HP OpenView Business Process Insight

For the Windows® Operating System

Software Version: 02.10

Integration Training Guide - Business Events

hp ®

i n v e n t

Legal Notices

Warranty

Hewlett-Packard makes no warranty of any kind with regard to this document, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

A copy of the specific warranty terms applicable to your Hewlett-Packard product can be obtained from your local Sales and Service Office.

Restricted Rights Legend

Use, duplication, or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause in DFARS 252.227-7013.

Hewlett-Packard Company
United States of America

Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19(c)(1,2).

Copyright Notices

Trademark Notices

Java™ is a US trademark of Sun Microsystems, Inc.

Microsoft® is a US registered trademark of Microsoft Corporation.

UNIX® is a registered trademark of The Open Group.

Windows® and MS Windows® are US registered trademarks of Microsoft Corporation.

Documentation Updates

This manual's title page contains the following identifying information:

- Software version number, which indicates the software version
- Document release date, which changes each time the document is updated
- Software release date, which indicates the release date of this version of the software

To check for recent updates, or to verify that you are using the most recent edition of a document, go to:

   http://ovweb.external.hp.com/lpe/doc_serv/

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HP sales representative for details.

Support

Please visit the HP OpenView support Web site at:

http://www.hp.com/managementsoftware/support

This Web site provides contact information and details about the products, services, and support that HP OpenView offers.

HP OpenView online software support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valuable support customer, you can benefit by using the support site to:

- Search for knowledge documents of interest
- Submit enhancement requests online
- Download software patches
- Submit and track progress on support cases
- Manage a support contract
- Look up HP support contacts
- Review information about available services
- Enter discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and log in. Many also require a support contract.

To find more information about access levels, go to:

http://www.hp.com/managementsoftware/access_level

To register for an HP Passport ID, go to:

http://www.managementsoftware.hp.com/passport-registration.html

Contents

1 Introduction

This chapter looks at the two different types of events that are received by the OpenView Business Process Insight (OVBPI) Business Impact Engine, and then discusses the overall requirement for the OVBPI Business Event Handler.

Overall Architecture

The OVBPI Business Impact Engine receives information using two types of events:

Figure 1   The Two Types Of Event Feeds
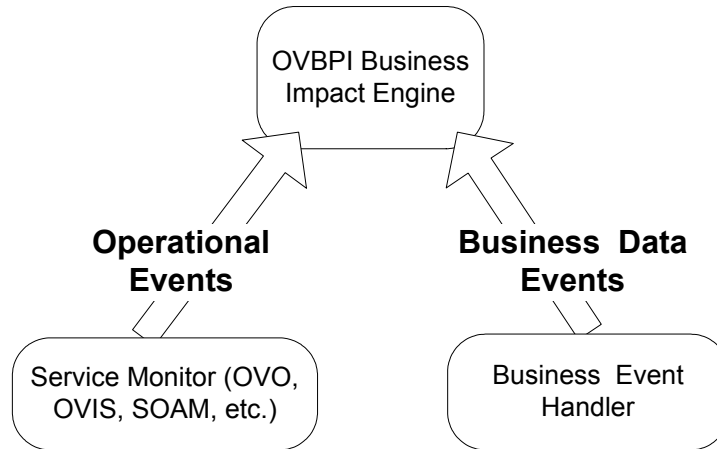
These two "event feeds" provide two fundamentally different types of event:

- Operational Events

  These come in from your service monitoring/management system - for example, HP OpenView Operations (OVO), HP OpenView Internet Services (OVIS), HP Service Oriented Architecture (SOA) Manager and/or some other package. These operational events tell the OVBPI Business Impact Engine about the status of your underlying services - whether a machine has gone down, whether a database or application has crashed, etc..

- Business Data Events

  These events tell the OVBPI Business Impact Engine what is happening in your underlying application systems. It depends how the Business Event Handler is configured, but typical examples might be an event to indicate when a new order has been placed, by whom, and for how much. An event to indicate when a customer has changed his or her payment details, etc..

This manual focuses on **Business Data Events**.

Implementation

By expanding Figure 1 and focusing on the Business Event Handler, the picture can be redrawn as follows:

Figure 2   The Business Event Handler:
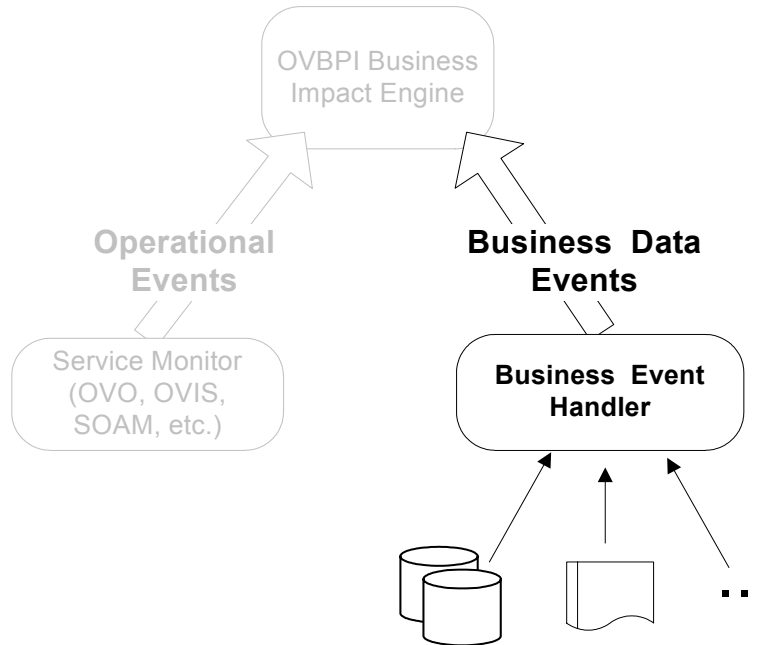
The Business Event Handler's job is to monitor the available data sources (files, databases, applications, etc.) and as certain events occur within these sources, the Event Handler needs to be able to raise these as business events and send these events to the OVBPI Business Impact Engine.

This monitoring of the available data sources, and the ability to raise business events, does not happen automatically. This is something that needs to be configured. You need to configure what data sources you will monitor, specify what situations to look out for, and then specify exactly how you want these raised as business events.

Thus, the Event Handler needs to be:

- Easily configurable.

- Able to access a wide range of data sources (flat-files, databases, applications, sockets, etc.)

- Able to transform the source data into a consistent format that is appropriate for the OVBPI Business Impact Engine to receive and process.

Needing to access data in a range of different formats, and synthesize a common format, is very much the territory of Enterprise Application Integration (EAI) solutions. However, for the needs of OVBPI, a full EAI solution is simply not required. It would be overkill.

After a detailed investigation of available software technologies, the OVBPI Business Event Handler has been implemented using the **openadaptor** open source technology.

openadaptor comes with a wide range of existing adaptors, allowing access to most of the "standard" data sources with very little effort. And being an open source product, you have full access to the source such that it is very easy to extend the list of adaptors should you ever require it.

To understand how to configure the Event Handler it is essential that you gain an understanding of the openadaptor product. Once you understand openadaptor and all its capabilities (and there are many), you are then able to understand the various enhancements that the OVBPI project has made to openadaptor to enable a higher level of usability and maintainability. You will then be able to consider your options for how you might use this technology to architect and implement your Business Event Handler.

Let's now turn to Chapter 2, openadaptor, and gain an understanding for openadaptor and all it has to offer.

2 openadaptor

This chapter provides a basic introduction to openadaptor.

This chapter looks at the origins of openadaptor, discusses some basic concepts, introduces some new terminology, and then progresses through various examples of how to configure and run openadaptor.

At the end of this chapter there is a section listing some useful openadaptor configurations and options.

If you are already comfortable with running and configuring openadaptor then you can skip this chapter.

What is openadaptor?

openadaptor is an open source adaptor integration toolkit, written in Java.

The code was originally developed in-house by the investment banking division of Dresdner Klienwort Wasserstein (DrKW) under the name "Dealbus".

openadaptor 1.0 was released to the open source community in January 2001 under a public license.

openadaptor comes complete with a wide range of pre-written adaptors, including:

- JDBC
- Flat Files
- Sockets
- JMS
- ...and many more

As well as the standard adaptors, you have full access to the actual Java code and thus you can extend any of these adaptors, or develop your own.

Web Site

openadaptor has its own Web site which you can visit at **www.openadaptor.org**. This site provides full documentation, tutorials, mailing lists, a technical forum, access to source code, and more. It is well worth a visit.

What Version?

The OVBPI Event Handler is based upon openadaptor release version **1_6_5**.

Documentation

Although the openadaptor Web site has the product documentation, they tend to only make the most recent release documentation available. So you always need to be careful which version of documentation you are looking at.

To make things easy for you, OVBPI ships a complete set of documentation for the version of openadaptor that it uses. That is, all the openadaptor documentation that you need is shipped with OVBPI.

openadaptor is installed under the directory:

```
OVBPI-install-dir\nonOV\openadaptor\1_6_5
```

Under this directory you find the following sources of documentation:

- The openadaptor **programmers guide** is found in the subdirectory:

  ```
  docs\pg
  ```

  You can browse the index.htm file.

- The openadaptor **javadocs** are located in the subdirectory:

  ```
  javadocs
  ```

  You can browse the index.htm file.

- The openadaptor **example configuration files** are located in the subdirectory:

  ```
  cookbook
  ```

- The actual openadaptor **source code** is located under the subdirectory:

  ```
  src
  ```

Basic Concepts

Suppose you are asked to write a custom piece of software to read records from a simple flat file and write these records into a database. There was only one condition: if the data record in the flat file starts with a # character, then you are to ignore this line as it is a comment.

Your custom application would probably end up containing the following main code segments:

- Code to handle the input file

  This would need to know how to open the flat file, and then how to read each record. It would then need to know the format of each record of data. For example, maybe it would read the data record as a comma separated list of values, or maybe it was a fixed length record structure.

- Code to apply the filtering

  This is probably just a simple test looking for the value of the first character in each record read from the flat file.

- Code to handle the output to the database

  This would need to know how to open the database (connection strings, user name, password etc.) and then how to write the data to the correct data table.

- A "main" module

  You would need code that actually invokes these various modules, handling any error conditions etc.

You could visually represent this code as follows:

Figure 3   A Custom file-to-database Adaptor
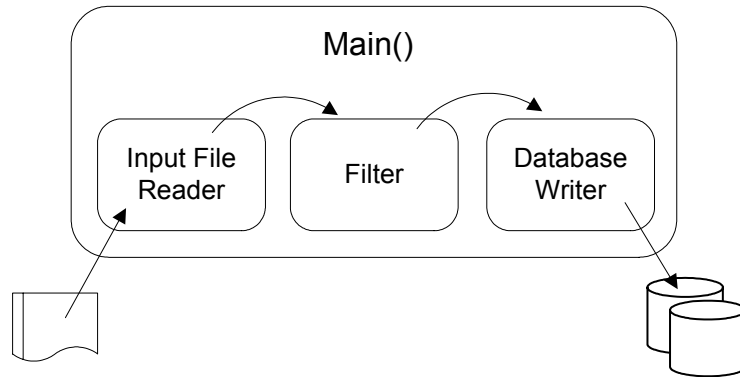
Where the `Input File Reader` reads the next line of the file. This line is then passed to the `Filter`. If it is not a comment line, then it is passed to the `Database Writer` to be written to the database.

You now have a custom "file to database adaptor" which adapts from that particular flat file format to your particular database.

Suppose a few months later, you are asked to write another piece of custom software. This time, you are to read from a flat file and then output each data record to a specific TCP/IP socket. You are again asked to ignore comment lines that start with a # character.

Straight away you realize that you can probably reuse quite a large part of your existing "file to database" adaptor code. You might need to alter the flat file reader to handle a different file format, and you would need to replace the database writer with the appropriate socket writer code.

Your new "file to socket adaptor" would look something like this:

Figure 4   A Custom file-to-socket Adaptor
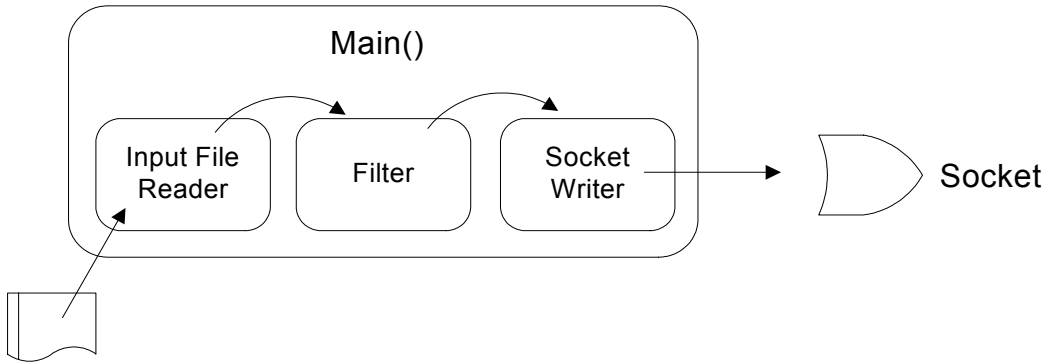
You can probably see a basic pattern starting to emerge.

The developers at openadaptor recognized this, and other, patterns and have put together a configurable system of readers, filters and writers - which all run under a configurable "main".

However, they do not use these terms of readers, filters, writers, etc.

openadaptor uses the following terms:

- **Source**

  This is the name given to the input data reader.

  There would normally be only one source component, however, openadaptor allows you to have more than one source component. If you do have more than one source component then it just means that each record is read from a source and processed, then read from the next source and processed, and so on through all the sources. Indeed, openadaptor make no promises as to the order the sources are processed. But, typically you see one source component within an adaptor.

- **Pipe**

  This is the name given to a filter. In fact, a pipe does not always have to be a filter. You might use a pipe to add extra attributes to each data record, or to rename (or alias) attribute names. In other words, a pipe is what you use if you want to provide any type of data transformation or enrichment.

  You can have as many pipes as you require for your adaptor...or none.

- **Sink**

  This is the name given to the output data writer. An adaptor may have one or more sinks. If you have more than one sink this just means that the resultant output record is written (duplicated) to all sinks.

- **Controller**

  For each adaptor, there is a controller module that handles the flow of data between the various source, pipe and sink components.

So you could now redraw the previous "file to socket adaptor" using openadaptor terminology. It would look something like this:

Figure 5   Using openadaptor Terminology.
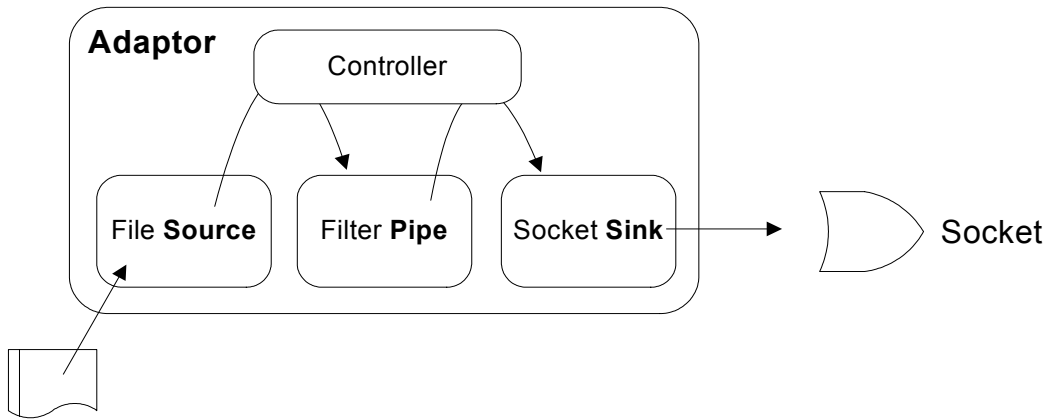
So, using openadaptor terminology:

- You have an adaptor which consists of the components:
    — A File Source
    — A Filter Pipe
    — A Socket Sink

You tend not to mention the controller component because every openadaptor adaptor has a controller component. It is there by definition.

Building Blocks

openadaptor comes complete with a large array of pre-built components. They provide sources for JDBC, flat files, sockets, etc.. There are pipes for adding timestamps, aliasing attribute names, etc.. And there are sinks for virtually all the corresponding source formats (JDBC, flat files, sockets, etc.).

Most components tend to limit their processing ability to the minimum. That is, a source is focused on reading data records. They typically do not build in much (or any) filtering, or data enrichment, into a particular source component. They prefer to put all that functionality into a separate pipe. That way, the pipe can be easily reused to provide its data enrichment or filtering for use with a different source. Hence, you have a series of basic "building blocks" - sources, pipes and sinks - that you can then use to build whatever adaptor is required for your needs.

This approach also makes it very easy to add new components. If you were to write a new pipe (for example) that could carry out database lookups and issue data retrieval based on the contents of the current incoming data record, you would be able to use this pipe with any existing, or new, source or sink. It makes for a very powerful suite of software.

The other important aspect of an openadaptor component is that it is configurable. All components, whether source, pipe or sink, are configured with entries in a properties file. For example, the FileSource is a source that is able to read data records from a file. The source is written such that you configure whether each data record attribute is delimited or fixed in length, how many attributes there are in each record, attribute names, etc.. This allows for a fairly generic source (or pipe or sink) to be written to cope with a wide range of capabilities.

Using openadaptor

Let us take a look at how to actually configure an adaptor and then get it running.

Configuring an Adaptor

All openadaptor components are configurable by placing entries in a properties file. Typically people set up one properties file for each adaptor. That is, one properties file contains the configuration details for that adaptor's source(s), pipe(s) and sink(s). However, you could quite happily set up one properties file containing the configuration details for many adaptors.

Let's go through and see how to configure an adaptor to read a simple text file and output each data record to the screen (stdout). That is, let's configure the following adaptor:

Figure 6   A Simple file-to-screen Adaptor.
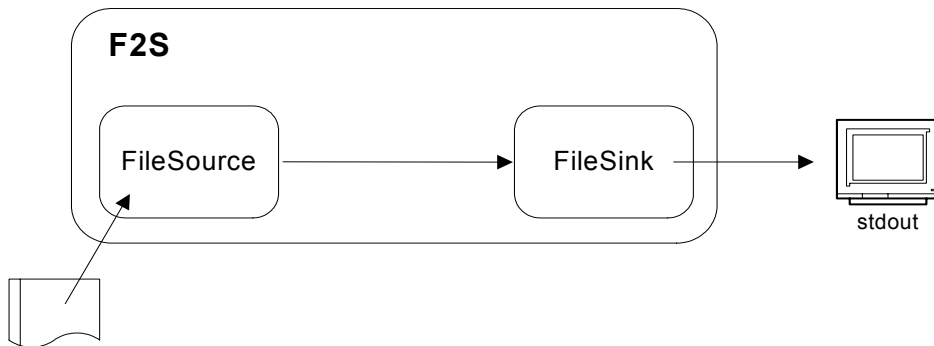
where:

- The adaptor is called: F2S - which stands for "File to Screen"

- You use a File Source to read the input file

- You use a File Sink to write to the screen

  It just so happens that if you configure the File Sink and do not give it an output file name, it defaults to your stdout.

You do not show the controller in this diagram as it is assumed to be part of any adaptor.

For this example, the input file is as follows:

```
Bovis|Geoff|54 Home St|1344-999456
Cranks|Monty|39 New Street|1276-888345
Saddit|Fred|1 Old Street|3-59-7427060
Thompson|Rita|37 London Rd|207-222567
Vexum|Harry|15 Home Place|208-333456
```

where:

- There are four attributes: Surname, Firstname, Address, Phone
- Each attribute is separated by a pipe (|) character

The configuration file (properties file) to configure this F2S adaptor is as follows:

```
# List and name all components for this adaptor
# -------------------------------------------
F2S.Controller.Name  = Controller
F2S.Logging.Name     = Logging
F2S.Component1.Name  = C1
F2S.Component2.Name  = C2

# Show how the components link together
# ----------------------------------
F2S.C1.LinkTo1       = C2

# Configure the controller
# -----------------------
F2S.Controller.ClassName      = org.openadaptor.adaptor.SimpleController

# Tracing/Logging Settings
# -----------------------
F2S.Logging.LogSetting1       = INFO
F2S.Logging.LogLinesToCache   = 100
F2S.Logging.LoggingPackageInfo = false
F2S.Logging.LoggingThreadInfo  = false
F2S.Logging.LoggingTimeInfo    = true

# C1 is the FileSource
# --------------------
F2S.C1.ClassName      = org.openadaptor.adaptor.standard.FileSource

F2S.C1.InputFileName  = simple.txt

F2S.C1.DOStringReader = org.openadaptor.dostrings.DelimitedStringReader

F2S.C1.Type           = MyType
F2S.C1.NumAttributes  = 4
F2S.C1.FieldDelimiter = 124
F2S.C1.AttName1       = Surname
F2S.C1.AttName2       = Firstname
F2S.C1.AttName3       = Address
F2S.C1.AttName4       = Phone

# C2 is a FileSink
# ----------------
F2S.C2.ClassName      = org.openadaptor.adaptor.standard.FileSink
```

where each line starts with the name of this adaptor, F2S.

You can now understand how you can have more than one adaptor within a single configuration file.

Let us now discuss this configuration file section by section.

The Components of the Adaptor

The first section is where you list all the components that make up this adaptor, assign them names, and then specify the linkage.

```
# List and name all components for this adaptor
# -------------------------------------------
F2S.Controller.Name  = Controller
F2S.Logging.Name     = Logging
F2S.Component1.Name  = C1
F2S.Component2.Name  = C2

# Show how the components link together
# -------------------------------------
F2S.C1.LinkTo1       = C2
```

Here you have named the components C1 and C2. You could have named them FSource and FSink if you had wanted to. You are just assigning names that you then use throughout the rest of the configuration file.

You use the LinkTo<n> property to show how the components link from one to the next. In this case you are simply linking C1 (your FileSource) to C2 (your FileSink) - as in

The two lines:

```
F2S.Controller.Name  = Controller
F2S.Logging.Name     = Logging
```

are also defining components of the adaptor. Each adaptor has one controller. You can call it any name you like, however most configurations tend to call it Controller.

The Logging line is declaring a logging component and its name.

The Controller

```
# Configure the controller
# -----------------------
F2S.Controller.ClassName = org.openadaptor.adaptor.SimpleController
```

This is standard on all adaptors. There is a controller provided by openadaptor (`SimpleController`) and you specify its class name here.

Indeed, if you do not specify this line, the controller defaults to this anyway.

The Logging Component

```
# Tracing/Logging Settings
# -----------------------
F2S.Logging.LogSetting1         = INFO
F2S.Logging.LogLinesToCache     = 100
F2S.Logging.LoggingPackageInfo  = false
F2S.Logging.LoggingThreadInfo   = false
F2S.Logging.LoggingTimeInfo     = true
```

The main setting here is the `LogSetting1` property. This tells the adaptor to output "information" level logging messages to standard error. `INFO` is the default setting. A more verbose setting that is good for troubleshooting is `DEBUG`. Obviously `DEBUG` mode can slow the adaptor's performance greatly.

The other log settings just specify what addition information you might want to see in the log messages.

Configure the FileSource

Let's consider this in two parts:

```
# C1 is the FileSource
# ---------------========
F2S.C1.ClassName      = org.openadaptor.adaptor.standard.FileSource

F2S.C1.InputFileName  = simple.txt

F2S.C1.DOStringReader = org.openadaptor.dostrings.DelimitedStringReader
```

The `ClassName` property is where you specify the Java class for this component. This is where you specify the Java class that you want the adaptor controller to load in at run time. When this class is loaded, it initializes itself, in this case, as component `F2S.C1` and then configures itself according to any further configuration properties specified within this configuration file.

Obviously, for this to work, you need to set up a correct run time Java classpath when you come to run the actual adaptor.

As you might imagine, when this `FileSource` class runs, it looks for the `InputFileName` property - prefixed by the same prefix as itself (that is, `F2S.C1` in this example.) This then tells the `FileSource` class the full path name of its input file. In this example, the input file is `simple.txt` in the local directory.

The `FileSource` component then looks to see how it is to read the file. It does this using the `DOStringReader` property. You see, the `FileSource` component is written in such a way that you can specify whichever Java class you like (so long as it implements a `DOStringReader`) to read your input file. Two very popular classes that are provided with openadaptor are `DelimitedStringReader` and `FixedWidthStringReader`. This example specifies the `DelimitedStringReader` class.

The `DelimitedStringReader` class then loads the next set of properties:

```
F2S.C1.Type           = MyType
F2S.C1.NumAttributes  = 4
F2S.C1.FieldDelimiter = 124
F2S.C1.AttName1       = Surname
F2S.C1.AttName2       = Firstname
F2S.C1.AttName3       = Address
F2S.C1.AttName4       = Phone
```

where:

- `Type`

  Sets the overall "record type". This means that the data record is marked as being of this type. It is just a way of "typing" (or naming) record and can be useful for filtering within pipes. There are examples of this later on.

- `NumAttributes`

  This simply tells the string reader how many attributes to expect in each record.

- FieldDelimiter

  This specifies the character that acts as the field delimiter. You must specify the actual decimal value for this delimiter character, and Decimal 124 is the pipe (|) character. If no FieldDelimiter is specified then it defaults to a comma(,).

- AttName<n>

  This simply allows you to assign attribute names to each field in the data record.

Configure the FileSink

```
# C2 is a FileSink
# ------------------
F2S.C2.ClassName = org.openadaptor.adaptor.standard.FileSink
```

All you need to do is specify the class name for the FileSink. By default, a FileSink outputs all data records, as XML, to your stdout (the screen).

And that is it. You now have a configuration file which holds the necessary property settings to configure your F2S adaptor.

Running An Adaptor

To run your new adaptor you need to do two things:

- Set up the correct Java classpath
- Run the `org.openadaptor.adaptor.RunAdaptor` class giving it the name of your property file and the name of the adaptor to run

Setting Up The Correct Java Classpath

openadaptor provide templates for this in the *OA-install-dir*\examples directory.

The templates all start with the name: `set_classpath_jdk_`

They then have the version of the Java JDK to which they correspond, followed by one of the following extensions:

- `.bat` files for use on a Windows-PC installation
- `.profile` files for use on a UX installation
- `.cygwin` files for use if you are running the share-ware product Cygwin

    openadaptor encourages the use of Cygwin for PC installations as it provides many of the features of UX, however, it is not a requirement.

You choose the appropriate template and then ensure that the `CLASSES_DIR` variable is set up correctly.

The `CLASSES_DIR` variable needs to be set to the fully qualified path of your openadaptor `classes` directory. Depending on the installation you may find that this is already correctly set up, or you may find that the `CLASSES_DIR` variable is set to a `..\` relative directory.

So...check the line:

```
set CLASSES_DIR= ...
```
and make sure that the directory specified is fully qualified.

For example:

The line might end up looking as follows:

```
set CLASSES_DIR=c:\bin\openadaptor\1_6_5\classes
```
where your openadaptor installation is under the directory
c:\bin\openadaptor\1_6_5

Running org.openadaptor.adaptor.RunAdaptor

`org.openadaptor.adaptor.RunAdaptor` is the Java class that runs all openadaptor adaptors.

You simply provide the name of your property file and the name of the adaptor to run.

