Address line to the XML, but is necessary to get the actual data to be propagated to the Map_Salesorder_Addrline microflow (discussed later). Likewise, map *m3* in the figure (Map 2 in the tool) is used to flow the salesorder line data to the Iteration_Salesorder_Soline microflow. Maps m4 and m5 in the figure perform standard functions as described in the MQAB tutorial. Note that care should be taken in choosing the literal values to use in m5, especially mapping the source and destination logical Id's. All the values are placed in the outgoing message header and compared against values in the aqmconfig file at runtime to determine what happens next to the message.

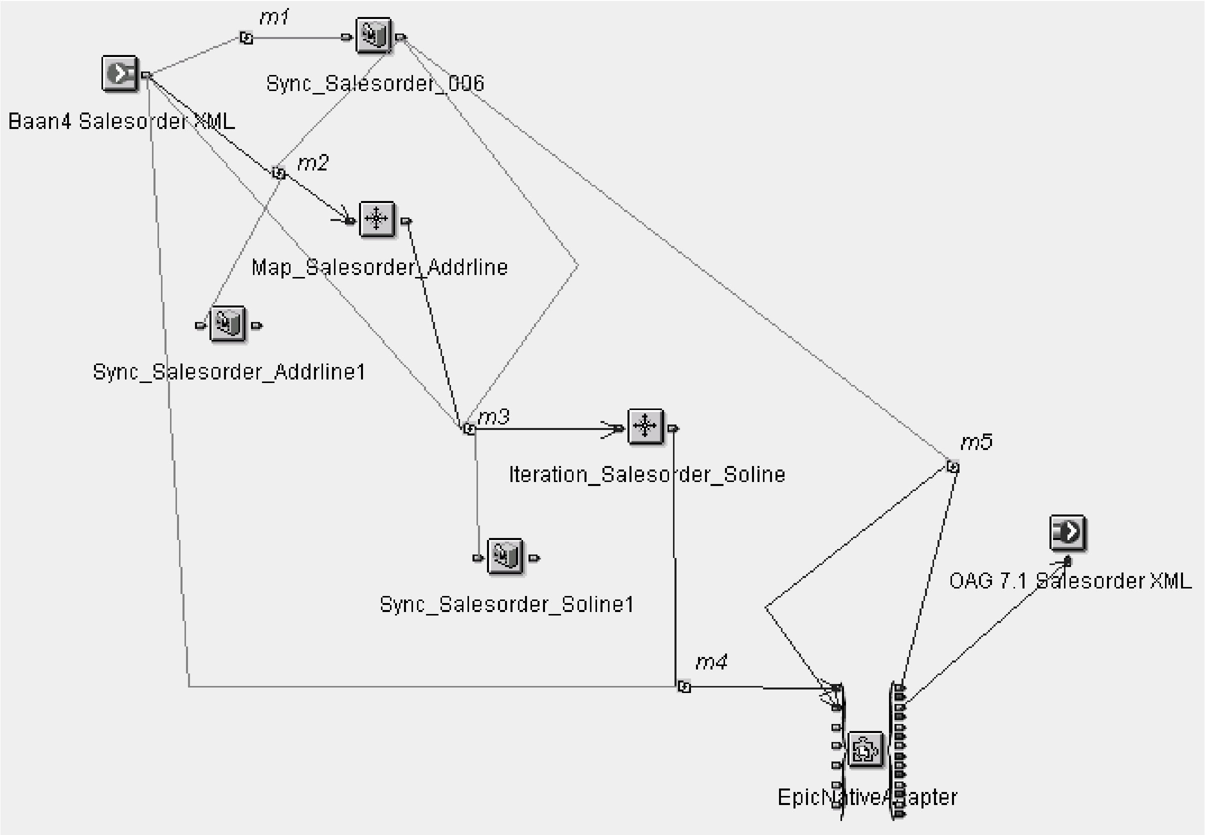


Figure 1: Salesorder_Java_SF microflow

The outgoing message is not tied to any synchronous response so the wiring of the EpicNative Adapter node is straightforward. Note that the Baan salesorder DTD is the Message Set specified in the flow's input node and the OAG salesorder DTD is specified in the output node. Also note that the data connector line from the input node to map *m4* is not necessary, but all other connections are.

Map_Salesorder_Addrline

The addressline microflow shown in Figure 2: Map_Salesorder_Addrline microflow is necessary because addrline is a repeating element within the OAG XML's SOHEADER.PARTNER.ADDRESS element, of which two instances are used, each mapped from a different element in the Baan XML (see maps *ma1*, *ma2* properties). This is not iteration, but mapping to multiple instances does require extra effort best done in a separate microflow. For an even busier microflow, look at the Map_Item_Property microflow in the tool.