The typical command line is:

```
$ java org.openadaptor.adaptor.RunAdaptor FilePrint.props F2S
```

So, to run the example adaptor you would do the following:

For PC:

```
$ cd OA-install-dir\examples
```

```
Edit the file: set_classpath_jdk_1_4.bat
and set CLASSES_DIR= to your fully qualified path for the
classes subdirectory within your openadaptor installation.
```

```
$ set_classpath_jdk_1_4.bat
```

```
$ cd the directory containing your FilePrint.props file
```

```
$ java org.openadaptor.adaptor.RunAdaptor FilePrint.props F2S
```

For UX:

```
$ cd OA-install-dir/examples
```

```
Edit the file: set_classpath_jdk_1_4.profile
and set CLASSES_DIR= to your fully qualified path for the
classes subdirectory within your openadaptor installation.
```

```
$ . ./set_classpath_jdk_1_4.profile
```

```
$ cd the directory containing your FilePrint.props file
```

```
$ java org.openadaptor.adaptor.RunAdaptor FilePrint.props F2S
```

The output then appears on your screen. You see the openadaptor copyright information, followed by various INFO log messages, followed by a series of five XML records.

You may encounter a problem when you run the `set_classpath_jdk...` script that sets the openadaptor classpath. If you do have a problem with this script, refer to Command Line Too Long on page 34.

Command Line Too Long

When you edit the appropriate `set_classpath_jdk...` template script file, you set the `CLASSES_DIR` variable to be the fully qualified directory path of your openadaptor `classes` directory. When you execute this `set_classpath_jdk...` file it uses this `CLASSES_DIR` to set up the `CLASSPATH` environment variable.

When setting up the `CLASSPATH` variable, the `set_classpath_jdk...` script appends a number of Jar files, all with fully qualified path names that are based on the value of `CLASSES_DIR`. This can result in an attempt to set the `CLASSPATH` variable to a value that is too large for your machine's operating system. For example, when you run the `set_classpath_jdk...` script you might get the error message:

```
The input line is too long.
```

If you do encounter this error when running the `set_classpath_jdk...` script, you need to shorten the directory path within the `CLASSES_DIR` variable. There are two main options available:

1. Move the openadaptor installation directory.
2. Map the openadaptor install directory to a shorter path.

Move the openadaptor Installation Directory

You can reinstall openadaptor to a directory with a shorter directory path name. Alternatively, you can just copy the openadaptor directory structure to a directory that has a shorter path name.

Map the openadaptor Install Directory to a Shorter Path

Rather than reinstall or move openadaptor, you can map a directory to your openadaptor install directory.

For example, on a UX installation, you can set up a symbolic link as follows:

```
$ ln -s OA-install-dir oadir
```

where `OA-install_dir` is the full path name of your openadaptor installation.

You can then set `CLASSES_DIR` to `oadir/classes` rather than the full path name.

For a PC installation, you can map a drive letter as follows:

```
subst w: /D
subst w: OA-install-dir
```

where `OA-install_dir` is the full path name of your openadaptor installation.

You can now set `CLASSES_DIR` to `w:classes` rather than the full path name.

The Output From RunAdaptor

When you run the F2S adaptor you see a combination of logging output and actual data output.

The first block of output is common to all adaptors when they execute. openadaptor outputs a copyright notice followed by some basic information such as the software version number, build information and the classpath value.

You then see a series of INFO and (probably) WARN output lines. This again is common for all adaptors when they execute. The amount of information printed here is dependant upon the logging settings in the adaptor's configuration file. Remember that for this example there were the following lines in the configuration file:

```
F2S.Logging.LogSetting1        = INFO
F2S.Logging.LogLinesToCache    = 100
F2S.Logging.LoggingPackageInfo = false
F2S.Logging.LoggingThreadInfo  = false
F2S.Logging.LoggingTimeInfo    = true
```

This means that the adaptor outputs any logging messages that are at level INFO or more serious (such as WARN, ERROR, FATAL).

As a general rule, adaptors are written to output lots of INFO output at initialization time. The idea is that they show an INFO line for each setting that is read from the configuration file, and show a WARN line if they are defaulting any values. This rule is not strictly adhered to, but it explains why you should expect to see an appropriate amount of INFO and (optionally) WARN output when an adaptor is starting up. This is all very useful for troubleshooting - should you ever have problems.

What then follows is the actual data output records. These appear as XML output.

The reason they appear as XML is because that is the default output format for the `FileSink`.

As you are running `FileSink` in default mode, it outputs the XML records to the screen, alongside and intermixed with any logging output (such as INFO, WARN, etc. messages).

In actual fact, the XML output is sent to **stdout** and the logging output is sent to **stderr**. So you can easily separate them if required. Mind you, most adaptors write their stdout to a file, socket or database, and thus it is typically not a problem when running real-world adaptors.

Lab - Using openadaptor

The purpose of this lab is to give you practical experience using openadaptor.

You start by using a pre-built configuration file and see how to run the adaptor. You then make some modifications to this configuration file and see how these changes affect the adaptor output.

This lab assumes that:

- openadaptor is already installed on your machine
- Java is installed and `$JAVA_HOME\bin` is correctly set in your `PATH` environment variable

Set Up the Classpath

- Change directory to: `OA-install-dir\examples`

  You should have openadaptor installed under the `OVBPI-install-dir\nonOV\openadaptor\1_6_5` directory.

- Edit the file: `set_classpath_jdk_1_4[.bat][.profile]`

  If you are running on a PC, then edit the `.bat` file. If running on a UX machine then edit the file ending in `.profile`.

  — Change the line that sets the `CLASSES_DIR=` variable, and set `CLASSES_DIR` equal to the full path of your `classes` subdirectory of your openadaptor installation

  — Save this file

- Start a new command window

  This is either a terminal window on a UX machine, or a command window on a PC.

- In this command window, source this `set_classpath_jdk_1_4` file

  For UX:

  You need to source the file using the "." command. For example:

  ```
  $ . ./set_classpath_jdk_1_4.profile
  ```

For PC:

You simply need to run the .bat file.

⚠ If you encounter an error where the set_classpath_jdk_1_4 script displays that the command is too long then refer to Command Line Too Long on page 34.

You now have a command window running with the CLASSPATH variable set correctly.

Running the Adaptor

- In this command window, change directory to the events-tg\labs directory

  In there you will find two files:

  — simple.txt

  The input data for this adaptor. Containing the data:

  ```
  Bovis|Geoff|54 Home St|1344-999456
  Cranks|Monty|39 New Street|1276-888345
  Saddit|Fred|1 Old Street|3-59-7427060
  Thompson|Rita|37 London Rd|207-222567
  Vexum|Harry|15 Home Place|208-333456
  ```

  — FilePrint.props

  The adaptor configuration file.

- Have a look at the FilePrint.props configuration file and make sure that you are happy with its basic layout

  If you have any questions you can refer back to Configuring an Adaptor on page 25.

Now you need to run this F2S adaptor:

- Within the correct command window (the one with classpath set correctly) run the adaptor as follows:

```
$ java org.openadaptor.adaptor.RunAdaptor FilePrint.props F2S
```

You should see the output to your screen, showing each data record as an XML output record.

Notice:

- The XML structure shows the correct names for each data item - `Surname`, `FirstName`, `Address`, `Phone` - as specified in your configuration file
- Each XML record is of type `MyType` - as specified by your configuration

Troubleshooting

Now let's alter the logging settings to see what sort of information you can output for this adaptor:

- Edit the `FilePrint.props` file
- Find the logging section and change it to be as follows:

```
F2S.Logging.LogSetting1          = DEBUG
F2S.Logging.LogLinesToCache      = 100
F2S.Logging.LoggingPackageInfo   = false
F2S.Logging.LoggingThreadInfo    = false
F2S.Logging.LoggingTimeInfo      = true
```

- Now run the adaptor again...

```
$ java org.openadaptor.adaptor.RunAdaptor FilePrint.props F2S
```

  — You should see a lot more output this time. The extra lines are marked as type DEBUG

  — You should also notice that the debug output occurs during the output of the actual data records as well

  — You can probably appreciate that this debug output can get quite large for some adaptors

- If you want to see some further output options, try setting the log settings as follows:

```
F2S.Logging.LogSetting1      = DEBUG
F2S.Logging.LogLinesToCache  = 100
F2S.Logging.LoggingPackageInfo = true
F2S.Logging.LoggingThreadInfo  = true
F2S.Logging.LoggingTimeInfo    = true
```

When you re-run the adaptor you see that each log message now shows you the thread and Java package name details.

Outputting to a File

Let's now modify the configuration file such that the `FileSink` outputs to an actual file rather than the screen.

- Edit the adaptor configuration file and add the following line:

```
F2S.C2.CreateOutputFile = true
F2S.C2.OutputFileName   = simple_output.txt
```

This tells the `FileSink` (component C2) to create an output file and place all output data into that file.

- Re-run your adaptor

```
$ java org.openadaptor.adaptor.RunAdaptor FilePrint.props F2S
```

Now you see the log messages output to your screen, but no actual XML data records. The XML data records have all been written to the file `simple_output.txt`.

Outputting Non-XML Records

The `FileSink` defaults to outputting its data records as XML. You can specify a different output writer if you wish...

- Edit your configuration file and add the following line:

```
F2S.C2.DOStringWriter = org.openadaptor.dostrings.DelimitedStringWriter
```

This tells the `FileSink` (component C2) to output its data records using the specified output writer. `DelimitedStringWriter` is a standard writer supplied by openadaptor and, if no delimiter is specified it defaults to using a comma(,).

- Re-run your adaptor and you should see that the output data is no longer XML, but instead is as follows:

```
Bovis,Geoff,54 Home St,1344-999456
Cranks,Monty,39 New Street,1276-888345
Saddit,Fred,1 Old Street,3-59-7427060
Thompson,Rita,37 London Rd,207-222567
Vexum,Harry,15 Home Place,208-333456
```

where the data records have been written out using the comma delimiter.

Well done! You have reached the end of the lab.

Adding a Pipe

As discussed earlier, within openadaptor, sources and sinks provide basic read and write capabilities (respectively) and little or no filtering capabilities.

If you require some form of filtering, or some other more general form of data transformation, then you add a **pipe** to your adaptor.

Let's consider the example where you are reading records from a simple flat file and outputting them to the screen. Suppose you now wish to add filtering such that the following two conditions are applied to each data record:

- Filter out any surnames that start with a character in the range A to M

  ...and then ...

- Filter out records where the telephone number starts with a "2"

Pictorially, you want to configure the following:

Figure 7   F2S Adaptor With A Filter Pipe
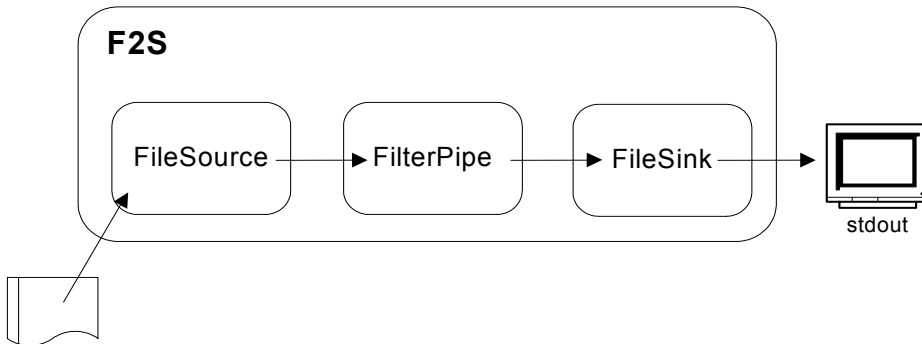
where the pipe applies the required filtering to either pass or block data records.

Choosing a Pipe

You must first see if there is a filter pipe available from openadaptor that provides the kind of filtering you require.

If one does not exist then you need to have someone write one for you.

By consulting the openadaptor documentation/Web site, you can see that there is a pipe available that provides filtering of this nature, and it is called:

```
org.openadaptor.adaptor.standard.FilterPipe
```

Configuring the Adaptor

When adding a new component to an adaptor configuration file there are two basic steps required - and this is true whether you are adding a pipe, source or sink:

- First you must edit the section (typically at the start of the configuration file) that describes the components that make up this adaptor, and then specifies the linkages between the various components

- Then you must add in the actual configuration for the new component

The configuration now contains the following section to label the components and describe the linkages:

```
# List and name all components for this adaptor
# --------------------------------------------
F2S.Controller.Name    = Controller
F2S.Logging.Name       = Logging
F2S.Component1.Name     = C1
F2S.Component2.Name     = C2
F2S.Component3.Name     = Filter

# Show how the components link together
# -------------------------------------
F2S.C1.LinkTo1         = Filter
F2S.Filter.LinkTo1     = C2
```

where:

- You have added the new component called `Filter`

  The name is whatever you wish to call it. Here it is called `Filter`. It's just a simple descriptive name. Some people might have called it `C3`. It is your choice.

- You have specified that C1 connects to the Filter component, and then Filter connects to the C2 component

This tells `RunAdaptor` that this adaptor is made up of three components, where C1 connects to Filter connects to C2 as shown in

You then add the section of configuration specific to this new `FilterPipe` component:

```
# Filter is a simple FilterPipe
# ----------------------==========

F2S.Filter.ClassName=org.openadaptor.adaptor.standard.FilterPipe

F2S.Filter.Filter1                 = block
F2S.Filter.Filter1.AttName1        = Surname
F2S.Filter.Filter1.AttValueRegExp1 = ^[A-M].*

F2S.Filter.Filter2                 = block
F2S.Filter.Filter2.AttName1        = Phone
F2S.Filter.Filter2.AttValueRegExp1 = ^2.*
```

where this specifies:

- The class to use to provide the filtering

- The `FilterPipe` allows you to specify "pass" or "block" conditions

- Filter1 specifies that you want to block any records where the `Surname` field starts with any character between caps-A and caps-M - followed by any length of text

- Any records that pass through Filter1 are then filtered through Filter2

- Filter2 specifies that you want to block records where their `Phone` attribute starts with the number `2`

As for more detailed examples of the `FilterPipe`, refer to the openadaptor documentation and the examples that come with the openadaptor installation.

Now if you run this adaptor against the input file:

```
Bovis|Geoff|54 Home St|1344-999456
Cranks|Monty|39 New Street|1276-888345
Saddit|Fred|1 Old Street|3-59-7427060
Thompson|Rita|37 London Rd|207-222567
Vexum|Harry|15 Home Place|208-333456
```

The only data record it outputs belongs to `Fred Saddit` - because all the other records have been filtered out.

Adding Multiple Pipes

If the requirement is for an adaptor to filter the data records, and then to perform some further transformation (or indeed, more filtering) then you simply add another pipe to the adaptor. You add as many pipes as you require to perform your transformations/filtering.

It is always worth checking what (if any) filtering is offered by the source you are using. Sometimes the source provides some basic filtering which might be all that you need.

Lab - Using Pipes

The purpose of this lab is to give you experience with adding pipes to your adaptor configuration and getting them working.

The FilterPipe

In the `labs` directory, find the file `FileFilterPrint.props`

This configuration file is for a basic FileSource->FileSink adaptor. It uses the input file (`simple.txt`) containing the data:

```
Bovis|Geoff|54 Home St|1344-999456
Cranks|Monty|39 New Street|1276-888345
Saddit|Fred|1 Old Street|3-59-7427060
Thompson|Rita|37 London Rd|207-222567
Vexum|Harry|15 Home Place|208-333456.
```

- Go ahead and add-in a `FilterPipe` to this adaptor where the record filtering required is as follows:

    — Filter out records where the first name starts with a letter in the range of caps-M to caps-Z

    This should output the records for `Geoff Bovis`, `Fred Saddit` and `Harry Vexum`.

    You need to refer to the `FilterPipe` javadocs from your openadaptor installation - by looking up the `org.openadaptor.adaptor.standard.FilterPipe` class.

- Now alter the filter to be as follows:

    — Filter out records where the first name starts with a letter in the range of caps-M to caps-Z

    ...and then...

    — of the remaining records, pass only those that do not have the number 2 at the start of there phone number

    This should output the records for `Geoff Bovis` and `Fred Saddit`.

The final filtering should look something like this:

```
F2S.Filter.ClassName=org.openadaptor.adaptor.standard.FilterPipe

F2S.Filter.Filter1                 = block
F2S.Filter.Filter1.AttName1        = Firstname
F2S.Filter.Filter1.AttValueRegExp1 = ^[M-Z].*


F2S.Filter.Filter2                 = pass
F2S.Filter.Filter2.AttName1        = Phone
F2S.Filter.Filter2.AttValueRegExp1 = !^2.*
```

where:

— The "!" character is used to negate the filter...but it must have no following space character. That is, `! ^2.*` does not produce the same result.

<div style="border:1px solid black; padding:1em; text-align:center;">

Well done! You have reached the end of the lab.

</div>

Using a Different Sink

openadaptor supplies a whole range of sources, pipes and sinks. You can consider these to be a set of "building blocks" with which you can build the adaptors you need for your particular integration effort.

Just as it is easy to add in a new pipe to provide the filtering required for an adaptor, it is easy to switch an adaptor to use a different sink.

Suppose you have an adaptor running as shown in :

Figure 8   File-to-Screen Adaptor - With Filtering
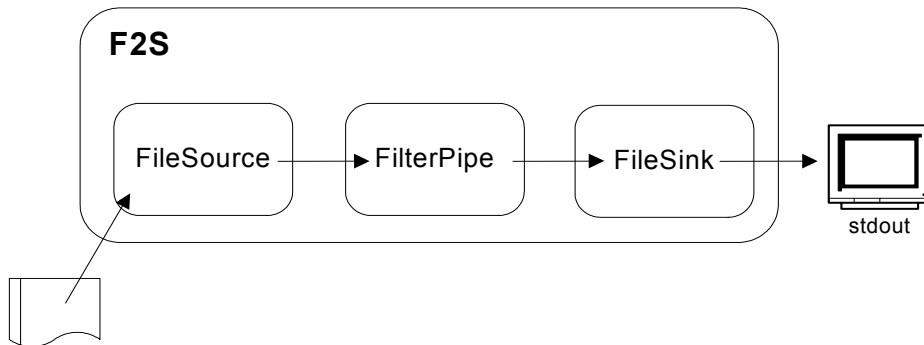
It might be that you actually need the resulting data records to be sent to a socket, however you wanted to test that you had your source and filter configuration correct before finally connecting it to the live socket.

Once you have tested out the adaptor by seeing the output to your screen, you are then able to reconfigure this adaptor such that it send its data records to a socket.

You want your new adaptor to look like this:

Figure 9   File-to-Socket Adaptor - With Filtering
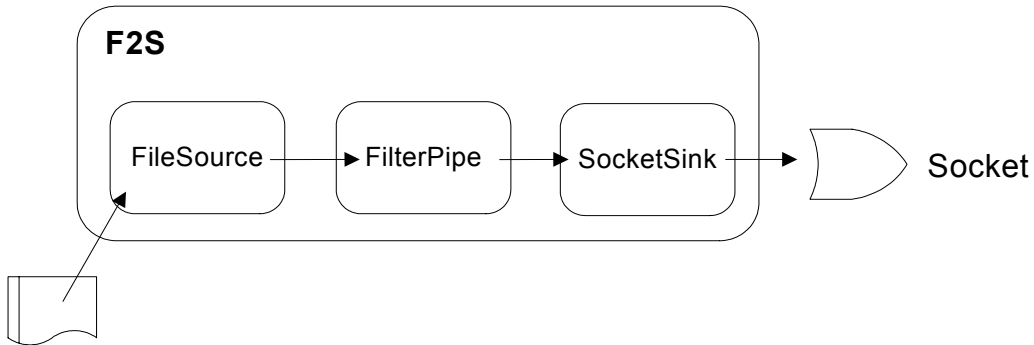
Obviously, for this to work you need a socket sink, and openadaptor provides one. It is the class: org.openadaptor.adaptor.standard.SocketSink

Now, to configure this sink, you simply do the following:

- Remove the current FileSink configuration

  There is no need to remove the actual component from the top of the configuration file.

  You simply need to remove the line that configures the actual FileSink class and any behavior. In this case the lines to remove are:

```
# C2 is a FileSink
# ----------------
F2S.C2.ClassName = org.openadaptor.adaptor.standard.FileSink
```

- You now add in the necessary configuration to declare C2 as a socket sink and tell it on which port to send its output data records

  To see how this sink is configured, refer to the openadaptor javadocs in your openadaptor installation directory, and look up the class org.openadaptor.adaptor.standard.SocketSink

The new configuration for your socket sink might be as follows:

```
# C2 is a SocketSink
# -------------------
F2S.C2.ClassName = org.openadaptor.adaptor.standard.SocketSink
F2S.C2.Port      = 29292
F2S.C2.HostName  = localhost
```

where:

— The classname specifies that you are running the `SocketSink`

— You specify the appropriate port

You would need to choose a port that was available on your server

— You specify the hostname

This example shows the use of `localhost`, but you could obviously specify whatever hostname was appropriate for your network.

And that is all it takes to switch your `F2S` adaptor from being a File-to-Screen adaptor to be a File-to-Socket adaptor.

Lab - A Socket Sink

The purpose of this lab is to give you some experience reconfiguring an existing adaptor to use a socket sink.

Configure The New Sink

In the `labs` directory, find the `FileFilterSocket.props` file

This file configures the F2S adaptor to have the following configuration:

- A `FileSource` - reading the `simple.txt` file
- A `FilterPipe` - filtering out certain records
- A `FileSink`   - displaying data to the screen

• For this lab, you are to alter the configuration so that the adaptor outputs its data to a socket sink over port `29292` on your `localhost`.

   When you think you are ready, the next section shows you how to test your adaptor...

Running The Adaptor

Now that your adaptor (hopefully :-) writes its output to port `29292` on your localhost, you need some way to test it.

If you were to run your adaptor now it would fail complaining that it could not open the socket - probably saying that the connection was refused.

This is because there is nothing at the other end of the socket to read the incoming data.

So, there is another adaptor, called SOCKPRINT, that is configured by the file `SocketPrint.props` (in the `labs` directory).

To test your adaptor:

• Open up a new command window

• Set the classpath for this new command window

• Run the SOCKPRINT adaptor

   This should then sit there saying that it is waiting for a connection.

- In another command window, with the classpath set up correctly, run your F2S adaptor

  You should see the data records appear over in the SOCKPRINT adaptor's command window.

You can hopefully see now how easy it is to switch in/out the various openadaptor components to build whatever adaptor you require.

Renaming An Adaptor

Once you have your F2S adaptor working successfully and sending its data records across port 29292, you might like to try the following:

- Try to rename your F2S adaptor to be called: `SocketSend`

Once renamed, you should be able to run the adaptor with the following command:

```
$ java org.openadaptor.adaptor.RunAdaptor FileFilterSocket.props SocketSend
```

> **Well done! You have reached the end of the lab.**

DataObject and Messages

You don't have to use openadaptor for long before you start seeing references to "DOs" and "DOStrings". Indeed, when you configured your first file-to-screen adaptor, you saw the configuration line:

```
F2S.C1.DOStringReader = org.openadaptor.dostrings.DelimitedStringReader
```

So you ask yourself: "What is a DOStringReader?"

When a data record is read in by an openadaptor source it is held as a **DataObject**. It is the internal openadaptor class used to represent the data record(s) being passed between sources/pipes/sinks.

So, a DOStringReader is simply a class that reads data records and converts them into openadaptor DataObjects - or DOs.

DataObject Type

The DataObject class is able to hold all of the data attributes for the data record, and for each attribute it holds its value along with its data type. But as well as that, you also have the idea that each DataObject can be "typed".

You saw this when you configured your file-to-screen adaptor. You had the configuration line:

```
F2S.C1.Type  = MyType
```

This tells the FileSource that each data record is to be held in a DataObject of type `MyType`. It is purely a way of assigning a "name" to each DataObject. It does not in any way affect what data can be held inside this DataObject. It is just a way of giving that DataObject a name. It might be less confusing if the configuration property was called RecordType, RecordName or even DOType, however, it is called `Type`.

Whilst the configuration property is called Type, it is sometimes reported in the log messages as the `DOType`.

As a general rule for all openadaptor sources, if you do not specify a type for the data record then the type is set to the string `Any`.

Assigning a record type can be important when using pipes.

Obviously it depends on the pipe and what that pipe does, however, by assigning a type to each data record, your pipe can be configured to behave differently depending on what record type it is handling.

This is probably best shown by an example...

Multi Record Type Example

The `FileSource` is an example of a source that is capable of setting the record type based on the contents of the data it is reading. Not all sources are capable of this, but the `FileSource` is one example.

Suppose you have the following input data:

```
TEST A neworders entry|NEWORDER|12|A comment|Another comment
TEST An update record|UPDATE|13|A comment string
TEST Another new orders entry|NEWORDER|15|Comments|More comments
TEST Another update record |UPDATE|33|comments
11|DELETE
More Neworders|NEWORDER|44|comments|more comments
```

The `FileSource` can be set up such that it can type each data record according to the value within (for example) field two (assuming a | delimiter).

The configuration would look something like this:

```
A.C1.InputFileName = multiTypes.txt

A.C1.ClassName      = org.openadaptor.adaptor.standard.FileSource
A.C1.DOStringReader = org.openadaptor.dostrings.DelimitedStringReader
A.C1.BatchSize      = 1

# The separator 124 decimal is |
A.C1.FieldDelimiter = 124

# There are three "types" available from this Source
A.C1.Type1          = NEWORDERS
A.C1.Type2          = UPDATE
A.C1.Type3          = DELETING
```

```
# The TypeFieldNumber sets the field number that determines the
# openadaptor record "type" of each record

A.C1.NEWORDERS.TypeFieldNumber   = 2
A.C1.NEWORDERS.TypeFieldMatch    = "NEWORDER"

A.C1.NEWORDERS.AttName1          = Field1
A.C1.NEWORDERS.AttName2          = EventType
A.C1.NEWORDERS.AttName3          = OrderNumber
A.C1.NEWORDERS.AttName4          = Comment
A.C1.NEWORDERS.AttName5          = More

A.C1.UPDATE.TypeFieldNumber      = 2
A.C1.UPDATE.TypeFieldMatch       = "UPDATE"

A.C1.UPDATE.AttName1             = Control
A.C1.UPDATE.AttName2             = EventTypeInThisField
A.C1.UPDATE.AttName3             = OrderNumber
A.C1.UPDATE.AttName4             = Event

A.C1.DELETING.TypeFieldNumber    = 2
A.C1.DELETING.TypeFieldMatch     = "DELETE"

A.C1.DELETING.AttName1           = OrderNumber
A.C1.DELETING.AttName2           = EventTypeField
```

where:

- You declare that this source can produce three different types of record

```
A.C1.Type1                = NEWORDERS
A.C1.Type2                = UPDATE
A.C1.Type3                = DELETING
```

- You then go on and specify how to decide the record type

```
A.C1.NEWORDERS.TypeFieldNumber   = 2
A.C1.NEWORDERS.TypeFieldMatch    = "NEWORDER"

A.C1.NEWORDERS.AttName1          = Field1
A.C1.NEWORDERS.AttName2          = EventType
A.C1.NEWORDERS.AttName3          = OrderNumber
A.C1.NEWORDERS.AttName4          = Comment
A.C1.NEWORDERS.AttName5          = More
```

This says that if the value in field number two is equal to the string NEWORDER, then this data record is of record type NEWORDERS.

You also specify the set of attribute names for this type of record.

- You then do the same for setting up the UPDATE and DELETING record types, and their definitions

It may then be that you wish to apply a filter (using the `FilterPipe`) that ignores any NEWORDERS records where the first attribute starts with the word TEST. So this is a filter that makes use of the record type information.

The filter would look as follows:

```
A.Filter.ClassName = org.openadaptor.adaptor.standard.FilterPipe

A.Filter.Filter1                  = block
A.Filter.Filter1.Type             = NEWORDERS
A.Filter.Filter1.AttName1         = Field1
A.Filter.Filter1.AttValueRegExp1 = ^TEST.*
```

where the filter is only applied to data records of type NEWORDERS. All other data records are passed through untouched.

So, being able to type data records can be of particular use when needing to apply filters.

Record Batching

An openadaptor source typically defaults to processing one record at a time. This can be configured (typically) by using the `BatchSize` property. For example:

```
A.C1.BatchSize= 30
```

This causes the source to read up to 30 records from the input source and then send them as one array of DataObjects to the receiving pipe or sink. The source might send less than 30 records if it encountered an EOF on the input source.

You can see how this might dramatically speed up your adaptor if you could process 30 (or more) records in one go, rather than handling each individual record as a single unit. And this is true, it is a feature that can greatly increase the throughput of your adaptor.

However, there is a restriction within openadaptor that a **sink** can only accept an array of DataObjects if all the DataObjects within the array are **of the same type.** That is, if your source or pipe tries to pass an array of DataObjects containing a mixture of DOTypes, then the sink will terminate with an error.

To get around the issue of sinks not being able to handle DataObject arrays that contain multiple DOTypes, there is a pipe called `ArraySplitterPipe`.

ArraySplitterPipe

The `ArraySplitterPipe` acts as a buffer in front of the sink. The idea is that your source and/or pipes can produce arrays of DataObjects that contain a mixture of DOTypes, and the `ArraySplitterPipe` is able to separate the DataObjects and pass them one at a time to the sink.

Figure 10 shows how the `ArraySplitterPipe` receives a single DataObject array, and send each DataObject as a separate message to the sink.

Figure 10 The ArraySplitterPipe
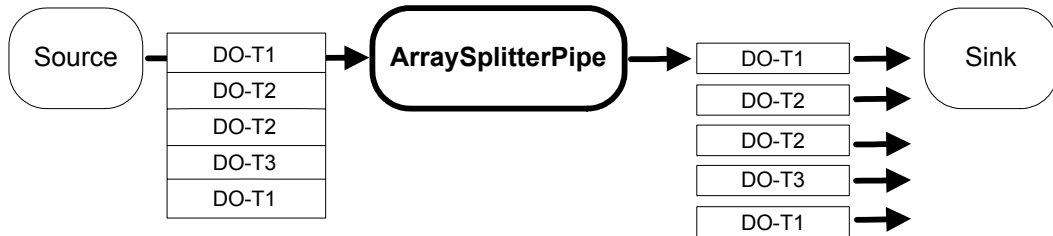
The full class name for the `ArraySplitterPipe` is `org.openadaptor.adaptor.standard.ArraySplitterPipe`.

You do not need to specify any properties for the `ArraySplitterPipe`. Just link this component before the sink in your adaptor configuration file.

You only need to use the `ArraySplitterPipe` component if your adaptor is going to send DataObject arrays with mixed DOTypes.

Messages

The unit of data that is passed between adaptor components is called a **message**. That is, data is read in by a source and it is held as an array of DataObjects. This array of DataObjects is called a message. If the source only reads a single record of data, then this DataObject array has a length of one, and the current message therefore contains only this one DataObject.

AFEditor - Configuration GUI

In all the examples so far, you have configured each adaptor by editing a configuration file. There is an alternative ... AFEditor.

AFEditor (the **A**daptor **F**ramework **Editor)** is a graphical configuration tool that lets you configure adaptors.

AFEditor is an excellent way to:

- Find out what sources, pipe, and sinks are available to you
- Find out, for a particular component, what configuration options are available

However, you should **be very careful** when using AFEditor to configure adaptors. When configuring a component, it lists the configuration properties, which is excellent. It also shows which settings are mandatory and which settings are optional. However, AFEditor lists all the options in alphabetical order, and this can often make it difficult to see which properties depend on which other properties, or to see the "logical" groupings. When you are manually editing a configuration file you are able to group the properties in the order that makes sense to you, and this can sometimes mean easier maintenance and troubleshooting.

Anyway, let us have a look at AFEditor as it can be useful...

Running AFEditor

To run AFEditor, you need to set up your classpath in the same way as you do for running an adaptor.

You then issue the command:

```
$ java org.openadaptor.adaptor.editor.AFEditor
```

You should see something that looks like this:

Figure 11 AFEditor - Main menu
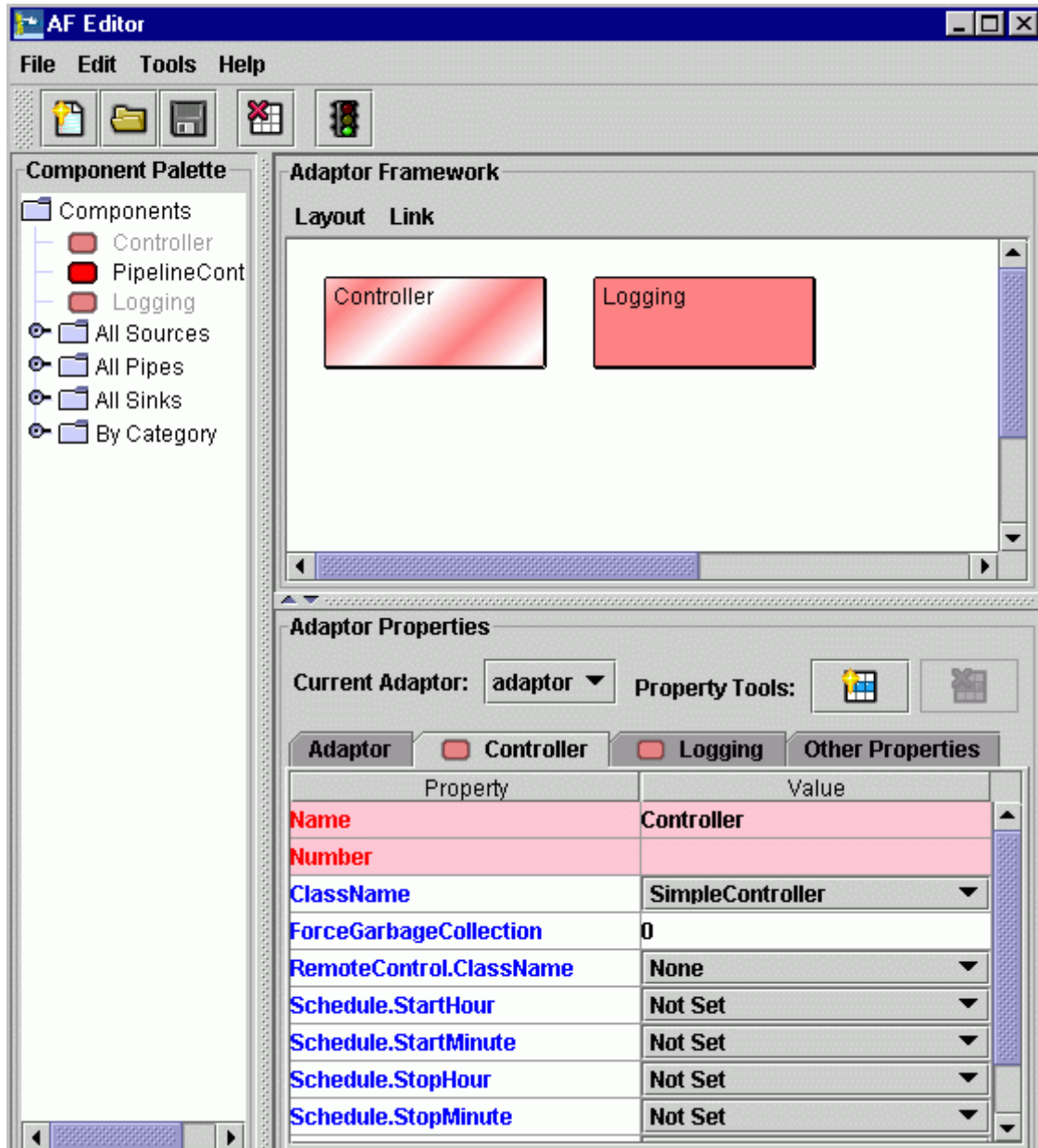
The `Component Palette` lets you see what sources, pipes and sinks are available to you.

To select a source:

- Expand the `All Sources` option within the Component Palette
- Then double-click on the desired source
- Then configure this source using the `Adaptor Properties` section of the screen

You can then repeat this procedure for a pipe (using `All Pipes`) and sink (using `All Sinks`).

Notice that you are also able to list sources, pipes and sinks `By Category`. This can sometimes make it easier to find the component(s) you need.

You can see how this is a very useful way to see what components are available to you and to see their configuration options.

AFEditor can be used to create an adaptor from scratch. That is, you run AFEditor, draw out and configure your adaptor, give it a name, and then save the configuration.

You can even run an adaptor from within AFEditor.

AFEditor can also be used to read in an existing configuration file and to then modify this further. The configuration file that you read in just needs to be a valid configuration file. It may have been created by a previous run of AFEditor or have been created by hand.

Lab - Using AFEditor

The purpose of this lab is to help you use AFEditor to create and run an adaptor.

For this lab you use AFEditor to construct a simple File-to-Screen adaptor similar to the F2S adaptor that you used in previous examples.

Your input is the file `simple.txt` that contains the following records:

```
Bovis|Geoff|54 Home St|1344-999456
Cranks|Monty|39 New Street|1276-888345
Saddit|Fred|1 Old Street|3-59-7427060
Thompson|Rita|37 London Rd|207-222567
Vexum|Harry|15 Home Place|208-333456
```

Running AFEditor

- Open a new command window

- Set the classpath as if you were running an adaptor

- Now run AFEditor with the command:

```
$ java org.openadaptor.adaptor.editor.AFEditor
```

This brings up the main menu of AFEditor...

Naming the Adaptor

In the `Adaptor Properties` pane:

• Click on the `Adaptor` tab

  Notice that some of the configuration options are in the color red (such as "Name") and some are in blue. Red properties are mandatory. Blue properties are optional. This is true for all components.

• Type in the correct name for this adaptor. Set it to: `MyAdaptor`

• Press carriage return
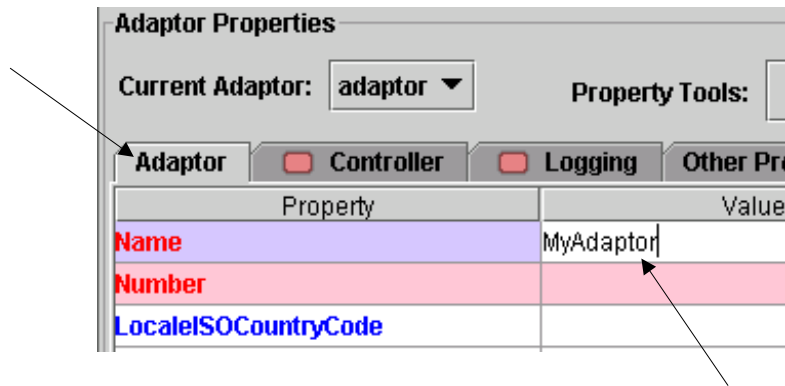
Configuring the Source

- In the `Component Palette`, open up the `All Sources` selection

  It now lists all the possible sources available to you.

  If you hold the mouse over each source, a tooltip shows up to tell you a little about each source and what they do.

- Double-click on the `FileSource`

  This now creates a FileSource icon in the `Adaptor Framework` pane.
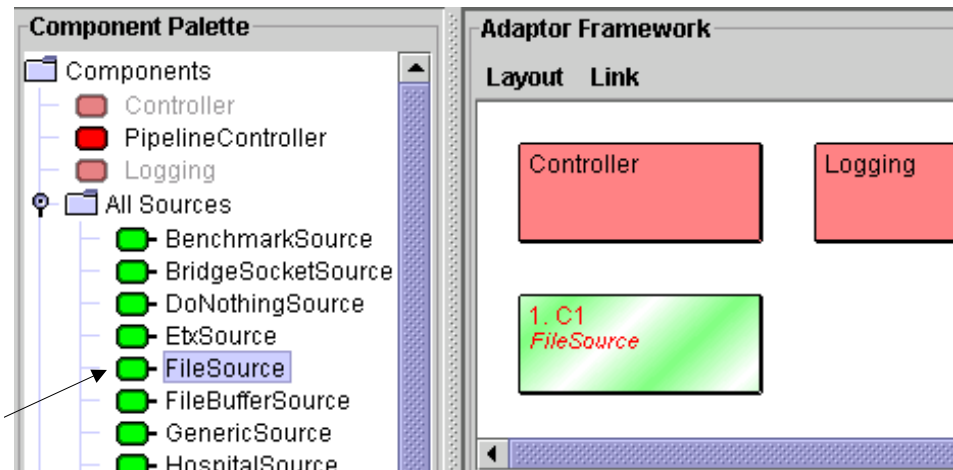
Now in the `Adaptor Properties` pane:

- Select the `C1` tab
- Set the `InputFileName` to be: `simple.txt`
- Set the `DOStringReader` to be: `DelimitedStringReader`

  Notice how more configuration options appear.

- Set `BatchSize` to be: `1`
- Set `Field Delimiter` to be: `124`

  124 is the pipe(|) character in decimal.

At this point, your Adaptor Properties pane should look something like this:

Adaptor Properties

Current Adaptor: MyAdaptor ▼   Property Tools: [🔳] [🔳]

Adaptor	🔴 Controller	🔴 Logging	Other Properties	🟢 C1

Property	Value
Name	C1
Number	1
BatchSize	1
CommentRegExp++	
DOStringReader	DelimitedStringReader ▼
DateFormat	
EmptyStringAsNull	false ▼
EndOfData	
ExitOnError	true ▼
FieldDelimiter	124
IgnoreHeaderLines	0
IgnoreQuotes	false ▼
InputFileName	simple.txt
MaxCallsPerPoll	
MessageReaderHook	
NullString	
PollPeriod	
RecordRegExp++	
StartOfData	
Type++	

In earlier examples and labs you have been given a basic adaptor configuration file. Using AFEditor lets you see all the possible configuration options. You can use the tooltips within AFEditor or the openadaptor documentation to find out about any particular options if you like.

If you remember your previous examples, when you defined the `FileSource` you also defined the number of attributes within the data record and the names for each record attribute. You also defined an overall record type (called `MyType`). So you might be wondering how you achieve this within AFEditor?

The answer is the `Type++` property.

Whenever a configuration property can have more than one value, AFEditor shows this by offering the user that option followed by the `++` character string.

The `Type` configuration option is where you specify the data record type for the DataObject as it is read from the source. You might remember that it is possible for the `FileSource` to produce many different record types based on the incoming data records. To allow for this, AFEditor offers the `Type++` option.

So, let's configure it:

- For the `Type++` option, type in: `MyType`

  You now have a `Type1` option set to `MyType`, and a series of configuration options for describing the `MyType` record structure have also appeared.

- For the `MyType.AttName++` option, type in: `Surname`

  This creates the setting: MyType.AttName1 = Surname

- Again, in the `MyType.AttName++` option, type in: `Firstname`

- Again, in the `MyType.AttName++` option, type in: `Address`

- Again, in the `MyType.AttName++` option, type in: `Phone`

That is it. You have now configured the FileSource.

Configuring the Sink

- In the `Component Palette`, open up the `All Sinks` selection

  It now lists all the possible sinks available to you.

  If you hold the mouse over each sink, a tooltip shows up to tell you a little bit about each sink and what they do.

- Double-click on the `FileSink`

  This now creates a FileSink icon in the `Adaptor Framework` pane, and it automatically connects it to the FileSource.
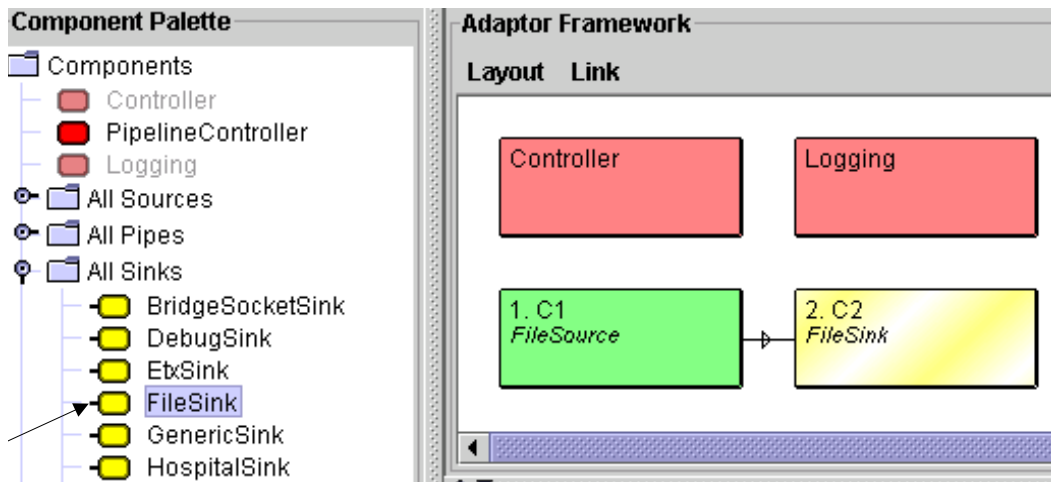
By default, the FileSink writes output records to the screen (stdout), so for this example, there is no need to configure any of the properties for the FileSink.

Save the Adaptor Configuration

Go ahead and save your configuration:

- Select: `File->Save As...`

  And save it to the file name `MyAdaptor.props` within your `labs` directory - the one that contains the simple.txt input file.

Running the Adaptor

To run this adaptor, you can either exit out to a command window with the correct classpath, and type:

```
$ java org.openadaptor.adaptor.RunAdaptor MyAdaptor.props MyAdaptor
```

...or...

you can simply click in the traffic light icon...

and let AFEditor run the adaptor for you.

The MyAdaptor.props File

Have a look at the `MyAdaptor.props` file that was produced by AFEditor.

Notice that it has configured the FileSource using a slightly different set of configuration options that you did in previous labs and examples (see Configuring an Adaptor on page 25). This is something that you see with a number of openadaptor adaptors. It simply means that the adaptor has been enhanced over time and as new features are added, new configuration options have been introduced. Sometimes the new configuration options replace the old options. But openadaptor always try to maintain backward compatibility such that the component supports all types of configuration options.

Adding a Pipe

If you have time, you might want to try and add in a pipe that sits between the `FileSource` and `FileSink` and applies the following filters:

• Filter out records where the first name starts with a letter in the range of caps-M to caps-Z

...and then...

• of the remaining records, pass only those that do **not** have the number 2 at the start of there phone number

This should filter the input records and only output records for `Geoff Bovis` and `Fred Saddit`.

Well done! You have reached the end of the lab.

Possible Problems

- AFEditor can only configure components that it knows about

  When someone develops a new openadaptor component they have to also produce some configuration information to allow AFEditor to know how this component is to be configured. Sometimes there are components contributed to openadaptor where the developer has forgotten to provide the necessary AFEditor configuration files. This means that when you run AFEditor you are unable to see that particular component.

- AFEditor can corrupt configuration files

  If you ask AFEditor to load in a configuration file that uses a component that AFEditor does not know about, it tries to interpret what it can of these new settings and ignores those it cannot. In other words, you can easily corrupt a working configuration file. So be careful!

- The properties are in alphabetic order

  As you would have seen during the lab work, AFEditor displays the properties in alphabetic order. This can be especially hard to work with when you are setting any multiple occurrence (++) fields.

Logging

Since the release of version 1_5_0, openadaptor switched to using the log4j logging package as its logging mechanism.

To configure the openadaptor logging system you can either specify log4j directives, or continue using the pre-1_5_0 logging configuration properties, and openadaptor maps these and configures log4j accordingly.

Log Levels

The following log levels are available:

- FATAL

  Only FATAL messages are output.

  This level can also be called CRITICAL.

- ERROR

  Error messages and above are output.

- WARN

  Warning messages and above are output.

- INFO

  Info messages and above are output.

  This is the **default** setting.

- DEBUG

  Debug messages and above are output.

  This level can also be called TRACE.

Configuring Logging the Normal Way

Using the "normal", or "old style", mechanism of configuring openadaptor logging, all log messages are output to standard error (stderr).

In the adaptor configuration file, you need to specify a "Logging" component for the adaptor, and give it a name. This is achieved with the following line:

```
A.Logging.Name = Logging
```

where:

- `A` is the name of the adaptor in this example
- The name you give (on the right hand side of the equals sign) is the name of the component. You must use this name further down this configuration file when you specify each logging option

Most example configurations set the name of this logging component to be `Logging`, but you could choose any name.

You then need to configure this component.

Each property must be prefixed by the adaptor name and the name of this logging component (`A.Logging.whatever`).

The overall property for setting logging levels is `LogSetting1`, and it is this property that you set to your required logging level (Fatal, Error, etc.).

You can then set a number of further properties to specify what details you would like to see with each log message.

An example configuration block might be:

```
A.Logging.LogSetting1       = INFO
A.Logging.LogLinesToCache   = 100
A.Logging.LoggingTimeInfo   = true
A.Logging.LoggingThreadInfo = false
A.Logging.LoggingPackageInfo = false
```

where:

- `LogSetting1`

  This sets the overall level.

  It actually takes two parameters: `log_level component_name`

  If a component name is not specified, it defaults to DEFAULT.

The fact that it is "LogSetting1" implies that you might have a "LogSetting2", or 3. Well you can...

The idea is that you can set individual log levels for each component of your adaptor. However, you cannot simply mention the component by name :-(Instead, you must mention the component by its full Java class name.

For example:

```
A.Logging.LogSetting1    = INFO
A.Logging.LogSetting2    = DEBUG org.openadaptor.adaptor.standard.FileSink
```

This sets the overall log setting to be at level INFO for all components except the FileSink component which has its logging level set to DEBUG.

This can be extremely useful when you are debugging a complex adaptor, as turning DEBUG logging on as the default can produce vast amounts of output from every component of the adaptor. Being able to turn on DEBUG for a specific component is very useful.

- LogLinesToCache

  Prior to 1_5_0 this used to allow the Remote Control interface (see Remote Control on page 74) to see log output and therefore setting this value determined how much output the Remote Control interface could view at any one time.

  However, since switching to log4j this appears to no longer work.

  You can just leave out this option.

- LoggingTimeinfo

  This sets whether you want the log output to include timestamp information.

- LoggingThreadInfo

  This determines whether you want the log output to include the name of the actual java thread that produced this message.

- LoggingPackageInfo

  Whether you want to see the full java package name of the class that output each message.

If you do not configure a logging component then the adaptor logs at the default level of INFO.

Configuring log4j Directly

If you are comfortable configuring log4j, then you can directly specify log4j directives for your adaptor.

If you specify any log4j directives then this overrides any other logging settings. That is, log4j settings take precedence.

log4j allows you to specify multiple log outputs, rolling file policies, etc..

log4j directives can be specified directly within the adaptor's configuration file or you can specify a filename containing the log4j directives.

To specify a file containing the log4j directives you can either specify this file name within your adaptor configuration file by using the following option:

```
log4j.configuration = filename
```

or you can specify this option on the Java command line when you start up the adaptor:

```
$java -Dlog4j.configuration=filename org.openadaptor.adaptor.RunAdaptor ...
```

If you want to see an example log4j configuration file that you can use when running an adaptor, refer to the file:

```
sols\ex_log4j.props
```

For detailed information about configuring log4j, refer to the log4j Web site at **jakarta.apache.org/log4j/docs**.

Remote Control

The standard openadaptor controller (`SimpleController`) offers remote control capabilities.

Through remote control, you can:

- Pause a running adaptor

  This causes the adaptor to pause its operations.

  Some components are written to pause and to close any current database/ socket connections. These are then reinstated with a resume. Not all components do this - it depends on how each component was written. But as a general rule, if you have an adaptor that seems to have some problems, it is worth your while issuing a pause and resume to see if it clears out the problem.

- Resume an adaptor that is paused

  This tells the adaptor to resume operations.

  As mentioned, with some components, this causes them to reconnect to their input or output.

- Terminate (gracefully) an adaptor

  This causes the adaptor to call its final cleanup method before closing down. This is the only way to gracefully shutdown a running adaptor.

- Kill (immediately) an adaptor

  This tells the adaptor to die immediately. No cleanup method is called and so you need to use this option with caution.

- Check the status of an adaptor

  This outputs an overall status message which may or may not be of interest. It tends to indicate how many data records have been processed.

- See the last set of log output for an adaptor

  Prior to openadaptor version 1_5_0 this used to be very useful. However, since moving to log4j, this option no longer seems to output any log messages. This might one day be fixed in a later release of openadaptor.

- Change the current log level for an adaptor

  This is a useful way of upping/lowering the log level on your adaptor and/or its components whilst it is up and running.

Probably the only major task that you can not perform by remote control is to **start** an adaptor. All remote control capabilities are performed on a currently running (or paused) adaptor.

An adaptor does not automatically support remote control. It is something that must be configured into the adaptor.

To configure an adaptor to have remote control, you just need to specify the Java class that handles the remote control requests as they come into the adaptor, and then any further options that this class might require.

openadaptor provides a remote control interface, so that you can write your own remote control class. openadaptor also provides the following remote control implementations:

- Remote control over HTTP
- Remote control over TIBCO Rendezvous
- JMX compliant remote control
- RMI remote control

For the purposes of this document let's restrict ourselves to looking at remote control over HTTP.

Remote Control over HTTP

The class that provides remote control over HTTP is:
`org.openadaptor.adaptor.standard.HTTPRemoteControl`.

To configure an adaptor to offer remote control over HTTP you need to tell the controller about this remote control class. You then specify the port number over which the remote control comes in, and (optionally) a password that must then be included in the HTTP remote control requests - as a minor piece of security.

Here's an example configuration segment:

```
A.Controller.Name = Controller

A.Controller.RemoteControl.ClassName =
           org.openadaptor.adaptor.standard.HTTPRemoteControl
A.Controller.RemoteControl.HTTPPort        = 29293
A.Controller.RemoteControl.ControlPassword = specialword
```

Notice that the controller is declared as having the name "Controller" (the value on the right hand side of the equals sign). It seems that although you can change this to be whatever value you like...you should always set the controller to the name **Controller**. Otherwise the remote control does not seem to work :-(

In the above sample configuration, the remote control class name is set to be the supplied HTTPRemoteControl class. This class then requires up to two more configuration settings:

- HTTPPort

  This defaults to 80.

  This value must be **unique within all your adaptors** that use remote control. That is, no two adaptors can share the same HTTP port number for remote control. If the HTTP port is already in use then the adaptor is not able to start.

  To allow for easier management of your remote control adaptors, you might wish to refer to Adaptor Administration on page 107. This introduces a way to manage all adaptors from a single Web page.

- ControlPassword

  In the above example, the control password is set to specialword. This means that when remote control commands come in to the controller they must include this password or else they are ignored.

  If you do not specify a ControlPassword then the commands do not need to supply any special password (and if they do, it is ignored). The following warning message is output when your adaptor starts up:

```
WARN: RemoteControl password is being set to null - NOT RECOMMENDED!
```

  You do not need a password. It is just a way of adding a small amount of security to your adaptor.

Once this adaptor is up and running, you can send remote control requests using any Web browser on the network.