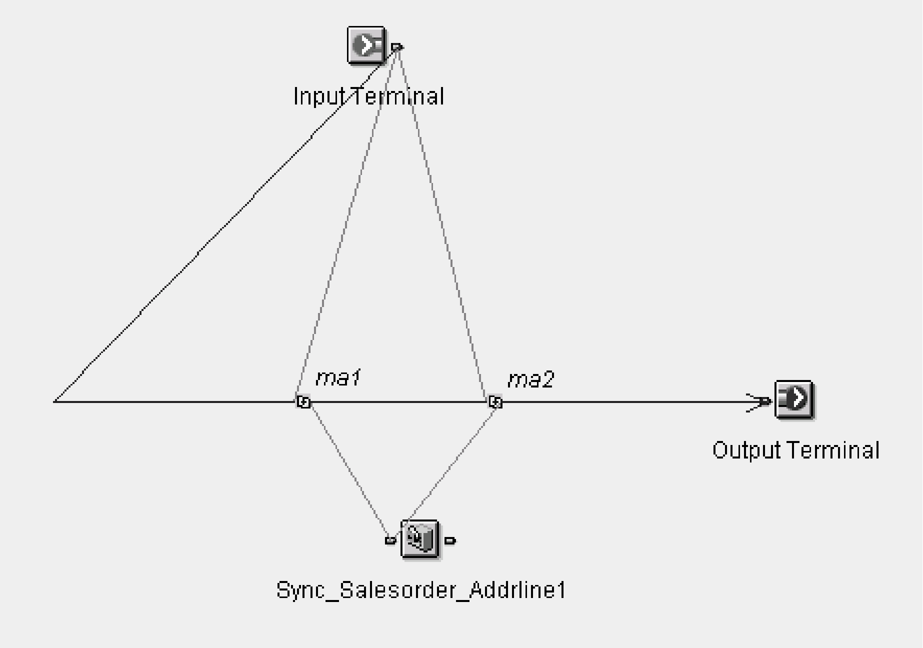


Figure 2: Map_Salesorder_Addrline microflow

Map_Salesorder_Soline

Figure 3 shows the microflow where the mapping of an individual Baan salesorder line to an OAG SOLINE occurs (see map *mL1* in the tool). This microflow becomes a node in the iteration flow that follows.

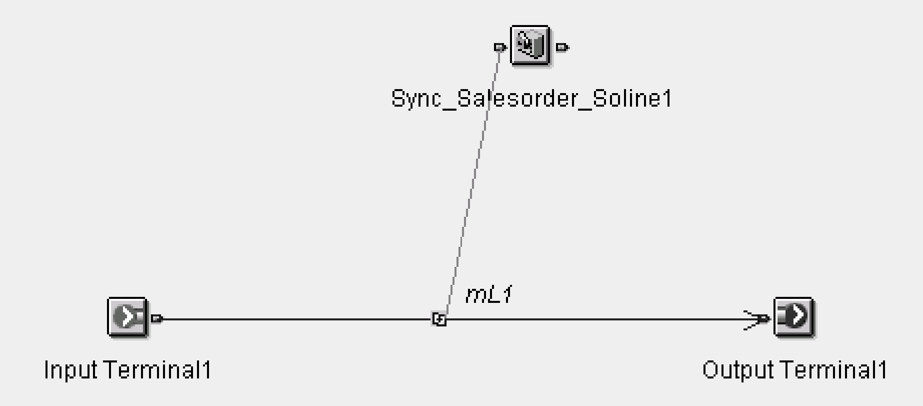


Figure 3: Map_Salesorder_Soline microflow

Iteration_Salesorder_Soline

Figure 4 is the sales order line iteration microflow. This is similar to the iteration example in the MQAB tutorial.

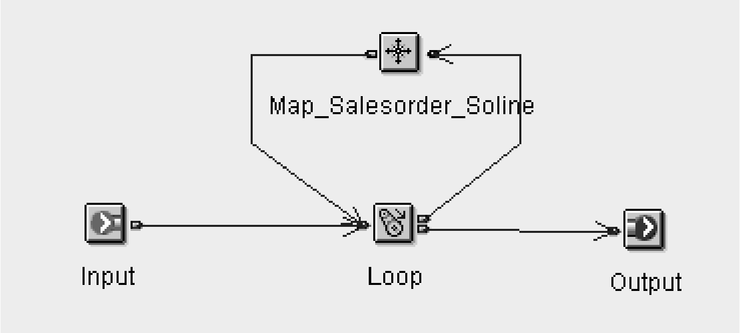


Figure 4: Iteration_Salesorder_Soline microflow

Chapter 7. Problem Determination

Reporting problems

The author is interested in any reports of problems encountered when using this SupportPac. The author's email address is given on the front page of this document. To assist in problem diagnosis, the author will probably require the following:

- Description of the problem
- General description of the environment
- Screenshots or trace files indicating Java runtime errors

The author will attempt to provide a solution to problems that are reported.

End of Document