You just need to type in a URL that consists of the host name of the server running the adaptor, followed by a colon (:) and then the specific port number on which the adaptor is listening.

For example:

```
http://localhost:29293
```

connects you to the adaptor (assuming it is running) that is listening for remote control commands on port 29293 of this server. This then provides you with a Web page that lets you issue remote control command to this adaptor.

Lab - Using Remote Control

The purpose of this lab is to show you how to configure and use HTTP remote control for an adaptor.

For this lab you use the adaptor that was provided for a previous lab - the SOCKPRINT adaptor. You configure in remote control and then see how to use it.

Configuring the Adaptor

In the `labs` directory, find the file: `SocketPrint.props`

- Edit this `SocketPrint.props` file
- Add in the following lines:

```
# Remote Control
# --------------
SOCKPRINT.Controller.RemoteControl.ClassName =
            org.openadaptor.adaptor.standard.HTTPRemoteControl

SOCKPRINT.Controller.RemoteControl.HTTPPort        = 29293
SOCKPRINT.Controller.RemoteControl.ControlPassword = specialword
```

- Now save this configuration file
- Now run the SOCKPRINT adaptor

This runs as normal, and then say that it is waiting for connections...

Using Remote Control

- Open up a Web browser, on the same machine as the SOCKPRINT adaptor

- Type in the URL:

  http://localhost:29293

You should see a screen which looks partly like this:

The fact that this screen has appeared means that your adaptor has received the URL request and sent back this Web page. If the adaptor had not been running you would have received a Page not found message in your browser.

- Try clicking on the pause button

  It should say that an invalid password was provided.

- Enter the correct password in the box and then re-click the pause button

  You should see the text OK appear in the Web browser.

If you look back in the command window running the SOCKPRINT adaptor you should notice that it says it has paused.

- In the Web browser, click the resume button

  The SOCKPRINT adaptor resumes.

- Now select the log level DEBUG, and click the setloglevel button

- Now click the pause button

  You should see some addition DEBUG message(s) printed this time as well as the normal INFO messages.

  Great, you can change the log levels of an adaptor whilst it is running.

Now let's compare the actions of `terminate` and `kill`.

- Click the `terminate` button

  You see the SOCKPRINT adaptor terminate.

  Notice that there are some DEBUG outputs to indicate that the adaptor has called its "cleanup" method. This means that your terminate request caused the adaptor to properly close everything down and then exit.

- Re-start your SOCKPRINT adaptor
- Back in the Web browser, set the log level to DEBUG
- Now click on the `kill` button

  Notice that the adaptor simply exits! There is no controlled clean up of anything that the adaptor might have had open etc.

It is important to realize that using `terminate` is the only way to cleanly shut down a running adaptor. Only use `kill` if terminate is unable to stop the adaptor.

Now, it might have been more exciting if there was real data coming into the SOCKPRINT adaptor while you were doing all that remote control, however, you now know how to configure an adaptor to offer remote control and how to issue remote control commands.

Well done! You have reached the end of the lab.

Handling Errors

What if an adaptor encounters an error whilst processing a data record? How do you handle errors within openadaptor?

Obviously there are errors such as `unable to open input file` or `unable to open destination socket`. These are the types of errors that typically occur on adaptor start-up and there is really only one course of action, and that is to terminate the adaptor.

The errors that are of greater interest are those that occur whilst the adaptor is up and running.

openadaptor adaptors are transactional. That is, only when the last DataObject within the current message has been successfully processed, is this message committed. (If you are unsure about the definition of an openadaptor message please refer to Messages on page 57)

Default Error Handling

If an adaptor component cannot process a message, an exception is thrown. This exception is caught by the **preceding component** in the adaptor. If this component cannot cope with the exception then the exception is re-thrown, and so on until the exception reaches the source component. It is then up to the source component to decide how to deal with this exception.

The source component always rolls back the current transaction. It then depends on the type of source as to what behavior it can offer.

For example:

• JDBC Source

If the source is a JDBC source, then it is working with a database and thus could be configured to mark the source records within this message as being "in error". Exactly how you do this is configurable. You are able to specify an SQL statement to do this. Typically you might have a status

column that you can set to a value of "Error", or something like that. Once the record has been marked, the JDBC source is able to continue processing.

Alternatively, you could configure your JDBC source to simply terminate, leaving all source records untouched.

- Other

  For all other openadaptor sources, it depends on the source. It typically applies some sort of rollback - in whatever way may be appropriate for its environment - and then exit. But this behavior depends on the source.

So, the default error handling behavior is to rollback the transaction and either terminate the adaptor, or if, for example, your source is a JDBC source, you might configure it to mark the records within the database as being in error and then to continue on to the next batch of source records.

For some situations this might be acceptable behavior, but openadaptor does offer an alternative...

Message Hospitals

A message hospital can be thought of as a place to put openadaptor messages that have errors. The idea is that, rather than your adaptor getting an error and terminating, it places the error message (the array of DataObjects) into the hospital, and continues processing with the next message.

Why is it called a hospital? It is the analogy that a message is placed into the hospital when it is sick (has an error within it). Whilst in the hospital this message is called a patient. The patient can be examined. Once diagnosed, the problem can (hopefully) be cured and the patient can be discharged and made available to the adaptor for later re-processing. Of course, while being examined, it might be decided that the patient should not be discharged, but marked-dead because its illness is too severe.

Rather than getting caught up any further in the hospital analogy...let's move on to describe how a hospital can help your adaptors.

Hospital

A hospital is basically just a set of tables and stored procedures within a database. They can be placed in any database of your choosing.

When you come to configure a hospital within an adaptor, you simply specify the full connection string for how to connect to the database.

Before you can allow any adaptors to use this connection, you must set up the necessary data tables and necessary database stored procedures.

openadaptor provides a number of hospital implementations for various databases, including sysbase, MSSQL, etc.. The set up scripts for all these are located in your openadaptor installation directory, under the `sql` directory.

The main files you need are:

- `HospitalSchema.sql`

  This creates the basic data tables and views that make up a hospital.

- `HospitalProcs.sql`

  This creates the stored procedures that are used when admitting and discharging patients to/from the hospital.

If you need to set up a hospital, you simply get your database administrator to load up these sql scripts and provide you with the necessary details so you are able to connect to the database containing your hospital. There is a readme file that guides you through this.

➤ When installed as part of OVBPI, the openadaptor hospital and HAT are fully installed and configured for you. You do not need to run any set up scripts. Refer to Hospitals on page 177

Once you have created the necessary tables and procedures, your hospital is all set up to receive patients.

➤ With the standard openadaptor Version 1_6_5, the MSSQL and Oracle hospital implementation have problems and **do not** work. The OVBPI enhancements to openadaptor provide corrected MSSQL and Oracle hospital implementations that do work. Refer to Hospitals on page 177

You then configure a connection to this hospital within the configuration file of the adaptor needing the hospital.

The details you require are as follows:

```
A.H1.ClassName  = org.openadaptor.adaptor.jdbc.dbname.HospitaldbnameImpl

A.H1.JdbcUrl    =
A.H1.JdbcDriver =
A.H1.UserName   =
A.H1.Password   =
A.H1.Database   =
```

where:

- The class name would obviously depend on the underlying database that you were using. Possible values might be:

  — `org.openadaptor.adaptor.jdbc.oracle.HospitalOracleImpl`

  — `org.openadaptor.adaptor.jdbc.mssql.HospitalMSSqlImpl`

  — `org.openadaptor.adaptor.jdbc.sybase.HospitalSybaseImpl`

  — `etc.`

- The `Database` value is optional depending on your database implementation. For example, the database name is required for MSSQL, but not required for Oracle databases

- The hospital in this example is called `H1`

  This is why each line is declare as: A.**H1**.*something* ...

  You do **not** declare your hospital as a component within the adaptor. The hospital is not part of the adaptor chain of components. It is an entity in its own right.

  In this example you called the hospital `H1`. You can use any name you like.

Hospital Pipe

If a sink or pipe throws an exception, this exception is thrown back to the preceding component in the adaptor chain. If it can not handle this exception, the exception is re-thrown back to the preceding component. This continues until it reaches the source.

In openadaptor there are essentially two main types of exceptions that a component can throw:

- Hospital

  This is thrown by a component when an error has occurred that affects the processing of this message, however the error is not so bad that the adaptor should terminate.

  For example, a sink might be written to receive and process certain data types, but if it receives other data types it throws a HOSPITAL exception.

- Fatal

  This exception is thrown when something major has gone wrong. For example, a FileSink would throw this exception if it is suddenly no longer able to write to its output file. There is no point receiving any more data, so the component throws a FATAL exception.

openadaptor provide a special pipe that knows how to handle HOSPITAL exceptions. It is the **HospitalPipe**.

Pictorially, your adaptor might look like this:

Figure 12 Using A Hospital Pipe
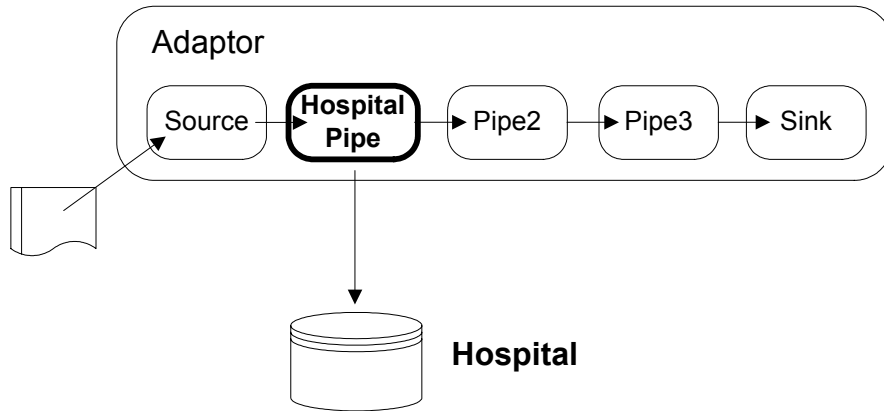
where:

- You want the hospital pipe to be able to catch any HOSPITAL exceptions, so you place it immediately after the source component

- Now if any HOSPITAL exceptions occur in Pipe2, Pipe3 or Sink, these are fed back up the component chain and handled by the hospital pipe

The hospital pipe is configured into an adaptor just like any other pipe:

- You first give the component a name:

    ```
    A.Component<n>.Name = HP
    ```

    Here the component is called HP - for hospital pipe. Any name will do.

- Next, you link the component in to be after the source:

    ```
    A.C1.LinkTo1 = HP
    A.HP.LinkTo1 = ...the next component
    ...etc...
    ```

- Then you configure the hospital pipe:

```
A.HP.ClassName            = org.openadaptor.adaptor.hospital.HospitalPipe

A.HP.HospitalName         =
A.HP.DefaultApplication   =
A.HP.DefaultSubject       =
```

The hospital pipe needs to know the name of the hospital into which it is to write its patients. All you specify here is a name. For example, H1. Later in the configuration file you configure this hospital name with the details of how to connect to the database. (As described in Hospital on page 83.)

You then specify two very important attributes. The DefaultApplication and DefaultSubject are fields in which you provide text strings. Every patient that this adaptor writes to the hospital is tagged with these two strings. This becomes important when you come to fixing up these patients and wanting to send them back for re-processing. This is discussed in a moment (see Hospital Source on page 95).

Here is an example hospital and hospital pipe configuration extract:

```
# Declare the components of this adaptor
# -----------------------------------
A.Controller.Name = Controller
A.Logging.Name    = Logging
A.Component1.Name = Src
A.Component2.Name = Sink
A.Component3.Name = HP

# The linkages
# ------------
A.Src.LinkTo1    = HP
A.HP.LinkTo1     = Sink

# Configure the FileSource
# -------------------=========
A.Src.ClassName            = org.openadaptor.adaptor.standard.FileSource
...
# Configure the FileSink
# -----------------========
A.Sink.ClassName           = org.openadaptor.adaptor.standard.FileSink
...
# The Hospital Pipe
# --===============
A.HP.ClassName             = org.openadaptor.adaptor.hospital.HospitalPipe
A.HP.HospitalName      = H1
A.HP.DefaultApplication  = MyApplication
A.HP.DefaultSubject      = MySubject

# The Hospital
# --==========
A.H1.ClassName   = org.openadaptor.adaptor.jdbc.oracle.HospitalOracleImpl
A.H1.JdbcUrl     = jdbc:oracle:thin:@localhost:1521:globsid
A.H1.JdbcDriver  = oracle.jdbc.driver.OracleDriver
##A.H1.Database  = ...not required for Oracle
A.H1.UserName    = system
A.H1.Password    = manager
```

where:

- The hospital pipe is called HP

- The hospital pipe tags its patients with MyApplication/MySubject

- The hospital pipe says to which hospital it writes its patients. In this case it is H1

- You then configure the connection details for hospital H1

Hospital Administration Tool (HAT)

If an adaptor is going to write patient records to a hospital, you need some mechanism to allow you to access these patients.

openadaptor provides the Hospital Administration Tool (HAT) and it allows you to:

- Browse patients
- Edit the patients
- Discharge patients

Setting up the HAT

▶ When installed as part of OVBPI, the openadaptor hospital and HAT are fully installed and configured for you. You do not need to run any set up scripts. Refer to Hospitals on page 177

When the hospital installation scripts run, they create the following set of data tables:

- The table that holds the actual patients

  ```
  DBusMH_Patient
  ```

- A set of tables that handle administration of the HAT itself:

  ```
  DBusMH_Attribute
  DBusMH_EditableAttribute
  DBusMH_Role
  DBusMH_User
  DBusMH_UserRole
  ```

Before you can run the HAT you need to set up a HAT user and give this person the correct permissions. The HAT gives the impression that you can configure many users, however it really only allows one. This user must have the same name as the database user that created the hospital tables. That is, if you created the hospital tables as the database user "Fred", then you must configure a HAT user called "Fred" and log on as that HAT user.

User and Roles

Using your favorite database manipulation tool, you need to create a HAT user. You do this by inserting a record into the `DBusMH_User` table. You specify the `UserName` field and assign a unique ID for the `UserID` field.

The `DBusMH_Role` table is already set up to contain the standard roles.

You need to assign your new user the roles of:

- `HATUser (1)`

  You need this level of access to be allowed to login to the HAT.

- `HATAdmin (4)`

  You need this access to be allowed to edit patients.

- `HATSecurity (5)`

  You need this level of access to set up the attributes and record-types that you are allowed to edit.

So, access the `DBusMH_UserRole` table and insert three rows that specify that your `UserID` has each of these roles (`RoleID`)

The hat.props File

Before you can start up the HAT you need to set up the `hat.props` file.

The `hat.props` file basically tells the HAT how to connect to the hospital database.

The format for the property names is slightly non-openadaptor-standard (which is a shame :-) but is reasonably easy to work out. Here is an example that points to a MSSQL hospital, called `H1-MSSQL`:

```
HAT.hospitalName.1        = H1-MSSQL
HAT.dbDriver.H1-MSSQL     = com.inet.tds.TdsDriver
HAT.dbURL.H1-MSSQL        = jdbc:inetdae7://localhost:1433
HAT.dbUser.H1-MSSQL       = dbuser
HAT.dbPassword.H1-MSSQL   = dbpass
HAT.dbHospital.H1-MSSQL   = dbschema
```

You can have multiple entries like this in the `hat.props` file to allow the user at login to select from them. You would just need to specify sequential numbers for the `hospitalName` property.

For example:

```
HAT.hospitalName.2              = AnotherHospital
HAT.dbDriver.AnotherHospital    = driver
...
```

There is an example `hat.props` file in the openadaptor `cookbook` directory. Just remember that it uses the `CookbookParams.include` file to set various database details.

Running the HAT

The `hat.props` file needs to be in your classpath when you come to actually start up the HAT.

To run the HAT, you issue the command:

```
java org.openadaptor.hospital.HAT
```

This brings up the login screen. It looks something like this:

Figure 13 The HAT Login Screen

where:

• If you had more than one hospital connection configured within the `hat.props` file, you could select them from the pull-down menu in the `Hospital` field (top-right of the screen)

- The login details default to those specified in the hat.props file. Unfortunately, the password is in plain text :-(

You then click the OK button and log in.

You are then presented with the main HAT screen:

Figure 14 The Main HAT Screen

where:

- The patients are listed down the left-hand side of the screen
- If the patient (message) contained multiple records (DataObjects) then these would be listed as SDO 1, SDO 2, etc. under the Payload Details column. SDO standard for "Simple Data Object"
- You can see the contents of the message and the reason for rejection

You can then navigate their way around the various menus and work out how to do things. The only non-obvious thing is how to "edit" attributes within a patient...

Editable Attributes

By default, a user has read-only access to the messages in the hospital. Even the user that you have set up as having `HATAdmin` and `HATSecurity` access just has read-only access.

The user who has `HATSecurity` access is able to configure who can edit what.

Having logged on as your user with `HATSecurity`, you are able to go to the menu:

    `Admin->Editable Attributes`

> (If this menu is greyed-out, then you do not have `HATSecurity` access and need to go back and set up your user access and roles again.)

Selecting the `Editable Attributes` menu brings up the Editable Attributes Administration screen.

On this screen you specify, for each data object type (DOType), which user roles are allowed to edit which attributes. You can allow access to the whole record by using the attribute tag: `EDIT_ALL_ATTRIBUTES`.

For example:

> Here you allow the `HATAdmin` user to fully edit records of type `Any` and `MyType`:

DOType	HAT User Role	Editable Attributes
Any	HATAdmin	EDIT_ALL_ATTRIBUTES
MyType	HATAdmin	EDIT_ALL_ATTRIBUTES
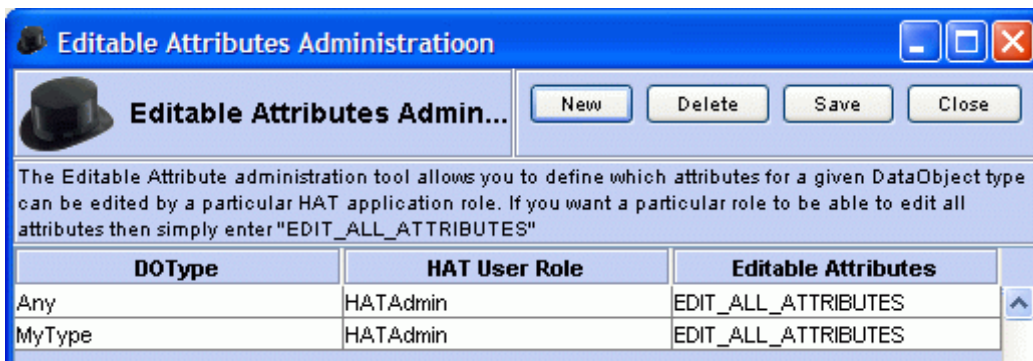
Editing a Patient

Once you have the ability to edit an attribute within a patient, you can edit the patient as follows:

- On the main HAT screen, select the patient
- Locate the attribute value you wish to modify
- Double-click on the value field for this attribute
- Make your changes ... **and press return**
- Then, to make the changes permanent, select another patient in the list of patients - this prompts the HAT to ask you if you wish to save your changes

Discharging a Patient

You simply select the patient and then:

```
Patient->Discharge
```

It is by running this Hospital Administration Tool (HAT) that you are able to investigate why a particular patient had problems, and once you feel you have sorted out the problem, you can discharge the patient.

Once discharged, how does a patient get resubmitted back into the adaptor?

Hospital Source

openadaptor provide a `HospitalSource` class which is able to pick up patients from a hospital and resend them.

The hospital source is configured like any other source. You specify the class name, configure its behavior, and link it to the appropriate pipe or sink depending an where you want the discharged hospital records to be handled.

The hospital source is configured as follows:

```
A.Component<n>.Name   = HS

A.HS.ClassName        = org.openadaptor.adaptor.hospital.HospitalSource
A.HS.HospitalName     =
A.HS.ProcessPatients  =
A.HS.PollPeriod       =
A.HS.QueryAppName     =
A.HS.QuerySubject     =
```

where:

- You assign a name to the hospital source component

- You specify the name of the hospital from which it reads

- `ProcessPatients` can be set either true or false

  You want to set this to `false` - this means that the patient record is correctly sent out in a way that other pipes and sinks can process it.

  The setting of true is for a special case which you do not need to learn about here. If you want to find out more about it then please refer to the openadaptor Web site.

- `PollPeriod` is the same as for many other sources

  The value is specified in milliseconds, and defaults to 1000 (1 sec).

  This just specifies how often the hospital source polls the hospital for discharged patients.

- The `QueryAppName` and `QuerySubject` fields

  You specify here the Application name and Subject for which you want this hospital source to look. The hospital source only picks out discharged patients that match this Application name and Subject.

You could now imagine having the following two adaptors:
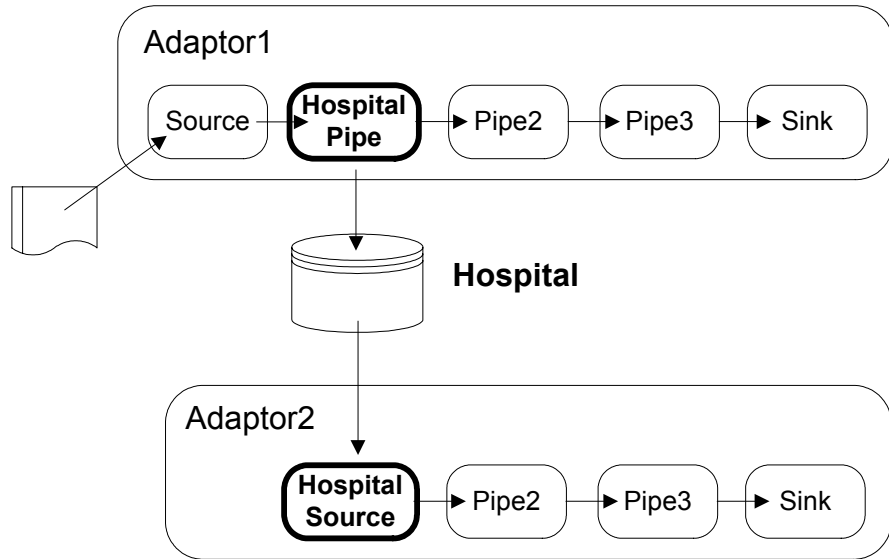
where:

- One adaptor is set up to catch any HOSPITAL exceptions and write these into a hospital

- The second adaptor is set up to periodically poll for discharged patients, and then send them back through to Pipe2

  If Adaptor2 encountered any HOSPITAL exceptions whilst trying to process a discharged patient, that patient is readmitted to the hospital, and its retry count is incremented.

- The Application and Subject must be the same for the hospital pipe in Adaptor1 and the hospital source in Adaptor2

This mechanism works just fine, however, it also means that should you change the configuration of one of the pipes within Adaptor1 you have to duplicate this change in Adaptor2.

There is an alternative which might be useful. You could configure the above scenario within one single adaptor, as follows:
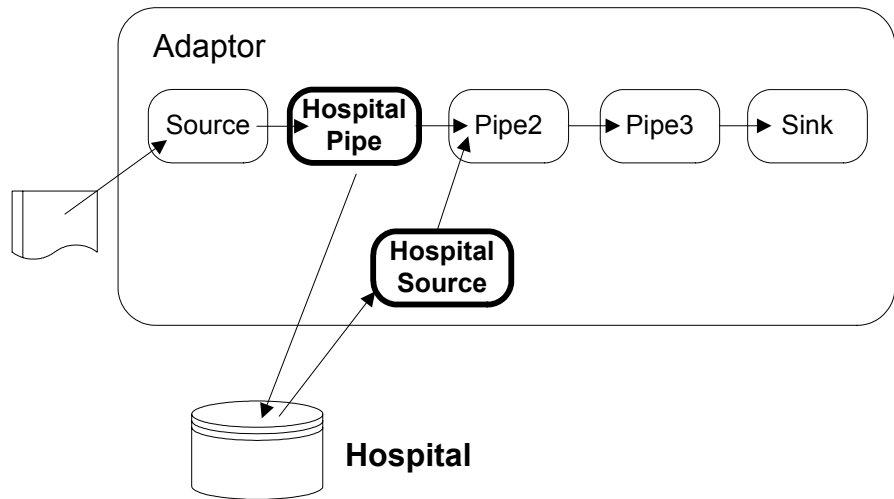
where:

- You configure the hospital source within the one adaptor configuration file
- This way makes it easier to keep the hospital pipe and hospital source using the same Application/Subject tagging

Multiple Adaptors and Hospitals

Let's just consider a few scenarios and discuss how you might go about configuring them.

Suppose you had the following scenario:

Figure 15 Two Adaptors To Different Sinks
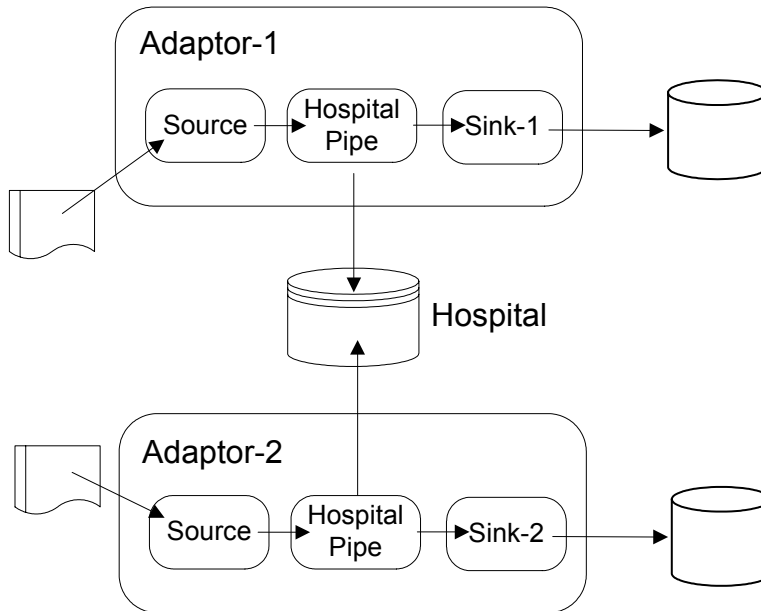
where:

- You have two adaptors, doing completely different things
- Each adaptor writes out to a different sink

If you want these two adaptors to share the same hospital then you just need to ensure that each adaptor specifies a unique ApplicationName/Subject tag.

Of course, another alternative would be to have each adaptor use its own hospital. It is your decision as to which you would find easier to manage.

Suppose you had the following scenario:

Figure 16 Two Adaptors To The Same Sink
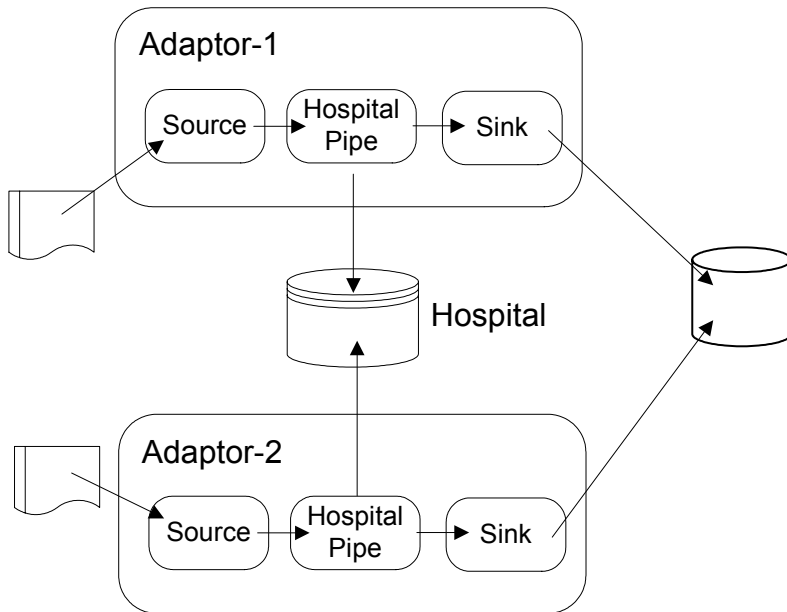
where:

- Both adaptors are sharing the same hospital
- Both adaptors are outputting to the same sink

In this scenario both adaptors can share the same hospital and they could even use the same ApplicationName/Subject tag for their patients. This is because, it does not matter which adaptor picks up the discharged patients as they are then going to send the message to the same sink.

You do not need to configure a hospital source within both of these adaptors. You can have one adaptor configured with a hospital source pulling out discharged patients for both adaptors, and just sending them on to the destination sink. Indeed, you could configure a completely separate adaptor that had the hospital source handling all the discharged patients of both adaptors.

If you have many adaptors all feeding to the same sink, you can consider having them share the same hospital and then setting up one hospital source adaptor to handle all their discharged patients. For example:

Figure 17 A Single Hospital Source Adaptor
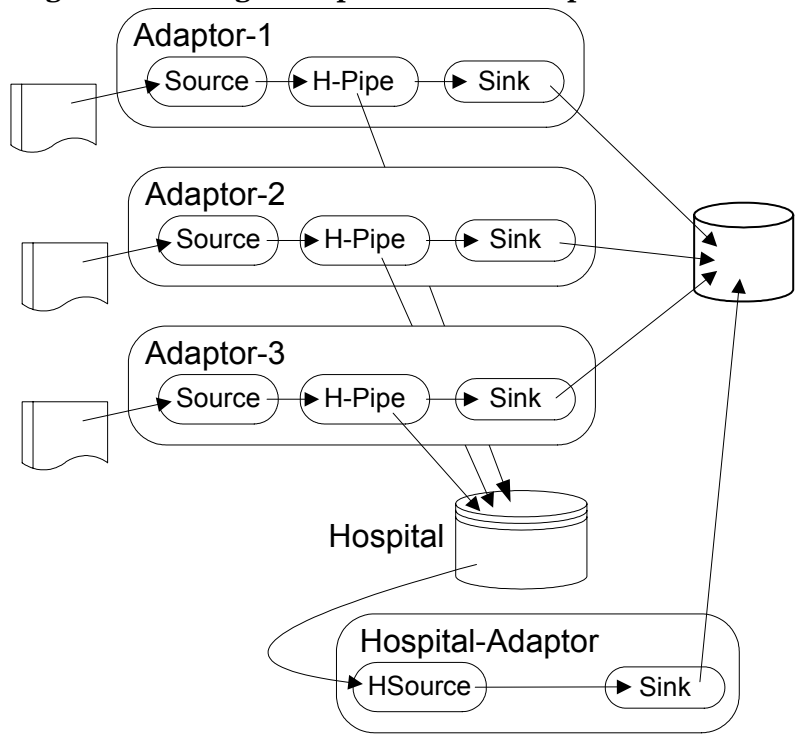
Of course, the configuration you choose depends on your actual requirements.

Summary

openadaptor is an open source adaptor integration toolkit. A collection of sources, pipes and sinks, that work as a collection of `building blocks`, enabling you to configure powerful adaptors to suit your data interchange needs.

As well as providing a vast collection of standard components, you have full access to the source code, documentations and tutorials. This means that if there is not already a component for your particular data needs, you can write your own. It then has complete plug-and-play capability with all the existing components.

openadaptor adaptors are transactional. If errors occur, the transaction is rolled back.

You can use message hospitals to trap messages that encounter errors.

Typically there is one configuration file for each adaptor. However, you could have one configuration file containing the configuration for many adaptors.

openadaptor provides a graphical tool, AFEditor, for configuring your adaptors. However, you must be very careful if using this, and always check your configuration files manually. Indeed, you may find it generally easier to build all adaptor configuration files manually, and just use AFEditor as a useful way to see what components are available, and to see their configuration options.

You can remotely control running adaptors. Remember that each adaptor must have a unique port number for receiving its remote control commands. As this can become difficult to manage, you might wish to refer to Adaptor Administration on page 107 where you can see how to manage all your remote control adaptors from a single Web page.

3 OVBPI Enhancements to openadaptor

This chapter looks at the various additions and enhancements that have been made to openadaptor to support OVBPI.

Runtime Script for Adaptors

openadaptor provide the class `RunAdaptor` which allows you to run any specified adaptor.

OVBPI provides a front-end script to this `RunAdaptor` class which calls `RunAdaptor` for you, but also presets a number of standard settings.

This script is called **biaeventadaptor.bat** and can be found in the `bin` directory of your OVBPI installation.

You use `biaeventadaptor` in much the same way as you would `RunAdaptor`, by specifying the configuration file and the adaptor name.

For example:

```
$ biaeventadaptor.bat myconf.props ABC
```

`biaeventadaptor` first looks for the specified configuration file in the current directory. If the file can not be found, `biaeventadaptor` then looks in the *OVBPI-install-dir*\data\conf\bia directory.

The idea is that you can maintain all your adaptor configuration files within the `conf\bia` directory and `biaeventadaptor` is always be able to locate them.

The `biaeventadaptor` script makes use of the environment variable `OA_ROOT`.

If you are using the installation of openadaptor that comes with OVBPI, then you do **not** need to set this environment variable. It defaults to the correct location of *OVBPI-install-dir*\nonOV\openadaptor\1_6_5.

However, if you were wanting to run an adaptor using a different installation of openadaptor, then you can set this variable to point to this other openadaptor installation directory.

For a standard installation **you should not need to set this variable** as the OVBPI installation process sets it up for you.

However, you might want to **add *OVBPI-install-dir*\bin to your PATH** environment variable so that the `biaeventadaptor` script can always be located.

biaeventadaptor Default Behavior

When you run `biaeventadaptor`, it sets up the following:

- The classpath
- Log output (stderr)
- The Adaptor history

The classpath

The classpath is constructed to include the openadaptor classes plus the OVBPI additions and enhancements. The classpath also includes the necessary JDBC libraries for both MSSQL and Oracle.

Log Output (stderr)

By default, all log output is redirected to a file in the `OVBPI-install-dir\data\log` directory. All the output for this adaptor run is found in the file:

    Adaptor_*name*.log

where *name* is the name of this adaptor.

Any previous `Adaptor_*name*.log` file (for this particular adaptor) is renamed to include the last modified timestamp for that file.

The file name format is as follows:

    Adaptor_*name*_yyyy-MM-dd_HH-mm-ss-SSS.log

For example:

You run the adaptor `ABC` and its output is saved in the file `Adaptor_ABC.log`

You then re-run the adaptor ABC, and:

— The previous log file is renamed to be (for example) `Adaptor_ABC_2003-4-15_16-6-16-218.log`. Where this timestamp is the value of the last modified time of this file.

— A new `Adaptor_ABC.log` file is opened and the log output for this run of adaptor ABC is written to it.

The Adaptor History

Some source components are capable of maintaining history information about the processing they have performed. This allows the adaptor to continue from this point the next time it is run.

It is vital that this information be correctly maintained.

The `biaeventadaptor` script specifies that all adaptor history is maintained in the *OVBPI-install-dir*\data\conf\bia directory. See AdaptorHistory on page 117 for more details about adaptor history.

Command Line Options

`biaeventadaptor` provides the following command line options:

- Background execution (**-b**)

  This runs the adaptor in the background.

- Output to stdout (**-stdout**)

  This means that all log output for this adaptor run is directed to the screen and not to the log directory.

  This option only works if you are running the adaptor in foreground.

Example Command Lines

The following are some example command lines:

- To run an adaptor in the normal way:

  ```
  $ biaeventadaptor.bat configfile.props adaptorname
  ```

- To run an adaptor where the output comes to the screen:

  ```
  $ biaeventadaptor.bat -stdout configfile.props adaptorname
  ```

Adaptor Administration

openadaptor provides the ability for an adaptor to be controlled remotely over HTTP. This is something that must be configured into every adaptor, and each adaptor must specify a unique port number.

Remote control provides:

- The ability to pause and resume an adaptor

  For some adaptors this can enable source and sink connections to be refreshed. Should you suspect that an adaptor may have encountered some problems, this may well clear things up and get the adaptor running again.

- The ability to gracefully terminate a running adaptor

  This is important as it allows an adaptor to cleanly close down any database/file connections.

The fact that you need to configure a unique (and available) port number for each adaptor makes it quite difficult to keep track of which port number you have assigned to which adaptor. This becomes quite difficult when you need to send a command to one of the adaptors, as you spend half your time trying to remember on what port number it is listening.

To make remote control easy to administer you can use the **JSP Adaptor Console** contributed utility.

Installing the JSP Adaptor Console

- Locate the OVBPI install CD

  The JSP Adaptor Console is located in the file:

OVBPI-install-CD\contrib\openadaptorConsole\openadaptorConsole.war

- Copy this openadaptorConsole.war file into the directory:

  OVBPI-install-dir\nonOV\jakarta-tomcat-5.0.19\webapps

- Using the OVBPI Administration Console, stop and start the Servlet
  Engine component

  The Servlet Engine sees this new Web application and expands it into the
  webapps\openadaptorConsole directory, ready for use.

Configuring the JSP Adaptor Console

Edit the webapps\openadaptorConsole\HTTPRemoteControl.props file.

This file tells the JSP Adaptor Console where to look for its adaptors.

The format of the file is as follows:

```
Dir1 = D:/mydir/myadaptors
Dir2 = D:/another/directory
Dir<n> = ...etc...

File1 = c:/my/special/file.props
File<n> = ...etc...

Remote1 = remoteHost:11223
DisplayName1 = My Adaptor on MachineX
Remote<n> = ...etc...
```

where:

- Dir<n>

  You can specify a directory name and the JSP Adaptor Console searches
  all .props and .properties files, and extracts the remote control
  details for every adaptor it finds. You can specify as many directories as
  you like.

- File<n>

  You can specify individual files by name.

The JSP Adaptor Console can quite happily cope with configuration files that contain more than one adaptor configuration.

- Remote<n>

    If for some reason you do not have access to an adaptor's configuration file, you can hard code a connection using the Remote<n> option. You specify the host name followed by a colon(:) followed by the port number. You can optionally specify a DisplayName<n> which appears within the JSP Adaptor Console. If there is no display name specified, the JSP Adaptor Console simply shows the hostname:portname values.

If you keep all your adaptor configuration files within one or two directories, you simply need to use the Dir<n>= option(s). This way, as you create more adaptor configuration files in this(these) directories, your JSP Adaptor Console sees them.

Running the JSP Adaptor Console

Simply browse to the openadaptorConsole directory. For example:

http://localhost:44080/openadaptorConsole

The JSP Adaptor Console goes through all the adaptor configuration files specified by the HTTPRemoteControl.props file, and then builds a menu frame containing the appropriate remote control link to each adaptor. This adaptor list is then sorted alphabetically.

The result is displayed with the adaptors appearing in the left hand frame of the Web page.

The result might look something like this:

Figure 18 The JSP Adaptor Console
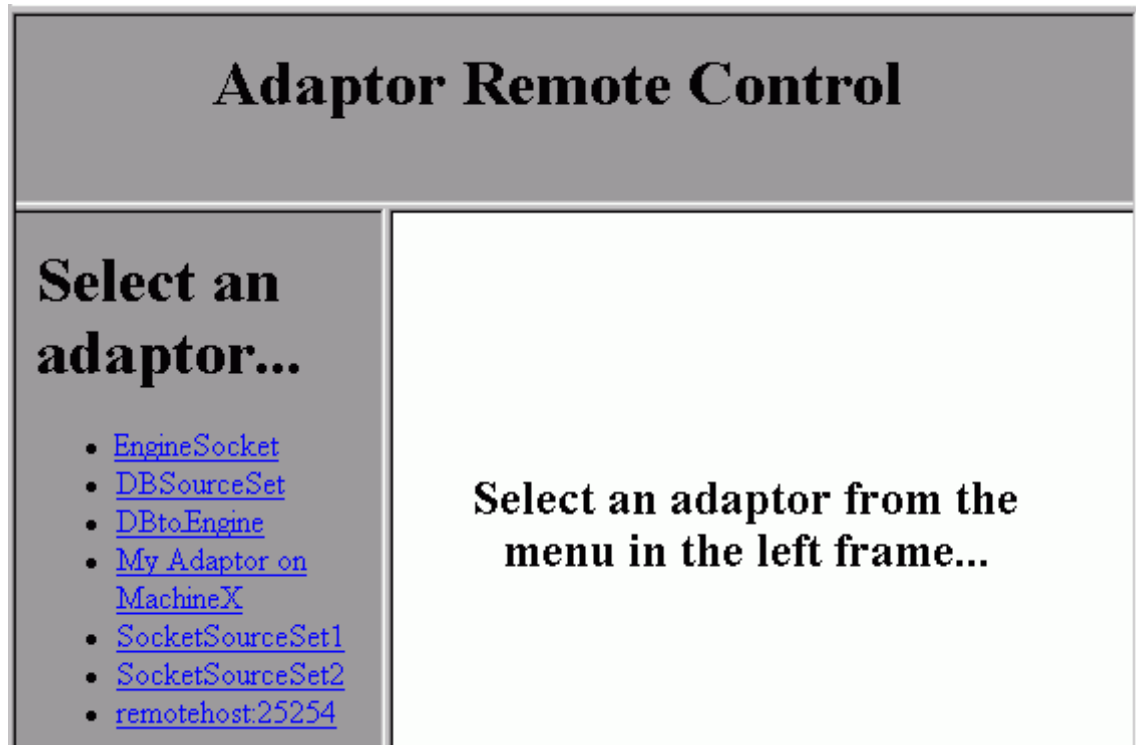
Refreshing the JSP Adaptor Console

If you make any modifications to the HTTPRemoteControl.props file, or you add new adaptor configuration files to directories listed within the HTTPRemoteControl.props file, simply **refresh the browser** and it picks up these changes straight away.

Example Configurations

Here are some example `HTTPRemoteControl.props` files:

```
Remote1 = remotehost:11111
Remote2 = remotehost2:25254
DisplayName2 = My Adaptor on MachineX

File1=c:/test/sources.props
File2=D:/work/demo/mytest.props
File3=d:/work/newdir/testa.txt

Dir1 = d:/lab/events-tg/sols
Dir2 = d:/lab/events-tg/sols/hospitals
Dir3 = c:/workarea/examples
```

...or...

If you keep all your adaptor configuration files within one directory, your `HTTPRemoteControl.props` files might simply look like this:

```
Dir1 = d:/work/adaptors
```
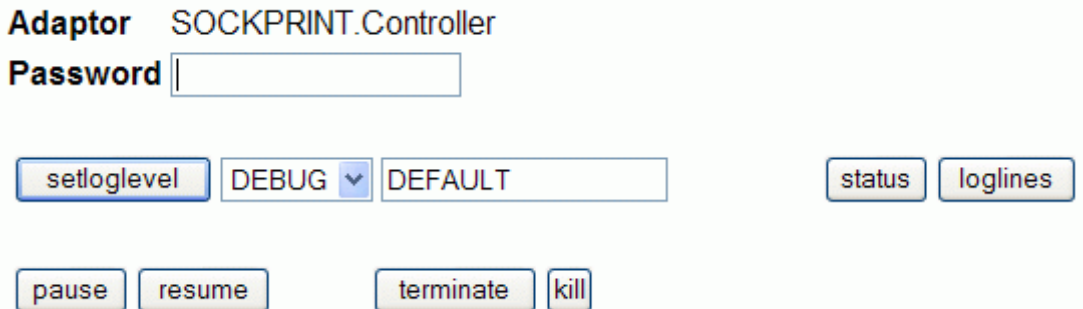
Modified Remote Control Web Page

The section Remote Control on page 74, looked at the standard Web page that is drawn when you access an adaptor over HTTP remote control.

For OVBPI, there are some slight alterations to this screen.

The new remote control screen for an adaptor now looks like this:

Figure 19 Adaptor Remote Control Screen

The page operates in exactly the same way as the standard openadaptor remote control page. The only difference is that the buttons are laid out in a (hopefully) more user friendly manner.

Lab - Using the JSP Adaptor Console

The purpose of this lab is to configure and use the JSP remote control Adaptor Console and to run an adaptor using the new `biaeventadaptor` script.

Install the JSP Adaptor Console

Install the JSP Adaptor Console contributed utility as described in the instructions Installing the JSP Adaptor Console on page 108

Configure the JSP Adaptor Console

- Edit the `openadaptorConsole\HTTPRemoteControl.props` file:
  - Find the line that sets `Dir1=`
  - Add a new line after this line, and set `Dir2` to point to your `labs` directory

    This should point to the directory that contains your lab work from the previous chapter.
  - Save this file
- Start up a Web browser and type in the URL to connect to your `openadaptorConsole`

  By default, it should be something like:

      http://localhost:44080/openadaptorConsole

  where `44080` is the configured default port for the OVBPI servlet engine.

  This should automatically bring up the JSP Adaptor Console, and you should see your SOCKPRINT adaptor as an available option in the left-hand pain of the Web page.

  If you place the mouse cursor over this SOCKPRINT link you see that it is a connection to remote control port as configured in the SOCKPRINT adaptor's configuration file.
- If you try and click on this SOCKPRINT link now, with your SOCKPRINT adaptor not yet running, then it fails with a "Page not found" error

So at this point, you have your JSP Adaptor Console up and running...but you probably do not have any remote controllable adaptors running.

Using the biaeventadaptor Script

Let's start up the SOCKPRINT adaptor...**but**...this time let's use the new biaeventadaptor script:

- Make sure that you have added the *OVBPI-install-dir*\bin directory to you system PATH environment variable

- Open a new command window

- Go to your labs directory, and run your adaptor as follows:

  ```
  $ biaeventadaptor SocketPrint.props SOCKPRINT
  ```

  Your command window appears to stop responding!!! This is because the adaptor is running and the output has been redirected to the *OVBPI-instal-dir*\data\log directory.

- Look in the *OVBPI-install-dir*\data\log directory, and view the file: Adaptor_SOCKPRINT.log

  This should show that your adaptor is indeed running correctly.

- Now go back to your JSP console and click on the SOCKPRINT link

  You should now be able to pause and resume the adaptor. Check the adaptor's log output to make sure you see that the adaptor is indeed being paused, and resumed.

- From the JSP Adaptor Console, terminate the adaptor

  Notice that your command window (that was running the adaptor) now has the command prompt back.

- Run the adaptor again

- Now check the log file

  Notice that the previous log file has been renamed - with a timestamp - and the adaptor is now logging to a new file called Adaptor_SOCKPRINT.log

- Terminate this adaptor
- Now run the adaptor specifying the **-stdout** command option

  This should direct the adaptor's log output to your screen.

  In this situation, **no** output is written to any log files.

Well done! You have reached the end of the lab.

Files

The main file handler within standard openadaptor is the FileSource. OVBPI wanted to extend this in a number of ways. Extension such as:

- The ability to continually tail (poll) a data file

- The ability to handle binary data within files

- The ability to roll through log files in a sequence

    For example, files where their name contains a date/time stamp.

- The ability to save file position when shutting down - so the next run would continue from where it left off

- The ability to handle a multi-character delimiter within delimited text files

As a result, the following file-based adaptors were developed:

- `FilePollSource`

- `FileRollSource`

- `FileBinarySource`

These are all within the Java package `org.openadaptor.adaptor.standard`

Two additional Java classes were also developed:

- `org.openadaptor.adaptor.util.AdaptorHistory`

    This is a generic class that allows a source to store/maintain its positional and status information between successive runs.

- `org.openadaptor.dostrings.DelimitedStringReader`

    This is an extension of the existing `DelimitedStringReader class` to allow it to handle a multi-character delimiter.

AdaptorHistory

Source components can be configured to maintain their position between invocations. This is achieved by using the `org.openadaptor.adaptor.util.AdaptorHistory` class.

The `AdaptorHistory` class maintains the adaptor history details using two files for each adaptor:

1. `AdaptorHistoryDB.`*`name`*`.`*`component`*`.props`, and

2. `AdaptorHistoryDB.`*`name`*`.`*`component`*`_BACKUP.props`

where:

- *name* corresponds to the name of the adaptor

- *component* corresponds to the name of the source component within this adaptor

For example:

```
AdaptorHistoryDB.MyAdaptor.C1.props
AdaptorHistoryDB.MyAdaptor.C1_BACKUP.props
```

The use of two files is simply to guard against the situation where the adaptor is killed whilst writing the actual history file.

The information written to these `AdaptorHistroyDB` files can be whatever the source component needs to save. For the `FileRollSource` component this is typically the file name being processed and the current line in that file. This allows an adaptor to start from the position it reached on its last invocation.

The AdaptorHistory.Directory Property

It is vital that each time an adaptor is run it is able to locate any previously saved adaptor history.

By default, the `AdaptorHistory` class saves the `AdaptorHistoryDB` files in the current working directory. That is, the directory in which you are running your adaptor. Because you may well run you adaptor from different directories, you can specify the property `AdaptorHistory.Directory` on the adaptor Java command line. This tells the `AdaptorHistory` class the directory in which to find/save the adaptor history for this adaptor.

For example:

```
$ java org.openadaptor.adaptor.RunAdaptor file.props adaptor
            -property AdaptorHistory.Directory /ovbpi/history
```

would save any adaptor history into the directory `/ovbpi/history`.

If you use the `biaeventadaptor` script then this property is automatically configured for you. `biaeventadaptor` configures all adaptor history to be saved in the *OVBPI-install-dir*\data\conf\bia directory.

FilePollSource

The standard openadaptor `FileSource` processes its input file from start to finish. Once it reached end-of-file (EOF) the adaptor exits.

`FilePollSource` gives users the ability to be able to run a file adaptor such that it can process the contents of the file and then (optionally) continue to "poll" that file (sometimes called "tailing") awaiting any new data.

The `FilePollSource` also allows an adaptor to save file position information. Thus, next time the adaptor is run, it simply continues from where it last saved its position.

To support these new options, the following new configuration properties have been added over and above the standard `FileSource` properties:

```
PollSource=

SaveFilePosition=
SaveFilePositionAtRecordInterval=
```

where:

- `PollSource=`

  The default setting is `false`. When set to `false`, the `FilePollSource` component processes all the remaining rows in the input file and then terminates.

  By setting `PollSource` to `true`, you are telling `FilePollSource` to continue to run and, when it reaches the end of file (EOF), wait for more rows to be appended to the input file, and continue to process these rows as and when they appear.

- `SaveFilePosition=`

  The default setting is `false`.

  When `SaveFilePosition` is set to `true`, the `FilePollSource`, at startup, tries to locate any existing `AdaptorHistoryDB` file. If there is a history file for this adaptor then the `FilePollSource` starts at the input record specified. When the `FilePollSource` reaches the end-of-file (EOF), it saves its current file position (line number). If `PollSource` is set to `true`, then it continues to poll the input file awaiting more records to be appended. Each time a new EOF is reached, `FilePollSource` saves its new file position.

For details of where the file position information is saved to disk, refer to AdaptorHistory on page 117

- `SaveFilePositionAtRecordInterval=`

  This property requires `SaveFilePosition` to be set to `true`.

  This property allows you to say how often you would like the file position information to be saved to disk. For example, you could set the record interval to be `5` and this would save the file position after every five records have been processed by the `FilePollSource` component. You could set the interval to `1` and this would save the file position after every record.

  Obviously, setting this property has an impact on the performance of the adaptor. However, you have the choice. Maybe you are willing to accept a slightly slower adaptor knowing that, if any unexpected errors occur, the adaptor is able to restart exactly where it left off.

  For example:

  ```
  A.C1.SaveFilePosition = true
  A.C1.SaveFilePositionAtRecordInterval = 1
  ```

  This would save the input file position information to disk after processing every record.

  If you specify `SaveFilePosition = true,` but do not specify the `SaveFilePositionAtRecordInterval` property, then the file position information is saved each time the `FilePollSource` reaches the end of file.

You configure the `FilePollSource` component in exactly the same way as you would a normal `FileSource`, but with these additional properties.

Example Configuration - Saving at EOF

```
A.C1.ClassName      = org.openadaptor.adaptor.standard.FilePollSource

A.C1.InputFileName  = delimited.txt

A.C1.DOStringReader = org.openadaptor.dostrings.DelimitedStringReader

A.C1.NumAttributes  = 4

A.C1.SaveFilePosition = true
A.C1.PollSource       = true
```

Example Configuration - Saving After Every Record

```
A.C1.ClassName      = org.openadaptor.adaptor.standard.FilePollSource

A.C1.InputFileName  = delimited.txt

A.C1.DOStringReader = org.openadaptor.dostrings.DelimitedStringReader

A.C1.NumAttributes  = 4

A.C1.SaveFilePosition                 = true
A.C1.SaveFilePositionAtRecordInterval = 1
A.C1.PollSource                       = true
```

FileRollSource

`FileRollSource` was developed to support the situation where a source of data records may come from not just one file, but a series of files.

You are able to configure the rolling mechanism used to move from file to file.

There are two rolling mechanisms available:

- `org.openadaptor.adaptor.standard.FileSelectByModifiedTime`

  This mechanism allows you to configure a directory name, and the files are read from that directory as and when they appear. If more than one file is present, the files are read in order of their last modified time.

- `org.openadaptor.adaptor.standard.FileSelectByRollingDate`

  This is the default rolling mechanism if none is specified.

  This mechanism allows you to roll from file to file based on a configurable and predictable file naming scheme.

The configuration property that you set to configure the file selection mechanism is:

```
FileSelector=
```

For example,

```
A.C1.FileSelector = org.openadaptor.adaptor.standard.FileSelectByModifiedTime
```

where:

- The adaptor name is `A`
- The component name within this adaptor is called `C1`

FileSelectByModifiedTime Properties

Once you have selected the `FileSelectByModifiedTime` file selection mechanism, you need to configure its behavior.

The following configuration options are available:

```
InputDirectoryName=
InputFileExtension<n>=
ArchiveFileWhenProcessed=
ArchiveDirectory=
PollPeriodWhenDirEmpty=
StayOnLastFile=
```

where:

- `InputDirectoryName=` (required)

    This is where you specify the directory to be monitored.

    You can specify a full directory path name, or a relative path name (relative to where you are running the adaptor).

    Files are read (processed) from within this directory as and when they appear - in order of their modification time stamp.

    By default, input files are deleted once they have been processed.

    Any third party application that wants to create a file for `FileRollSource` to read, must make sure the file is complete before placing it in this directory. Otherwise, the `FileRollSource` processes the file the moment it appears in the directory and (possibly) before the 3rd party application has had a chance to write any data records inside it. The 3rd party application could initially create the file in another directory, and when the file is complete and ready to be processed, rename the file to place it in the input directory. Alternatively, if `FileRollSource` is configured to only pick up files that end in a particular extension (see the `InputFileExtension<n>=` property), the 3rd party application can create the file in the input directory but with a different file extension. When this file is complete and ready to be processed, the 3rd party application can rename the file to have the correct file extension that is being used by `FileRollSource`.

- `InputFileExtension<n>=` (optional)

    If you do not specify this property, then any files that appears in the input directory are processed.

The `InputFileExtension<n>=` property allows you to limit the processing of file to only those that have a matching file extension. You can specify multiple file extensions as follows:

```
A.C1.InputFileExtension1=.xml
A.C1.InputFileExtension2=.myxml
```

In actual fact, the value you specify as an extension just has to match the end of the file name. For example, you could specify the following set of values:

```
A.C1.InputFileExtension1=.txt
A.C1.InputFileExtension2=last word
A.C1.InputFileExtension3=.csv
```

This would process all file names that ended with `.txt`, `.csv` and filenames that simply ended with the string `last word`.

- `ArchiveFileWhenProcessed=` (optional)

By default, once an input file has been fully processed, the file is deleted. If you wish to save each file once it has been processed, you can specify archiving.

Setting `ArchiveFileWhenProcessed=true` specifies that you want to archive the files once they have been processed.

If you set this property to true then you must also set the `ArchiveDirectory=` property.

- `ArchiveDirectory=` (optional)

This is where you specify the name of the directory in which you wish to archive (save) the input files once they have been processed.

You can specify either a fully qualified directory path name or a path name relative to where you adaptor is running.

You should specify a directory that is different to your input directory.

- `PollPeriodWhenDirEmpty=` (optional)

This specifies how often the input directory is polled waiting for the next input file to appear, when the input directory is empty.

You specify this value in milliseconds. The default is 1000 (1 second).

- `StayOnLastFile=` (optional)

  This specifies the behavior when you have read to the end of the last file in the input directory. Setting this property to `true` says that when you reach the end-of-file on the last file in the input directory you wish to stay on that file waiting for further data.

  The default setting is `false`.

FileSelectByRollingDate Properties

If you have selected the `FileSelectByRollingDate` file selection mechanism, you need to configure its behavior.

If you do not select a rolling mechanism, then it defaults to this `FileSelectByRollingDate` mechanism.

The `FileSelectByRollingDate` file selection mechanism is able to roll from one file to the next where the file names consist of a constant prefix and suffix but contain a date/time stamp to differentiate themselves.

The `FileSelectByRollingDate` file selection mechanism has the following set of properties:

```
InputFileName     =
InputFilePrefix   =
InputFileSuffix   =
InputFileFormat   =
InputRollUnit     =
InputRollIncrement =
PollSource        =
SaveFilePosition  =
```

where:

- `InputFileName` (required)

  Specifies the fully qualified name of the first file the adaptor should open and read.

  The file name part of this value must correspond to the Prefix/Format/ Suffix specifications that follow.

- `InputFilePrefix` (optional)

  Specifies any constant prefix for the file name. This option can also be left blank.

- `InputFileSuffix` (optional)

  Specifies any constant suffix for the file name. This option can also be left blank.

- `InputFileFormat` (required)

  Specifies the date/time stamp part of the filename the rolls. This is specified using the characters:

  — `yyyy` - to specify a four-digit year

- — `MM` - to specify a two-digit month

- — `dd` - to specify a two-digit day

- — `HH` - to specify a two-digit hour

- — `mm` - to specify a two-digit minute

Your pattern does not need to include all of these options.

Some example:

```
InputFileFormat=yyyy-MM-dd_HH-mm
InputFileFormat=yyyy-MM-dd
InputFileFormat=mm-yyyy__dd-sometext-HH
```

- • `InputRollUnit` (**required**)

  Specify which part of the `InputFileFormat` string is incrementing as each file is processed.

  Valid settings are:

  - — `y` - to specify that the year is rolling

  - — `M` - to specify that the month is rolling

  - — `d` - to specify that the day is rolling

  - — `H` - to specify that the hour is rolling

  - — `m` - to specify that the minute is rolling

  Clearly, for this to work, you must make sure that your format pattern does indeed contain the roll unit. It would be very foolish to specify a format pattern of:

  `yyyy_MM_dd`

  and then specify a roll unit of: `m`

  This is identified at run time and cause the adaptor to shutdown after processing the first file.

- • `InputRollIncrement` (**required**)

  Specify by how much to increment the roll unit. Must be a value greater than zero (0).

Overall FileRollSource Properties

There are some overall configuration properties for the `FileRollSource` component that are worth understanding. These apply no matter which file selection mechanism you have configured:

```
PollSource=
PollPeriod=
SaveFilePosition=
SaveFilePositionAtRecordInterval=
IgnoreNonExistPrevFile=
```

where:

- `PollSource=`

  The default setting is `false`. When set to `false`, the `FileRollSource` component processes all input files and then terminates.

  By setting the polling to `true`, you are telling the `FileRollSource` component to continue to run and wait for more files to appear - and process these as and when they are created.

- `PollPeriod=`

  This specifies how long the `FileRollSource` component waits in between each input file. That is, if there are (for example) two files ready to be processed, the first file is processed, there is a delay of `PollPeriod` milliseconds, and then the second file is processed, and so on.

  You must specify this value in milliseconds. The default is 1000 (1 second).

  You typically set this value to zero (0) so there is no delay when completing one input file and moving on to the next.

- `SaveFilePosition=`

  This specifies whether the component saves its file position as it processes each file. The default is to not save file position.

  If set to true, the `FileRollSource` component saves its file position - both the name of the current file and the position within that file - when it reaches the end of each input file. The adaptor also saves its file position information when the adaptor terminates.

  For details of where the file position information is saved to disk, refer to AdaptorHistory on page 117

- `SaveFilePositionAtRecordInterval=`

  This property requires `SaveFilePosition` to be set to `true`.

  This property allows you to specify how often you would like the file position information to be saved to disk. For example, you could set the record interval to be `5` and this would save the file position after every five records have been processed by the `FileRollSource` component. You could set the interval to `1` and this would save the file position after every record.

  Obviously, setting this property has an impact on the performance of the adaptor. However, you have the choice. Maybe you are willing to accept a slightly slower adaptor knowing that, if any unexpected errors occur, the adaptor is able to restart exactly where it left off.

  For example:

  ```
  A.C1.SaveFilePosition = true
  A.C1.SaveFilePositionAtRecordInterval = 1
  ```

  This would save the input file position information to disk after processing every record.

  If you specify `SaveFilePosition = true,` but do not specify the `SaveFilePositionAtRecordInterval` property, then the file position information is saved each time the `FileRollSource` reaches the end of a file.

- `IgnoreNonExistPrevFile=`

  This property can be useful when you are saving file position.

  When the `FileRollSource` component starts up, and you have `SaveFilePosition=true,` the component reads its adaptor history and tries to open the previously active input file. By default, if that file has been removed, the component fails to start up.

  However, by setting `IgnoreNonExistPrevFile=true,` the `FileRollSource` component, at start up, ignores any missing previous input file. If you have configured the `FileSelectByModifiedTime` file selection mechanism, the `FileRollSource` simply reads the input directory afresh and starts at the oldest file. If you have configured the `FileSelectByRollingDate` file selection mechanism, the `FileRollSource` starts with the original `InputFileName` and continues from their.

  The default setting is `false`.

Example Configurations

Process All Files Within a Directory

```
A.C1.ClassName    = org.openadaptor.adaptor.standard.FileRollSource
A.C1.FileSelector = org.openadaptor.adaptor.standard.FileSelectByModifiedTime

A.C1.InputDirectoryName = C:/input/files
A.C1.PollSource         = true

A.C1.PollPeriodWhenDirEmpty = 5000
A.C1.PollPeriod             = 0

...the input record layout...
```

This example configures the `FileRollSource` as follows:

- `A.C1.ClassName`

  Your `C1` component is to be the `FileRollSource`.

- `A.C1.FileSelector`

  This configures the `FileRollSource` component to use the `FileSelectByModifiedTime` file selection mechanism.

- `A.C1.InputDirectoryName`

  The component processes files from the directory `C:/input/files`

  Notice that you specify a forward slash (/) even on a windows machine because this property is going to handled by Java - and Java likes forward slashes.

- `A.C1.PollSource`

  The input directory continues to be polled even when there are no files in the directory.

- `A.C1.PollPeriodWhenDirEmpty`

  Whenever the input directory is empty, it is polled every five second (5000) looking for new files to process.

- `A.C1.PollPeriod`

  If there are more files in the input directory waiting to be processed, there is no delay when moving on to the next file.

- Each input file is deleted once it has been processed

Process All Files Within a Directory - and Archive

```
A.C1.ClassName    = org.openadaptor.adaptor.standard.FileRollSource
A.C1.FileSelector = org.openadaptor.adaptor.standard.FileSelectByModifiedTime

A.C1.InputDirectoryName       = C:/input/files

A.C1.PollSource               = true

A.C1.PollPeriodWhenDirEmpty   = 5000
A.C1.PollPeriod               = 0

A.C1.ArchiveFileWhenProcessed = true
A.C1.ArchiveDirectory         = C:/input/ArchiveDir

...the input record layout...
```

where:

- A.C1.ArchiveFileWhenProcessed
  A.C1.ArchiveDirectory

  These two properties tell the `FileRollSource` component that, as each input file is processed, the input file is archived (saved) in the directory `C:/input/ArchiveDir`

Process Selected Files Within a Directory

```
A.C1.ClassName    = org.openadaptor.adaptor.standard.FileRollSource
A.C1.FileSelector = org.openadaptor.adaptor.standard.FileSelectByModifiedTime

A.C1.InputDirectoryName    = C:/input/files
A.C1.InputFileExtension1   = .txt
A.C1.InputFileExtension2   = .csv

A.C1.PollSource            = true
A.C1.PollPeriodWhenDirEmpty = 5000
A.C1.PollPeriod           = 0

...the input record layout...
```

where:

- A.C1.InputFileExtension1
  A.C1.InputFileExtension2

  You can list as many extensions as you need. Just make sure that you increment the number at the end of the word `InputFileExtension`.

These two `InputFileExtension<n>` lines mean that only files ending in `.txt` or `.csv` are processed from the input directory. All other files are ignored.

Roll Through a Sequence of Files

```
A.C1.ClassName            = org.openadaptor.adaptor.standard.FileRollSource

A.C1.InputFileName        = d:/work/File_1990-09-27__12-36.log
A.C1.InputFilePrefix      = File_
A.C1.InputFileSuffix      = .log
A.C1.InputFileFormat      = yyyy-MM-dd__HH-mm
A.C1.InputRollUnit        = d
A.C1.InputRollIncrement   = 2

...the input record layout...

A.C1.SaveFilePosition     = true
A.C1.PollSource           = true
```

This would process files in the order:

```
File_1990-09-27__12-36.log
File_1990-09-29__12-36.log
File_1990-10-01__12-36.log
File_1990-10-03__12-36.log
...etc...
```

As it reaches the end of each file, it determines if the next file (of the name it is expecting) exists. As `PollSource` is set to true, if the next file does not exist, it keeps polling (tailing) the current file. As it reached EOF on this file it again tests for the existence of the next file. Only once that next file is present does it close the current file and roll onto the next.

FileBinarySource

The standard openadaptor `FileSource` is based upon reading "strings" from the input file.

`FileBinarySource` was developed to be able to read files that contained a mixture of string and binary data, so long as the file has a fixed record format.

Although based on the `FileSource`, the `FileBinarySource` is not an extension of `FileSource`, thus you see similar but different configuration options.

There are two sets of configuration options.

Overall Properties

```
InputFileName     = Fully qualified file name
InitialByteOffset = Bytes to ignore off the front of the file
Type              = Any string value
BatchSize         = Any number greater than zero(0)
NumAttributes     = Any number greater than zero(0)
PollSource        = true|false
SaveFilePosition  = true|false
```

Field Specific Properties

```
FieldName<n>      = Any string value
FieldType<n>      = (String|Integer|Short|Long|Double|Float)
FieldWidth<n>     = (only for strings - specified in bytes)
FieldTrim<n>      = (only for strings - true or false)
```

where:

- `InputFileName` (required)

  Specify the fully qualified file name of the input file.

- `InitialByteOffset` (optional - defaults to zero(0))

  Specifies how many bytes to ignore counting from the front of the file.

- `Type` (optional - defaults to "Any")

  Specifies the DO Record type.

- `BatchSize` (defaults to one(1))

  Specifies how many input records are to be batched together with each read from the input file.

  You must specify a value greater than zero(0).

- `NumAttributes` (required)

  Specifies the number of attributes (fields) that make up the input data record.

- `PollSource` (optional - defaults to false)

  Specifies whether you want to poll (tail) the input source file.

- `SaveFilePosition` (optional - defaults to false)

  Specifies whether you want the adaptor component to save its file position information when the adaptor is shut down.

  If you are saving this information then you must always shut down this adaptor gracefully.

- `FieldName<n>` (required)

  Specifies the name of this numbered field within the data record.

  If you specify the name `__FILLER` (that is underscore underscore `FILLER`) and the data type is "String", then this field does not appear within the output data record.

- `FieldType<n>` (optional - defaults to String)

  Specifies the data type of the field.

  Valid values are: String, Integer, Short, Long, Double, or Float.

  The actual data values, within the file being read, must be in the binary form expected by Java. For example: Short - two bytes, Integer - four bytes.

  These names are not case sensitive.

- `FieldWidth<n>` (only valid for string data types)

  Specifies the size in bytes of a String field. It is required for a string field and must be a number greater than zero(0).

- `FieldTrim<n>` (only valid for string data types)

  Possible values are true or false; this field is optional.

  If set to true, then the string field has leading and trailing white spaces removed.

Example Configuration

```
A.C1.ClassName          = org.openadaptor.adaptor.standard.FileBinarySource

A.C1.InputFileName      =  input.bin
A.C1.InitialByteOffset  =  7

# Define the file (record) layout

A.C1.Type               = MyRECORD
A.C1.NumAttributes      = 8

A.C1.FieldName1         = Name
A.C1.FieldWidth1        = 10
A.C1.FieldTrim1         = true

A.C1.FieldName2         = Quantity
A.C1.FieldType2         = Integer

A.C1.FieldName3         = Description
A.C1.FieldWidth3        = 12

A.C1.FieldName4         = AmountD
A.C1.FieldType4         = Double

A.C1.FieldName5         = AmountS
A.C1.FieldType5         = short

A.C1.FieldName6         = AmountF
A.C1.FieldType6         = float

A.C1.FieldName7         = AmountL
A.C1.FieldType7         = long

A.C1.FieldName8         = __Filler
A.C1.FieldWidth8        = 2
```

where the output record from this configuration consists of the attributes:

```
Name
Quantity
Description
```

```
AmountD
AmountS
AmountF
AmountL
```

The attribute called __FILLER does not form part of the output data record. In this case, the __FILLER attribute was specified to skip over the carriage return line-feed characters at the end of each line.

DelimitedStringReader

The standard openadaptor delimited string reader lets you specify a single-character delimiter only. This class has been enhanced by adding support for a multi-character delimiter.

You configure the org.openadaptor.dostrings.DelimitedStringReader class just as you would normally, however, you are now able to additionally specify the property **FieldDelimiterString**. If you specify this property it takes precedence over any FieldDelimiter that may have been specified.

Example Configuration

```
A.C1.InputFileName   =  multiCharDelimited.txt

A.C1.ClassName       = org.openadaptor.adaptor.standard.FileSource
A.C1.DOStringReader = org.openadaptor.dostrings.DelimitedStringReader

A.C1.FieldDelimiterString   = _|@
A.C1.NumAttributes          = 4
```

This can now correctly read the following example input record:

```
Field1_|@Field2_|@Field_|@Field4
```

Lab - File Adaptors

The purpose of this lab is to gain some experience configuring some of the file source components:

- FilePollSource
- FileRollSource

FilePollSource

The main two features of this component are that it can continuously poll (tail) a file, and that it can save file history. Let's configure this component and then try out the various options.

In your `labs` directory, locate the configuration file `FilePoll.props`.

- Edit this file such that it reads an input file called: `pollfile.txt`

- Now copy the file `simple.txt` and name this copy to be `pollfile.txt`

- Use `biaeventadaptor` to verify that this `FilePoll.props` configuration file now reads `pollfile.txt` and displays the content to the screen

- Now edit the configuration file to make the following changes:

  — Change the DO record type name to be: `MySpecialRecord`

  — Set the attribute names to be:

    ```
    AttrOne
    TheSecondAttribute
    AttrThree
    MyFourth
    ```

- Re-run the adaptor and verify that these changes have taken affect

Let's now experiment with the setting to save adaptor history:

- Edit the configuration file and set the `FilePollSource` component so that it saves its file position details

- Run the adaptor

  It runs just like it has in the past, displaying the contents of the file.

- Now run the adaptor again

  Notice that it outputs no more data records. This is because the previous adaptor run saved the fact that it had read a certain number of records from this file. When you run the adaptor again it sees that it has already read these records and so it looks to start from that point in the file. As the file hasn't grown since last time, no more records are output.

If you look in the `labs` directory there is a Java class called `FileAppend.class`.

- Run this class as follows:

  ```
  $ java FileAppend pollfile.txt
  ```

  This appends three more data records to this file.

- Now re-run your FPOLL adaptor...

  You should now see these new records.

- Re-run the adaptor...

  No records are output

So, by configuring the adaptor to save adaptor history it is able to know where it reached on its last run. This allows you to re-run an adaptor and bring in only the new data records since the previous run.

- Now edit the adaptor configuration file and change the input file name to be `simple.txt`

- Re-run the adaptor

  Notice that it outputs the entire contents of `simple.txt`. This is because the adaptor history information contains not only the file position but the name of the input file. Thus, if you reconfigure an adaptor (for whatever reason) to read from another file it starts afresh regardless of any previous adaptor history.

  Indeed, if you look at the log output of that last run, it tells you at the start of the run that it is ignoring any previous adaptor history.

Let's now experiment with the ability to continuously poll the input file:

- Edit the adaptor configuration file such that it:

  — Reads the `pollfile.txt file` as its input file

  — Stays polling this input file

  Make sure that you are still saving adaptor history.

- Run the adaptor

  The adaptor outputs the entire contents of the `pollfile.txt` file.

  The adaptor then appears to "hang". This is because it is continuing to poll the input file.

- Stop the adaptor

  Just kill the adaptor by hitting control-C in the command window.

- Re-run the adaptor

  Your adaptor starts up, sees the previous adaptor history, goes to the end of the input file, and then appears to "hang" again as it waits for more input data.

  Even though you killed the previous run of this adaptor, it still saved its file position!

  Remember, by default, when you ask to save the file position, this is saved every time the FilePollSource component reaches the end of file (EOF).

- Re-run your adaptor

- In a separate command window, go to the `labs` directory and run the command:

  ```
  $ java FileAppend pollfile.txt
  ```

  This appends three more data records to the pollFile.txt file.

  Notice the three new records are output to the screen by your adaptor.

- Kill your polling adaptor (using Control-C)
- Re-run your adaptor

  Notice that it again starts at the correct position in the input file and awaits more data.

So, if you are saving file position, each time the FilePollSource reaches the EOF it saves the current file position.

But what if your adaptor gets killed before it has had a chance to reach the EOF?

- Reconfigure your polling adaptor to read the input file: `large.txt`

  This file (`large.txt`) contains a few hundred records.

- Now re-run your adaptor...but...once it is displaying records to the screen, press Control-C to kill the adaptor

- Take a look at the last record to be displayed to the screen. In particular, take a note of the value shown in the `<AttrOne>` attribute

  Each record in the input file contains its record number within the `AttrOne` attribute (for easy testing in this lab :-)

- Now re-run your polling adaptor and see whether it starts at the record it reached in the previous run, or whether it starts afresh from the start of the file?

That's right! By default, when an adaptor is killed whilst processing records, it does not save it's file position to disc. But, you can enable this if you so desire...

- Configure your polling adaptor to save the file position after every record

- Re-run your adaptor...and when it starts displaying output records to the screen, kill the adaptor using Control-C

- Re-run your adaptor and see that it is now restarts from the next record in the input file!

FileRollSource

This source component can be used when your input data is being written across a sequence of files where the name of the file names can be predicted according to a particular roll policy...or...when you simply want to monitor a directory and read files as they appear in that directory.

In your `labs` directory, locate the configuration file: `FileRolling.props`

- Edit this configuration file and configure the FROLLING adaptor such that it:

  — Uses the `FileSelectByModifiedTime` file selection mechanism

  — Reads all input files from the directory: `lab_in`

- Create a subdirectory (within the `labs` directory) called `lab_in`
- Copy the file `log_2005-09-27.txt` into the `lab_in` directory
- Run the FROLLING adaptor and (hopefully) it processes the text file within the `lab_in` directory
- Re-run your adaptor

  Notice that the default action is to delete each file as it is successfully processed, hence there are no files to be processed.

  Notice also that the `FileRollSource` (using the `FileSelectByModifiedTime` file selection mechanism) waits until at least the first file appears before continuing.

- With your adaptor waiting for the next file, copy both the `log_2005-09-27.txt` and `next_roll_file` files from your `labs` directory into the `lab_in` subdirectory

  Both files are processed and each record displayed to the screen.

At the moment, your adaptor processes any file that appears in the lab_in directory. You may have also noticed that after processing the first file, there was a slight pause (for 1 second) before your adaptor moved onto the next file. And currently your adaptor exits when all the files in the directory have been processed.

- Edit your adaptor configuration file and set up the following:

    — Only process .txt and .stuff files

    — Do not pause at all between each input file

    — When the input directory is empty, poll for more files every three seconds

- Start your adaptor

- Now copy both the log_2005-09-27.txt and next_roll_file files from your labs directory into the lab_in subdirectory

    Notice that only the log file is processed, and the adaptor stays waiting for more files.

- In the labs directory, make three copies of the log_2005-09-27.txt file. It doesn't matter what the names are, just so long as they all end in .txt.

- Now copy all these .txt files into the lab_in subdirectory

    Notice that they are all processed (in the order of their last modified timestamp) and there is no delay when switching from one file to the next.

Let's now configure the adaptor to archive each input file after processing:

- Configure your adaptor to archive all input files to a directory called lab_archive

- Create a subdirectory (within the labs directory) called lab_archive

- Re-run your adaptor and then copy in the log_2005-09-27.txt file and the other file copies you made earlier

    You should see these files processed, and you should then find them archived in the lab_archive subdirectory.

Well done! You have reached the end of the lab.

Sockets

The standard `SocketSource` component supplied with openadaptor only allows a single connection. That is, once something has connected to a SocketSource (over an agreed port) no one else can use that port.

This is a limitation, so a new multi-threaded version of the SocketSource, called **SocketMTSource** (MT = Multi-Threaded), has been developed.

SocketMTSource

The `SocketMTSource` is configured in the same way as the standard `SocketSource`, with the following additional (optional) configuration properties:

- `MaxThreads`

  Lets you configure the maximum number of simultaneous client connections allowed to this `SocketMTSource`. The default is 100.

- `ThreadInactivityTimeoutMins`

  If any of the client socket connections are inactive for the specified period of time, their connection is closed and made available for other clients. Socket clients use the `SocketSink` to connect, and `SocketSink` automatically detects a lost connection and reconnects.

  This property allows the `SocketMTSource` to cope with clients that lose their connection through (for example) a networking or firewall issue where their connection is closed for whatever reason. The `SocketMTSource` is able to release the connection and make it available to new clients.

  The default time out value is 30 minutes.

- `ClientHostName<n>`

  The basic `SocketSource` allows you to use the `ClientHostName` property to restrict connections to be from a specified host name. The `SocketMTSource` allows you to configure a list of allowed client host names.

As with the standard `SocketSource` component, the `SocketMTSource` can be configured to work in one of two modes:

- `InitiateConnect = true`

  In this mode, the `SocketMTSource` starts up and then tries to open the configured port. It then behaves exactly as the original `SocketSource` component. That is, only a single connection.

- `InitiateConnect = false`

  In this mode, the `SocketMTSource` starts up and then waits for clients to connect to the configured port. As each client connects it spawns a new thread to handle that client connection, leaving the original port available for the next client connection.

  `SocketMTSource` allows up to the configured `MaxThreads` client connections.

Example Configurations

- To allow up to 1000 concurrent connections from any client, and time out inactive connections after one hour of inactivity:

```
A.C1.ClassName          = org.openadaptor.adaptor.standard.SocketMTSource

A.C1.Port            = 29200
A.C1.HostName        = localhost

A.C1.InitiateConnect = false

A.C1.MaxThreads                     = 1000
A.C1.ThreadInactivityTimeoutMins    = 60
```

- To restrict the clients that are allowed to connect, you can specify a series of host names from which to receive connections. This means that you only accept connections from these host names.

```
A.C1.ClassName       = org.openadaptor.adaptor.standard.SocketMTSource

A.C1.Port            = 29200
A.C1.HostName        = localhost

A.C1.InitiateConnect = false

A.C1.MaxThreads      = 1000

A.C1.ClientHostName1  = machineX.domain.com
A.C1.ClientHostName2  = another.somewhere.com.au
A.C1.ClientHostName3  = machineY.hp.com
```

- To restrict the number of concurrent clients to a low number:

```
A.C1.ClassName       = org.openadaptor.adaptor.standard.SocketMTSource

A.C1.Port            = 29200
A.C1.HostName        = localhost

A.C1.InitiateConnect = false

A.C1.MaxThreads      = 5
```

Databases

There is a small problem with the standard `PollingSQLSource` component that ships with openadaptor. As a result, OVBPI ships a replacement version of `PollingSQLSource` that you should use if you need to read data from a JDBC database.

The openadaptor utility class `JdbcConnectionParams` has also been extended to allow password encoding.

PollingSQLSource

The `PollingSQLSource` lets you configure an adaptor that reads records from a database. You configure the following main parts:

- The database connection details

- The SQL to select the records

- The SQL to be issued when the records have been processed successfully

- The SQL to be issued if an error occurred when processing the records

For an example of configuring and using the `PollingSQLSource` component, refer to Events From a Database on page 234

The replacement `PollingSQLSource` has these additional configuration properties over and above the standard version:

- `QuoteMultipleKeys` (optional)

  This configuration option was introduced to overcome a problem that occurred when specifying key values within an IN clause of the SELECT statement.

  This problem only occurs when the primary key being used in the IN clause, is of a STRING type.

  Suppose you want your SQL source to process all records where the key field is in a certain set of values. Suppose the key is a STRING field. Therefore the select statement must eventually expand to something like this:

```
SELECT * from table1
 WHERE TheUID IN ('100','200','300','400','500')
  ORDER BY GeneratedDate;
```

That is, each key value must appear as a valid STRING and individually quoted.

The PollingSQLSource lets you configure the SQL statement by providing a template. You provide an SQL statement to determine the set of key values. You then provide an SQL statement to process these key values.

For example:

```
A.C1.NextPrimaryKeySQL  = select TheUID from table1 where EventStatus='NEW'

A.C1.SelectSQL = select * from table1 where TheUID IN ('PK')
                                            order by GeneratedDate
```

PK is the default token to represent "primary key". The idea is that the NextPrimaryKeySQL statement is executed and PK then holds the result. Then the select SQL statement is expanded to contain this result so that the select can be issued to get the actual data records.

However, when the NextPrimaryKeySQL result is expanded within this SelectSQL string, the standard PollingSQLSource builds the command as follows:

```
select * from table1 where TheUID IN ('100,200,300,400,500')
    order by GeneratedDate
```

which is incorrect(!) as it has the entire list of values within one set of quotes.

So by using the QuoteMultipleKeys option you can tell the PollingSQLSource to quote each STRING key individually.

- BatchSize (optional - defaults to unlimited)

  By default, PollingSQLSource selects every record in the database that matches the configured criteria. In some cases this could be hundreds or thousands of records.

  The BatchSize option was added to provide a database generic way to limit the number of records retrieved with each poll of the data table. It is not necessarily the most efficient mechanism, however it does work.

  You specify a number greater than zero (0).

  Refer to Events From a Database on page 234 for further information.

As well as adding these new configuration options, the behavior of an existing setting has also been altered slightly:

- `UsingChainedTransactions`

  Although this is a SYBASE specific parameter, the standard `PollingSQLSource` set it in such a way that it affected every database type.

  For non-Sybase databases you no longer need to set this option.

JdbcConnectionParams

The `org.openadaptor.adaptor.util.JdbcConnectionParams` class is extended. The additional options are:

- `PasswordEncoding`

  Can be set to `true` or `false`. The default setting is `true`.

- `Retries`
  `RetryDelay`

  The `Retries` property configures the number of times an SQL select statement is retried in case of an error. Setting `Retries` to `-1` sets infinite retries. The default value is three(3).

  The `RetryDelay` property configures the delay, in milliseconds, between SQL retries. The default value is `1000` (one second).

An example database connection configuration might be as follows:

```
A.DB.JdbcDriver      = com.inet.tds.TdsDriver
A.DB.JdbcUrl         = jdbc:inetdae7://localhost:1433
A.DB.UserName        = ovbpiuser
A.DB.Password        = 1;3y17701q6x2b5a2l
A.DB.PasswordEncoding = true
A.DB.Database        = OvbpiSchema
```

where:

- The `PasswordEncoding` option is set to true and so the database password is specified as an encoded version of the password

  To see how to produce an encoded password refer to Password Encoding on page 152.

- The SQL error retries defaults to three retries, each with a one second delay inbetween.

Password Encoding

Having added the ability to specify encoded database passwords within openadaptor property files, it would be good to know how to produce an encoded password.

The encoding is produced by the class:

```
org.openadaptor.adaptor.util.Encoder
```

To run this class standalone, you do the following:

- Open a command window

- Run the command:

```
$ java -classpath OVBPI-install-dir\java\bia-event.jar
                org.openadaptor.adaptor.util.Encoder  password
```

(all on one command line)

where:

— You specify the full path of the bia_event.jar file as the classpath

— You supply the password in plain text

- The encoded version of the password is output to the command window display

For example:

```
$ java -classpath d:\ovbpi\java\bia-event.jar
          org.openadaptor.adaptor.util.Encoder myPassword
1;0p1a611u2d5z5q71411t5u1b1q581d2e
```

where the string 1;0p1a611u2d5z5q71411t5u1b1q581d2e is the encoded version of the test password myPassword.

Pipe Components

There are two new pipes to address the area of "data adornment":

- `AdornmentPipe`

    This pipe is for adorning data records with **static** data.

    The data to be added (adorned) to each DO record is contained within the adaptor configuration file.

- `IndirectionPipe`

    This pipe is for adorning data records with **dynamic** data.

    The data to be added (adorned) to each DO record is found through indirection - by performing configured database retrieval.

AdornmentPipe

The `AdornmentPipe` provides a simple mechanism for adorning known, constant data onto each data record as it passes through the pipe.

You can specify adornment on an individual record-type basis. You can specify a "default" adornment. You can specify that certain record types do not get adorned, but are simply passed through the pipe untouched.

The default action of the `AdornmentPipe` is "pass-through".

The `AdornmentPipe` class name is:
`org.openadaptor.adaptor.standard.AdornmentPipe`

The AdornmentPipe can be configured using the following properties...

To specify the record type for an adornment you have the property:

`AdornType<n>`

The values you can specify for this are:

- A valid record type name, such as `Any` or `MyRecordType`

- The value `__`**`DEFAULT`**

  This specifies that this adornment is to be applied to all data records except those that match any other adornment type criteria.

  This value is the word "default" preceded by two leading underscores.

You can specify as many AdornType<n> settings as you like, just so long as they start with AdornType**1**, and follow with AdornType**2**, AdornType**3** etc.

For any particular adornment type, you can specify what new attributes you would like added (adorned) to the data record. You configure these using the following:

```
AdornType<n>.AttName<m>=
AdornType<n>.AttValue<m>=
AdornType<n>.AttType<m>=
```

where:

- The name can be any valid attribute name

- The value can be any valid value, or the special tag `__`**`TIMESTAMP`**

  If you specify the value `__TIMESTAMP` then the `AdornmentPipe` sets this attribute to the current system date and time.

- The valid data type options for a `__TIMESTAMP` attribute are:

  String (default)
  Long
  Double

- The valid data type options for non timestamp attributes are:

  String (default)
  Integer
  Short
  Long
  Double
  Float

You can specify as many attributes as you like for each adornment type.

Example Configurations

The following sections provide example configurations for adornment pipes.

Adornment for a Specific Record Type

This example adorns records of type `NEWORDERS` with two new attributes. Records of any other type simply pass through untouched:

```
A.C3.AdornType1              = NEWORDERS

A.C3.AdornType1.AttName1     = EventGroup
A.C3.AdornType1.AttValue1    = Order Mgmt
A.C3.AdornType1.AttType1     = string

A.C3.AdornType1.AttName2     = GeneratedDate
A.C3.AdornType1.AttValue2    = __TIMESTAMP
A.C3.AdornType1.AttType2     = String
```

Configuring Specific, Default and Pass-Through Adornment

This example adorns all records of type `NEWORDERS` with two new attributes. Records of type `SPECIAL` are passed through untouched. All other record types are adorned with one new attribute:

```
A.C3.AdornType1              = NEWORDERS

A.C3.AdornType1.AttName1     = NewValue
A.C3.AdornType1.AttValue1    = 13.45
A.C3.AdornType1.AttType1     = double

A.C3.AdornType1.AttName2     = GeneratedDate
A.C3.AdornType1.AttValue2    = __TIMESTAMP

A.C3.AdornType2              = __DEFAULT

A.C3.AdornType2.AttName1     = Added_By_Default
A.C3.AdornType2.AttValue1    = WHATEVER

A.C3.AdornType3              = SPECIAL
```

IndirectionPipe

The `IndirectionPipe` allows people to be able to dynamically add data attributes to their data records.

For example, you might have a `FilePollSource` tailing a flat-file which contains a list of customer IDs. You are tailing this file because each time someone purchases something from your Web site their customer ID is written to this file. For each of these new records, you use the `IndirectionPipe` to dynamically access a database (or databases), using this customer ID, and add additional data values, such as customer name, customer type, business address, credit status, etc. You then send the nett result on to a socket sink - or wherever.

The `IndirectionPipe` lets you specify dynamic adornment on an individual record-type basis. You can specify a "default" adornment. You can also specify "pass through". Indeed, the default action of the `IndirectionPipe` is pass-through.

This pipe allows you to specify database connections (one or more, and to different databases) and SQL SELECT statements to be issued. These SQL select statements can specify attribute substitution, where the attribute value is found either from the current data record or from a previously issued SELECT statement within this lookup block.

The `IndirectionPipe` class name is:
`org.openadaptor.adaptor.standard.IndirectionPipe`

The `IndirectionPipe` is configured in two main parts.

Configure The Database Connection(s)

```
Connection1.JdbcDriver       =
           .JdbcUrl          =
           .Password         =
           .PasswordEncoding =
           .UserName         =
           .Database         =
           ...etc...
        :
ConnectionN.JdbcDriver       =
           .JdbcUrl          =
              :
```

You must specify at least **one** connection, and you can have as many as you need. These connections can be to different databases, across the network, or whatever is required for your needs.

The options available for the database connection are the same as for the `PollingSQLSource`. The `IndirectionPipe` uses the same generic openadaptor database connection mechanism.

Specify The Various Indirections (lookups) You Want Performed

```
Type1            =
Type1.Lookup1.SQL         =
              .Connection  =
              .Trim        =
              .TempVal     =
              .AttDelimiter=
         :
     .LookupM.SQL          =
              .Connection  =
              .Trim        =
              .TempVal     =
              .AttDelimiter=
     :
TypeN            =
TypeN.Lookup1.SQL          =
                   :
          :
```

where:

- Type<n>

  You specify the record type. All records of this type have this set of indirections performed on them.

  If you specify the value ___DEFAULT then this indirection is performed for all data records that do not match any other type that may be configured for this IndirectionPipe.

- Lookup<m>

  Each set of indirections is grouped by their lookup number. This is how you specify the order of your select statements within each record type block.

- SQL

  Here you specify a SELECT statement.

  To include values from the current data record, or previous lookups from within this lookup block, you include the attribute's name within "@@" delimiters.

  For example:

  ```
  select * from orders where orderno = @@field1@@
  ```

  This substitutes the value of the attribute field1 into that position within this select statement.

A select statement can only return **one** row of data. If it returns more than one row, the data from the first row is processed, and all following rows ignored.

If you need to issue multiple select statements then specify each one (in order) within separate `Lookup` blocks, starting with Lookup1 then Lookup2, etc. Each Lookup can refer to data attributes returned from the previous select(s) within this block of lookups.

Note that if you are substituting an attribute at the end of the SQL statement, you don't need the final @@'s. That is,

```
select * from orders where order_number = @@OrderNo@@
```

evaluates to the same final SQL statement as...

```
select * from orders where order_number = @@OrderNo
```

The select statements can be as complex as you require. It may be that your select chooses to join tables together etc. So long as your SQL expands at run time to a valid SQL select statement you can be as complex as you require.

- `Connection`

  Specifies the number of the (previously defined) database connection - starting from "1" - to be used for this select statement. This is how you specify which database you wish to use for this select statement.

- `Trim`

  This only applies to String values.

  If you specify Trim to be "true" then **all** String-type attributes returned by this select statement, have leading and trailing white spaces removed. This setting only applies to the return values from that particular SQL statement.

- `TempVal`

  If set to "true" then the return attribute(s) for this select statement do **not** go into the final output data record. They are placed into a temporary object so that they are available for further select statements (within this lookup block).

- `AttDelimiter`

  This lets you override the overall attribute delimiter. This allows you to use a different attribute delimiter for this one SQL statement.

The overall delimiter for attribute substitution within the select statements defaults to `@@`. If you want to alter this setting, you can specify the following configuration property:

`AttDelimiter`

This sets the overall attribute delimiter to be whatever value you have specified.

As mentioned above, this overall attribute delimiter can be overridden within individual select statements.

Example Configurations

Here are some example configurations showing the use of the
`IndirectionPipe`:

Indirection for a Specific Record Type

```
A.C3.Connection1.JdbcDriver= com.inet.tds.TdsDriver
A.C3.Connection1.JdbcUrl   = jdbc:inetdae7://localhost:1433
A.C3.Connection1.Password  = user
A.C3.Connection1.UserName   = password
A.C3.Connection1.Database   = mydatabase
A.C3.Connection1.TransactionIsolationLevel = TRANSACTION_READ_COMMITTED

A.C3.Type1                  = NEWORDERS

A.C3.Type1.Lookup1.SQL      = select customer_UID from orders
                                    where orderno = @@OrderNumber
A.C3.Type1.Lookup1.TempVal = true

A.C3.Type1.Lookup2.SQL      = select customerName, address from
                                  customers where
                                  customers_uid = '@@customer_UID@@'
```

where:

- Any data records that are not of type NEWORDERS simply pass through this `IndirectionPipe` untouched

- The first SQL select is issued to find the customer_UID, substituting the value of OrderNumber from the current data record

- As no Connection is specified, it defaults to using connection "1"

- TempVal is set to true, so the customer_UID that is returned from this select statement is held in a temporary object and does not appear in the final output data record

- The next select statement is issued, using the customer_UID from the previous select, and this returns the customerName and address attributes

- As there is no `TempVal` specified for this lookup (the default is false) these two attributes are placed into the output data record

▶ openadaptor is **case-sensitive**. So when you specify that the first select statement returns `customer_UID`, you must refer to it in exactly the same way in the next select statement. Also, when you specify that the second SQL statement returns `customerName`, that is the way the attribute name appears in the output record.

Specifying Default Indirection

```
A.C3.AttDelimiter              = $%$

A.C3.Type1                     = NEWORDERS

A.C3.Type1.Lookup1.SQL         = select customer_UID from orders
                                     where orderno =
                                            @@!!@@OrderNumber
A.C3.Type1.Lookup1.TempVal     = true
A.C3.Type1.Lookup1.AttDelimiter = @@!!@@

A.C3.Type1.Lookup2.SQL         = select customerName, aDDress
                                     from customers where
                                         customers_uid =
                                           '$%$customer_UID$%$'

A.C3.Type2                     = __DEFAULT

A.C3.Type2.Lookup1.SQL         = select * from orders where
                                            order_id='abc'
A.C3.Type2.Lookup1.Connection  = 3
A.C3.Type2.Lookup1.Trim        = true
```

where:

- Records of type `NEWORDERS` have two selects issued and the `customerName` and `aDDress` attributes are added to the output record

- All other record types have a single SQL statement issued where it adds the columns from the `orders` table to the output record

  The `__DEFAULT` tag matches all record types not otherwise specified in this configuration. In other words, in the above example, any record type other than `NEWORDERS`

- You need to be careful when using the `select *` construct if you then want to use one/some of these return values. Some databases return the attribute with their names capitalized, whereas some return them in mixed-case. This is important because openadaptor is case-sensitive

- In the above example you set the overall attribute delimiter to be the string `$%$`

  You would expect people to leave the overall delimiter as @@, however the ability to override this for all SQL statements is there.

Specifying Pass-Through

```
A.C3.Type1                    = __DEFAULT

A.C3.Type1.Lookup1.SQL        = select * from orders
                                      where order_id='abc'
A.C3.Type1.Lookup1.Connection = 3
A.C3.Type1.Lookup1.Trim       = true

A.C3.Type2                    = SPECIAL
```

In this example, records of type `SPECIAL` are passed through untouched. All other record types have the columns from the `orders` table added to their output record.

Lab - Using Adornment and Indirection

The purpose of this lab is to gain some practice configuring static and dynamic adornment.

Adornment

The `AdornmentPipe` allows you to statically configure the addition of attributes (sometimes called fields) to data records as they come through the pipe.

Although the `AdornmentPipe` allows you to configure different adornments depending on the record type, people often only need a default adornment. That is, every record coming through the pipe simply needs some additional attributes added on.

Let's configure a simple adornment where you do exactly this - every record that comes through the pipe has a certain set of attributes and values added to them.

- In the `labs` directory, edit the file `FileAdornPrint.props`

- Your mission is to configure the following:

  — Configure an extra component called `AP` which is the `AdornmentPipe`

  — The class to use is:
    `org.openadaptor.adaptor.standard.AdornmentPipe`

  — Link the `AP` component to be after the source and before the sink

  — Configure `AP` to add the following attributes to every data records:

    ```
    Name            Value               DataType
    ----            -----               --------
    EventName       MyEvent             String
    TheDate         current time        Long
    ExtraCode       123.45              Double
    ```

Once you have the adaptor working:

- Modify your adaptor configuration such that the existing data record field called `Address` is always set to the string `NOT-AVAILABLE`

  This just shows you that, by using the `AdornmentPipe`, you can override existing values within the data record if you like. The `AdornmentPipe` is not just for "adding" extra data attributes.

Indirection

The `IndirectionPipe` allows you to dynamically configure the addition of attributes (sometimes called fields) to data records as they come through the pipe, by carrying out database lookups (indirections).

Again, the `IndirectionPipe` is able to carry out indirection on an individual record-type basis, however many people simply wish to configure default indirection. They wish to configure that a certain chain of database indirection occurs for all records as they pass through the pipe.

Before you configure any indirection you need a database (so you can do lookups against it).

For this lab you can use either MSSQL or Oracle - depending on what is available to you at this time.

Let's load up the database with some sample values:

- In the `labs` directory, locate the files:

  ```
  dbsetup_orders.mssql
  dbsetup_orders.oracle
  ```

  Choose the one that is appropriate, and run that file against your database.

  For example, if you are working with an Oracle installation, you could use SQLPLUS and execute the script `dbsetup_orders.oracle` or cut and paste it into something like the SQL-Worksheet.

- Make sure that the new tables (`TestOrders` and `TestCustomers`) are now in your database and loaded correctly

- Also make sure that these changes have been committed for all users to see

The two tables that are now loaded into your database are as follows:

- The `TestOrders` table contains an entry for each order on your system. The key is the `ORDERNO` field

- Each order contains details such as the amount of the order, the date it was placed, etc. And it includes the `CUSTOMERID` which relates to the customer that placed this order

- The `TestCustomers` table contains an entry for each customer on your system. The key is the `CUSTOMERID` field

- Each customer entry contains details such as the customer's address, status etc.

For this lab, your adaptor reads the file `orders.txt` which is found in the `labs` directory. This file consists of records that contain only 1 attribute - the order number.

- You mission is to configure your adaptor to read this orders file as its input, and for each record the adaptor must do one (or more) database lookup(s) to add to each record the following attributes:

```
The order amount
The order priority
The customer ID placing this order
That customer's name
That customer's address
That customer's region
```

The basic steps for achieving this are as follows:

- In the `labs` directory, use the starting template file `FileIndirectionPrint.props`

- Configure your database connection

  There are empty entries already in the adaptor configuration file. You just need to assign the correct values for these.

- Configure a default indirection

  This uses the order number from the data record to look up the customerID, amount and priority from the TestOrders table. It then uses the customerID to lookup the customer name, address and region from the TestCustomers table.

- Don't forget to configure the actual pipe into the adaptor linkage!

> **Well done! You have reached the end of the lab.**

OVBPI Specific Components

Some openadaptor components have been developed that are quite specific to OVBPI. These being:

- `BIAEngineSink`

  This sink accepts the data records, convert them into the appropriate format for the OVBPI Engine and then transmit them into the Engine.

- `BIAEventStoreSink` and `BIAEventStoreSource`

  The "event store" is the name of a data table that can be used to hold every data record (event) as it is being sent towards the OVBPI Engine. That is, rather than sending data records directly to the `BIAEngineSink`, you send all your data into the event store - using the `BIAEventStoreSink`. You then have a separate adaptor which runs the `BIAEventStoreSource` to read the events out of the data table and sends them to a `BIAEngineSink`.

  The use of an event store is discussed in more detail in the section Event Store Architecture on page 199.

Let's consider how to configure each of these OVBPI specific components.

BIAEngineSink

The `BIAEngineSink` is the connection between the OVBPI Event Handler and the OVBPI Engine.

The `BIAEngineSink` **must** run on the same server as the OVBPI Engine.

The `BIAEngineSink` is responsible for receiving data records and turning them into OVBPI Engine events. Having verified this event against the run-time event repository (see The Run-Time Event Repository on page 218) the `BIAEngineSink` then transmits the event to the OVBPI Engine.

To configure the BIAEngineSink into an adaptor you simply need to link in a new component and specify its class name to be:

    org.openadaptor.adaptor.bia.BIAEngineSink

All other properties default to the OVBPI Engine defaults and therefore it should all work.

Should you need to alter the configuration for the BIAEngineSink, you have the following option available to you:

```
BIAEngineTransmitter =
```

where:

- BIAEngineTransmitter

  Specifies the actual transmitter to use when sending the event to the OVBPI Engine.

  Defaults to org.openadaptor.adaptor.bia.BIAEngineRMITransmitter

RMI Transmitter Configuration

The RMI transmitter class:

```
org.openadaptor.adaptor.bia.BIAEngineRMITransmitter
```

is the default transmitter. Indeed it is the only transmitter that is currently supported.

By default you do not have to set any configuration options.

The RMI transmitter has the following configurable options:

```
RmiEventConsumerService =
RmiMaxRetries          =
RmiNamingRegistryHost  =
RmiNamingRegistryPort  =
RmiRetryDelay          =
```

where:

- RmiEventConsumerService

  This is the service name under which the OVBPI Engine has bound its RMI RemoteEventConsumer to the RMI Naming service.

  The default is EventManagerService.

- RmiNamingRegistryHost

  This sets the host name on which the RMI Naming service is running, and on which the sink looks up the RMI RemoteEventConsumer.

  The default is localhost.

- RmiNamingRegistryPort

  This sets the port number on which the RMI Naming service is listening, and on which the sink looks up the RMI RemoteEventConsumer.

  The default is `44000`.

- RmiMaxRetries

  This sets the maximum number of times the sink should attempt to retry sending an Event to the RMI RemoteEventConsumer if the latter throws an exception.

  Defaults to `5`.

- RmiRetryDelay

  This sets the delay, in milliseconds, between retries if the sink should attempt to retry sending an Event to the RMI RemoteEventConsumer.

  Defaults to `30000` (half a minute).

Example Configurations

- A default configuration:

```
A.C2.ClassName       = org.openadaptor.adaptor.bia.BIAEngineSink
```

  This defaults everything.

- A configuration that chooses to communicate to the OVBPI Engine on a different port:

```
A.C2.ClassName       = org.openadaptor.adaptor.bia.BIAEngineSink
A.C2.RmiNamingRegistryPort = 1234
```

- To configure different retry settings:

```
A.C2.ClassName       = org.openadaptor.adaptor.bia.BIAEngineSink
A.C2.RmiRetryDelay   = 5000
A.C2.RmiMaxRetries   = 10
```

BIAEventStoreSink

The `BIAEventStoreSink` puts records into the event store data table.

The `BIAEventStoreSink` is based upon the standard openadaptor `PollingSQLSink` but is specific to handling the OVBPI event store data table and its specific format.

The event store is a data table that has the following columns:

- `GUID`

  A unique ID.

- `EVENT_GROUP`

  The event group.

- `EVENT_NAME`

  The event name.

- `DOXML`

  The actual openadaptor message - the array of DataObjects, stored in as XML.

- `STATUS`

  A status, either UNSENT or SENT.

- `TIME_RX`

  Timestamp written on receipt of event in the event store.

- `TIME_TX`

  Timestamp written on retrieval of event from the event store.

This event store data table is set up automatically when you install OVBPI, and it is called **EVENT_STORE**.

Should you wish to see the actual SQL for creating this table, you find the SQL script files in the directory: *OVBPI-install-dir*\misc\bia

To configure the `BIAEventStoreSink`, you specify the same configuration options as you would if configuring the standard `PollingSQLSink` component:

```
JdbcDriver              =  (required)
JdbcUrl                 =  (required)
UserName                =  (required)
Password                =  (required)
PasswordEncoding        =  (optional)
Retries                 =  (optional)
RetryDelay              =  (optional)
Database                =  (optional - depends on DB vendor)
TransactionIsolationLevel = (optional)
ExitOnError             =  (optional - defaults to true)
```

where:

- Valid values for `TransactionIsolationLevel` are:

  ```
  TRANSACTION_SERIALIZABLE (default),
  TRANSACTION_NONE
  TRANSACTION_READ_UNCOMMITTED
  TRANSACTION_READ_COMMITTED
  TRANSACTION_REPEATABLE_READ
  ```

  These settings are all described in the standard JDBC documentation if you want to understand more about them.

There is then one further **required** configuration property:

```
Table     = EVENT_STORE
```

You need to specify the actual name of the data table that holds the event store.

Example Configuration

```
A.C2.ClassName  = org.openadaptor.adaptor.bia.BIAEventStoreSink

A.C2.Table      = EVENT_STORE

A.C2.JdbcDriver                 = com.inet.tds.TdsDriver
A.C2.JdbcUrl                    = jdbc:inetdae7://localhost:1433
A.C2.UserName                   = ovbpiuser
A.C2.Password                   = ovbpi
A.C2.Database                   = OvbpiSchema
A.C2.TransactionIsolationLevel = TRANSACTION_READ_COMMITTED
```

BIAEventStoreSource

The `BIAEventStoreSource` reads events from the Event Store data table.

The `BIAEventStoreSource` is based on the standard openadaptor `PollingSQLSource` component but is specific to handling the OVBPI event store data table and its specific format.

For a description of the format of the event store data table refer to BIAEventStoreSink on page 171.

The `BIAEventStoreSource` requires the following configuration property:

```
Table       = EVENT_STORE
```

You need to specify the actual name of the data table that holds the event store. The default event store is created at OVBPI install time and called `EVENT_STORE`.

The `BIAEventStoreSource` then supports the following optional properties (over and above the normal `PollingSQLSource`):

```
PollSource          =
PollPeriod          =
BatchSize           =
Rdbms               =
DeleteHandledEvents =
```

where:

- `PollSource`

  Specifies whether the `BIAEventStoreSource` polls the event store continuously (as configured using the `PollPeriod`).

  Defaults to `true`.

- `PollPeriod`

  Specifies the time in Milliseconds between each poll of the event store data table.

  Defaults to `1000` milliseconds.

- BatchSize

  Specifies the number of events read from the event store with each poll.

  Defaults to one(1).

  If you wish to configure the BIAEventStoreSource to retrieve all events in a single poll, you can set the BatchSize to zero(0).

- Rdbms

  This is an optional property.

  Valid values are: mssql or oracle.

  This property is here to help with the BatchSize property. If you do not specify the Rdbms property, the BIAEventStoreSource uses the JDBC driver "size limiting" methods to limit the number of returned records to match that specified in the batch size. However, for some drivers, these JDBC methods do not always work. So, by specifying the Rdbms property, the BIAEventStoreSource is able to build SQL select statements specific to that database type.

  You really should use the Rdbms property.

- DeleteHandledEvents

  Specifies what the BIAEventStoreSource is to do with the events after processing them. By default the events are deleted from the event store after they have been processed. By setting this option to false you can tell the BIAEventStoreSource to leave the events marked as SENT.. This would then allow you to reply the events if you needed to.

  Defaults to true

The `BIAEventStoreSource` also supports the following connection properties (in the same way as the standard `PollingSQLSource` component):

```
JdbcDriver               = (required)
JdbcUrl                  = (required)
UserName                 = (required)
Password                 = (required)
PasswordEncoding         = (optional)
Retries                  = (optional)
RetryDelay               = (optional)
Database                 = (optional - depends on DB vendor)
TransactionIsolationLevel = (optional)
```

where:

- Valid values for `TransactionIsolationLevel` are:

  ```
  TRANSACTION_SERIALIZABLE (default),
  TRANSACTION_NONE
  TRANSACTION_READ_UNCOMMITTED
  TRANSACTION_READ_COMMITTED
  TRANSACTION_REPEATABLE_READ
  ```

  These settings are all described in the standard JDBC documentation if you want to understand more about them.

Example Configuration

```
A.C1.ClassName = org.openadaptor.adaptor.bia.BIAEventStoreSource

A.C1.JdbcDriver                = com.inet.tds.TdsDriver
A.C1.JdbcUrl                   = jdbc:inetdae7://localhost:1433
A.C1.UserName                  = ovbpiuser
A.C1.Password                  = ovbpi
A.C1.Database                  = OvbpiSchema
A.C1.TransactionIsolationLevel = TRANSACTION_READ_COMMITTED

A.C1.Table             = EVENT_STORE

A.C1.PollPeriod        = 5000

A.C1.DeleteHandledEvents = true

A.C1.BatchSize         = 20
A.C1.Rdbms             = mssql
```

AFEditor

If you run AFEditor and you are unable to see any of the OVBPI enhancements (such as the `AdornmentPipe`, `FilePollSource`, etc.) then it means that you do not have the correct classpath.

The OVBPI jar files contain all the necessary configuration (.ini) files such that AFEditor is able to configure and edit all the OVBPI additional and modified components, however these need to be at the start of your classpath.

biaeventafeditor Script

To make it easy to set the correct classpath for AFEditor, OVBPI provides the script `biaeventafeditor.bat`. This script sets up the correct classpath such that when you run AFEditor you see all the OVBPI enhancements as well as the rest of the standard openadaptor classes.

You must always use this script to run AFEditor, otherwise you will not be able to configure any of the OVBPI additional or enhanced components.

The `biaeventafeditor` script takes no parameters. You simply run it and it starts up AFEditor.

Hospitals

openadaptor provides a set of message hospital implementations for a number of databases such as Sybase, MSSQL, Oracle etc. However, the version 1_6_5 hospital implementations for MSSQL and Oracle have problems and simply do not work! So part of the effort during the OVBPI implementation was to fix the hospital functionality so that it worked for both MSSQL and Oracle.

The scripts for setting up the hospital are found under the `OVBPI-install-dir`\misc\bia directory.

The files are:

- `HospitalSchema_script.sql`

  This creates the basic data tables and views that make up a hospital.

- `HospitalProcs_script.sql`

  This creates the stored functions and procedures that are used when admitting and discharging patients to/from the hospital.

Default Hospital Installation

As part of the standard OVBPI installation, a hospital is configured for you automatically.

When you choose your database, the install scripts configures the necessary hospital tables and procedures, thus your hospital is set up and ready to be used. So you do not need to run any hospital set up scripts.

The hospital is configured into the OVBPI database schema that you named when you installed OVBPI. The default name for this schema is ovbpischema.

This schema contains all of the hospital tables (`DBusMH_*`), and stored procedures, all configured and ready for patients.

A user is pre-configured to access the Hospital Administration Tool (HAT). This user is the OVBPI user that you configured when you installed OVBPI. The default user name is `ovbpiuser`.

There is a default HAT properties file pre-configured to allow access to the hospital. This HAT properties file is called:

> `OVBPI-INSTALL-DIR`\data\conf\bia\BIAEventHAT.props

biaeventadaptor Script

For your adaptor to have access to the working MSSQL or Oracle hospital implementations you must ensure that your adaptor is run using the `biaeventadaptor.bat` script. This script sets up the necessary classpath such that these classes are available.

HospitalSource

The default `HospitalSource` from openadaptor reads discharged patients from the hospital and processes them. It then leaves the patient in the hospital. The OVBPI version of the `HospitalSource` has been enhanced so that you can specify whether you want patients deleted from the hospital after they have been successfully discharged and processed.

To delete patients after they have been processed, you configure the HospitalSource property:

```
A.HS.DeleteOnDischarge = true
```

If this property is not present, it defaults to false - and patients are left in the hospital.

So, an example hospital source configuration might be:

```
A.Component<n>.Name    = HS

A.HS.ClassName         = org.openadaptor.adaptor.hospital.HospitalSource
A.HS.HospitalName      = H1
A.HS.ProcessPatients   = false
A.HS.PollPeriod        = 10000
A.HS.QueryAppName      = My_Application
A.HS.QuerySubject      = My_Subject
A.HS.DeleteOnDischarge = true
```

biaeventhat Script

For you to be able to run the Hospital Administration Tool (HAT) against MSSQL or Oracle you must use the script `biaeventhat.bat` as supplied with OVBPI. This script sets up the necessary classpath such that these classes are available.

By default, if you run this script with no parameters it defaults to the hospital that is set up when you installed OVBPI.

If, however, you wish to point this script at another hospital then the script does take one parameter:

- The name of a HAT properties file

  This must configure your hospital name and database connection details.

  The `biaeventhat` script looks for this file in your local directory. If the file cannot be found, the script then looks for this file in the `data\conf\bia` directory.

Most people simply run this script with no parameters and let it default to the hospital configured at OVBPI install time.

For example:

```
$ biaeventhat.bat
```

HAT Screens

As part of the work to fix the HAT to run on MSSQL and Oracle, some minor changes were made to the interface:

The main differences are:

- The big "hat" icon has been replaced with an "hp" icon

  This just provides a visual cue to show that you are running the correctly updated version of the HAT.

- When you logon to the HAT the password field is not displayed in plain text

- The patients are listed showing their Patient ID followed by the admission time stamp

Here is a screen shot of the main HAT screen to show you how it now looks:
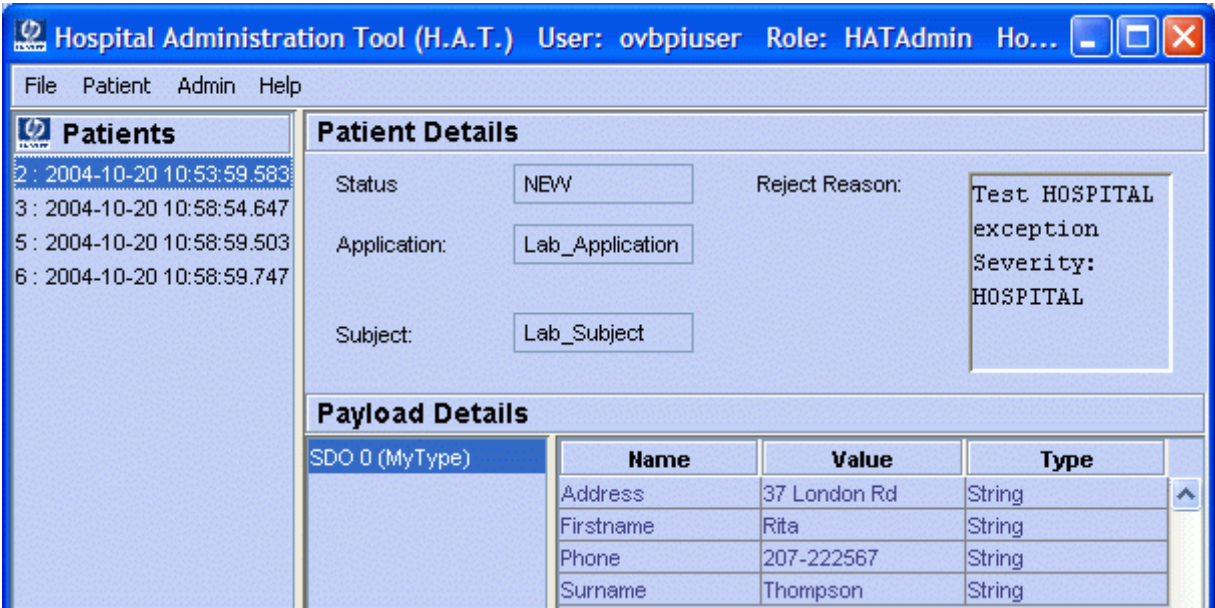
where:

- The list of patients on the left-hand side of the screen lists their patient ID, followed by a colon (:), followed by the time this patient was admitted to the hospital

Lab - Configuring an Adaptor to Use a Hospital

The purpose of this lab is to give you experience configuring an adaptor to make use of a message hospital.

For this lab you use the openadaptor `ExceptionPipe` class as this is an easy way to test out an adaptor's behavior when exceptions occur.

Configure the Actual Hospital (Database)

When OVBPI installs on your system, it installs the necessary tables and stored procedures required to run a message hospital.

Let's check that these are indeed set up:

- Connect to your database (MSSQL or Oracle) as a database administrator

- Drill into the `ovbpischema` database

- Locate the table: `DBusMH_Patient`

  This is the table that is used to hold all patient records.

- Locate the stored procedure: `DBusMH_AddPatient`

  This is just one of the stored procedures used when an adaptor is talking to the hospital. This procedure handles the admission of a new patient.

If this table and stored procedure do not exist then it is possible that you have connected to the wrong database, or that the OVBPI installation has installed them into a different database. If you feel you need to manually install these into your database then please refer back to the section Message Hospitals on page 82 for details of how to do this.

Assuming that the correct table and stored procedures are in place, let's configure an adaptor, get it working and then add in the hospital connection.

Running an Adaptor that Throws an Exception

- If you look in your `labs` directory, locate the file: `ExceptionAdaptor.props`

  This configures a simple file-to-screen adaptor that reads the input file `simple.txt` and outputs each record to the screen...however...it has an exception pipe configured such that it throws an exception on every second record. That is, it passes the first record, throws an exception on the next, passes the next, throws an exception on the next, etc.

- Have a look at this `ExceptionAdaptor.props` file and make sure you understand what it is doing

- Now run the adaptor and see what happens:

  `$ biaeventadaptor -stdout ExceptionAdaptor.props Except`

  You should see the first record printed to the screen and then an exception is thrown. And because you do not yet have a hospital connection configured...the adaptor terminates!

So, now to configure a hospital.

Writing Out to a Hospital

First let's configure enough details such that the adaptor is able to write error messages out to the hospital. For this you need to configure the following two things:

- A hospital connection - and give it a name
- A `HospitalPipe` that writes errors to this named hospital connection

The Hospital Connection

You may need to refer to the section Message Hospitals on page 82 for reference, but let's go ahead now and configure the adaptor's connection to the hospital:

- Edit the `ExceptionAdaptor.props` file

- Add in one of the following lines - depending on whether your hospital is in an Oracle or MSSQL database:

```
Except.H1.ClassName = org.openadaptor.adaptor.jdbc.oracle.HospitalOracleImpl
```

...or...

```
Except.H1.ClassName = org.openadaptor.adaptor.jdbc.mssql.HospitalMSSqlImpl
```

This declares that you are connecting to a hospital using this class, and that the hospital connection is now referred to (within this adaptor) as `H1`.

- You now simply need to configure the connections details for this hospital called `H1`. So you need to configure the following set of parameters:

```
Except.H1.JdbcUrl    =
Except.H1.JdbcDriver =
Except.H1.Database   =
Except.H1.UserName   =
Except.H1.Password   =
```

Note that if you are using an Oracle database then you do not need to specify the `Database` option.

Now that you have a hospital connection (`H1`) configured let's configure a hospital pipe to make use of it...

The HospitalPipe

- Still within the `ExceptionAdaptor.props` file, you need to declare a new component. Let's declare a component with the name `HP` - which stands for Hospital Pipe:

```
Except.Component<n>.Name  = HP
```

where `<n>` is replaced with the next component number.

- Alter the linkage of the adaptor components such that the file source passes messages to this new hospital pipe, then this hospital pipe passes them onto the exception pipe, then the exception pipe passes them onto the file sink

- Now declare the actual class name for the HospitalPipe:

```
Except.HP.ClassName = org.openadaptor.adaptor.hospital.HospitalPipe
```

- You now need to specify which hospital this `HospitalPipe` writes out to, and declare the "Application" and "Subject" headings under which all records are to be stored:

```
Except.HP.HospitalName       = H1
Except.HP.DefaultApplication = Lab_Application
Except.HP.DefaultSubject     = Lab_Subject
```

This declares that the component `HP` writes all error messages out to the hospital `H1`.

And all records are written with the headings: `Lab_Application/ Lab_Subject`

Once you have made all these configuration changes your adaptor should be ready to run - saving exceptions out to your hospital...

- Run the `Except` adaptor

```
$ biaeventadaptor -stdout ExceptionAdaptor.props Except
```

This time you should see three records output to the screen, and two sets of warnings saying that messages had been written to the hospital.

If you did not see this behavior then you need to go back and check your adaptor configuration - especially the hospital pipe and the hospital connection details.

Great! You have now configured your adaptor to save all error messages out to a hospital, allowing your adaptor to continue processing after any errors.

Let's now try and see these messages that have been saved to the hospital...

Hospital Administration Tool (HAT)

To run the hospital administration tool (HAT) you use the supplied script `biaeventhat.bat`

- You run the HAT, as follows:

```
$ biaeventhat.bat
```

The logon screen is pre-configured ready to log on as the ovbpiuser.

- Press `OK` to log on

- When the main HAT screen appears you should see your two new hospital patients

Let's go ahead and edit one of the attribute values.

You first need to allow yourself the ability to edit attributes within this particular record type:

- In the payload details, make a note of the data object type (This is shown in brackets next to the text `SDO 0`)

    In this lab it should be `MyType`

- Now choose the menu:

    `Admin->Editable Attributes`

- Set the DOType to be `MyType`

- Assign the HAT User Role to be `HATAdmin`

- Set the editable attributes column to `EDIT_ALL_ATTRIBUTES` and press carriage return

- Press the `Save` button ... and then press `OK`

- Now press the `Close` button - and return to the main HAT screen

You are now able to edit attributes for these `MyType` patients.

To activate this edit capability you might need to click on the next patient. The new edit capabilities do not always apply until you have moved on to a new patient and then moved back :-)

Let's go ahead and change the patient's `Surname` attribute value for the patient living at `39 New Street`:

- Click through the patients until you find the one you wish to modify

- Double-click on the `Surname` attribute value (currently containing the value: `Cranks`)

    The value field becomes active and you can type in the new surname value.

- Type in the value: `NewSurname`

- Press `carriage return` to have this change accepted by the HAT

Now to really write this change to the hospital - you need to move to another patient:

• Click on the other patient and the HAT prompts you to make your changes permanent...press `Yes` and then `OK`

Great! You have edited a patient. The HAT also let's you add attributes, delete attributes, etc..

Now let's discharge this patient:

• Select the patient that you just modified

• Select the menu:

    `Patient->Discharge`

• Then press `OK`

    This patient now shows as being in the state `DISCHARGE_WAIT`. That is, the patient is marked for discharge...is waiting to be discharged...but it is never discharged until you provide a hospital source that is able to handle this discharged patient...and any more that may be discharged in the future.

A HospitalSource

You could configure a completely separate adaptor that just contains the `HospitalSource`, however, for this lab, let's configure the hospital source within your current `Except` adaptor.

- Edit the file `ExceptionAdaptor.props`

- Configure a new component called `HS`:

  `Except.Component<n>.Name = HS`

  where you replace <n> with the next component number.

- You need to add to the current adaptor component linkage section, however, you do not insert the `HS` component into the current series of components - instead you simply define that this new `HS` component feeds its data to the File Sink (`C2`) component. That is:

  `Except.HS.LinkTo1 = C2`

  This means that you now have the following set up within your adaptor components:
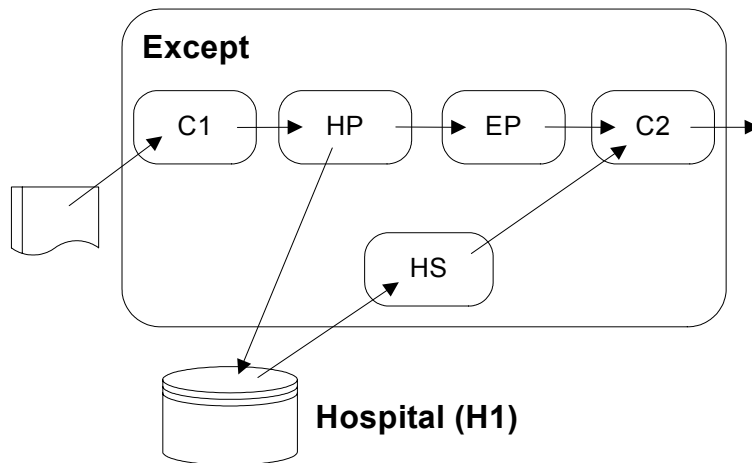
- You now need to configure this `HospitalSource` (`HS`) by specifying the class name, the hospital to which it connects, and the connection details. The section you need to complete is as follows:

```
Except.HS.ClassName          = org.openadaptor.adaptor.hospital.HospitalSource
Except.HS.HospitalName       = H1
Except.HS.ProcessPatients    = false
Except.HS.QueryAppName       =
Except.HS.QuerySubject       =
Except.HS.DeleteOnDischarge  = true
```

Make sure that you specify the correct values for `QueryAppName` and `QuerySubject`. These values must match the values which were used by the `ExceptionPipe`. That is, you want this `HospitalSource` to pull out only records that were written in by your `HospitalPipe`.

- Now re-run your `Except` adaptor

  You should see it process the usual records. The `ExceptionPipe` still causes some records to be hospitalized. And you should also see the record that you discharged earlier from the hospital.

- Notice that this time, when the `Except` adaptor has finished processing the input file, it stays active. It appears to "hang". If you look at the adaptor log output you see that it still has a thread running.

  This thread is the `HospitalSource`. This thread is still polling the Hospital for further patients incase they are discharged.

- Leave the `Except` adaptor running

- In a separate command window, run the HAT and discharge another patient

  You should see that the `Except` adaptor picks up this patient, processes it, and then deletes it from the hospital.

That's it! You have now:

- Configured an adaptor to send any errors to the hospital
- Configured and used the HAT to examine and discharge patients
- Configured a HospitalSource to process discharged patients

In this example, the HospitalSource feeds discharged patients straight to the sink component of the adaptor. Obviously, if your adaptor had a range of data formatting pipes, then your HospitalSource would feed discharged patients into those pipes and not by-pass them. Refer to the section Hospital Source on page 95 for more details on this.

Well done! You have reached the end of the lab.

Example Configurations

As well as the `labs` and `sols` directories provided with this training guide, there is also a set of example configuration files to be found under the directory:

`examples\bia\openadaptorExamples\cookbook`

This `cookbook` directory holds examples configurations for most of the components and is well worth a look. Hopefully the commented configuration files will help you.

The directory is called `cookbook` as that is the name that openadaptor use for their examples directory.

4 Event Handler Architecture

This chapter looks at various ways to configure your adaptors to feed data events into the OVBPI Business Impact Engine. This chapter then proposes some suggested architectures that provide the most straightforward and manageable network for your adaptors, allowing for easy expansion as and when adaptors are added to or removed from the system.

Introduction

With your knowledge of openadaptor you can now easily understand that to feed events into the OVBPI Business Impact Engine, you could simply configure and run an adaptor as follows:

Figure 20 A Simple Engine Adaptor
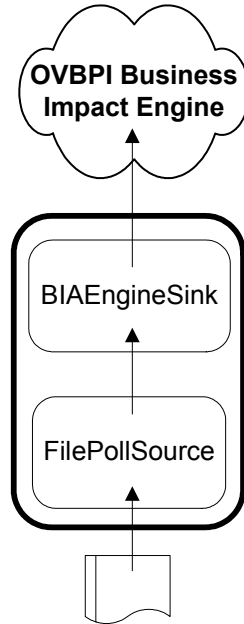
where:

- Your adaptor consists (in the example) of a `FilePollSource` reading from a file

- All records are passed to the `BIAEngineSink` where they are then passed into the OVBPI Business Impact Engine

There is absolutely nothing wrong with setting up such an adaptor. It works just fine.

What happens when you need to add a second input source?

You could configure a completely separate adaptor with this new input source feeding to its own BIAEngineSink, feeding up to the OVBPI Business Impact Engine.

Figure 21 Multiple BIAEngineSinks
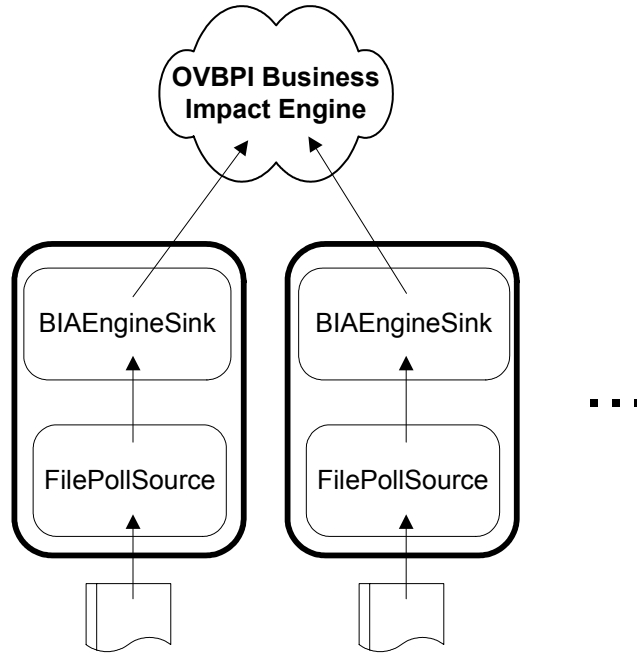
However, this would mean that when you come to adding a third, or fourth, adaptor you would have three, or four, BIAEngineSinks running and you don't really want to be configuring lots of BIAEngineSinks running on the system. The performance would degrade. You should only have one BIAEngineSink running at any one time.

Another alternative is to reconfigure the adaptor to have two input sources, both feeding up to the one `BIAEngineSink`, as follows:

Figure 22 Multiple Sources To A Single BIAEngineSink
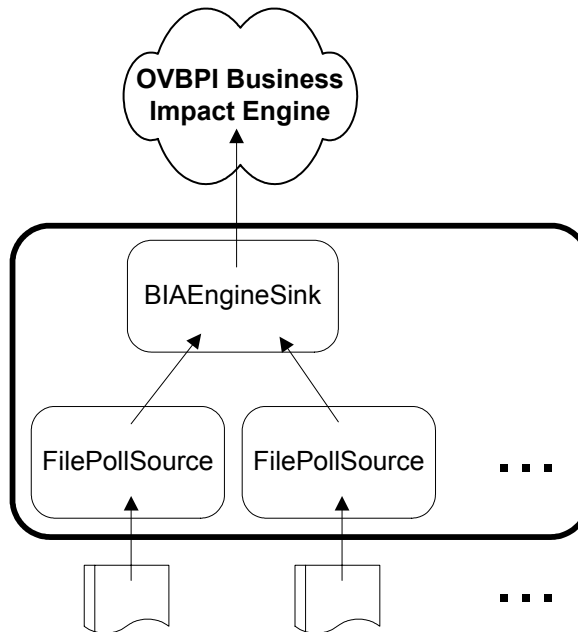
The problem with this approach is that for every new input source you must take down the current adaptor (which is handling all the current input source feeds), reconfigure it, and then start it all up again.

Another possible concern with all of the above approaches is that all your input sources would need to be available to the server on which this adaptor is running. Depending on your network topology, this may not be possible.

There is no simply answer to how you set up your adaptors. The simple adaptor feeding data events into an Engine sink works, however it is not going to provide the best starting point toward a flexible and manageable adaptor network.

So, what follows are some suggested architectures for setting up your adaptors. The only problem, however, is that each customer installation may have different requirements for their adaptor network. Hence the remainder of this chapter discusses various possible solutions, considering their pros and cons.

Socket Based Architecture

The socket based architecture looks something like this:

Figure 23 Socket Based Event Handler
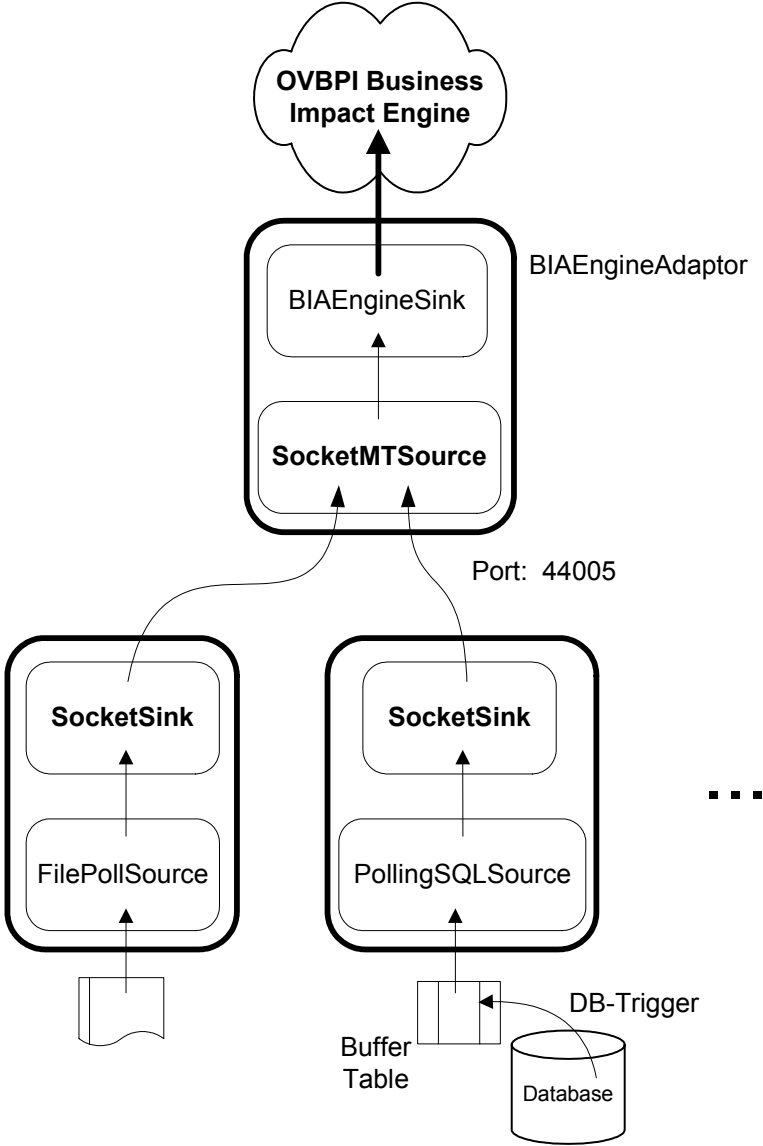
where:

- There is an adaptor running on the same server as the OVBPI Business Impact Engine

  Obviously the name of this adaptor could be whatever you want. In this diagram it is shown by the name: `BIAEventAdaptor`. (Where `BIA` was the original code name for the product now called OVBPI.)

  This `BIAEventAdaptor` is configured to get its input from a socket. It runs the `SocketMTSource` component as this allows for multiple client connections over the one port number.

  The port number can be whatever port you choose, so long as it is not used by another application. The diagram shows these adaptors connecting over port `44005`.

- For each of your input sources, you configure whatever adaptor components are required so long as you output to the `SocketSink` component and configure this to send to the port on which your `SocketMTSource` is listening

- When you need to add another input source, you simply configure a new adaptor that sends its output using a `SocketSink`, connecting to the same port as all the current adaptors

- Obviously for this architecture to work, the `BIAEngineAdaptor` needs to be running before any client adaptors can start up and connect

As you can see, this architecture allows for your source inputs to be on the local machine as well as being distributed across your network.

Adding new adaptors does not require any loss of up-time for your current running adaptors. Mind you, by default the `SocketMTSource` accepts up to 100 concurrent client connections. Thus, if you had that many (100) adaptors configured and running and you wished to configure an additional adaptor, this would require that you close down all running adaptors, reconfigure the `BIAEngineAdaptor` adaptor to accept more connections (for example, 200) and then re-start all your adaptors. For this reason, you should think carefully about how many adaptor you are likely to configure and set the `SocketMTSource` configuration accordingly.

Default Installation

When you install OVBPI on your system there is a default adaptor called `BIAEngineAdaptor`, and it is configured to run the `BIAEngineSink` and `SocketMTSource` components as shown in Figure 23 on page 195.

The `SocketMTSource` component defaults to listening on port 44005.

You are able to start this `BIAEngineAdaptor` adaptor using the OVBPI Administration Console - by starting/stopping the `Business Event Handler` component

The configuration file for this adaptor is found in the `OVBPI-install-dir`\data\conf\bia directory, in the file called: `bia_event_engineadaptor.props`

Hospitals

With this socket architecture the most obvious place to configure a hospital is in the `BIAEngineAdaptor`. That way, all data events are passed up to the OVBPI Business Impact Engine adaptor and, if an error occurs, that message can be written out to the hospital and the Event Handler can continue, processing the next message.

By default, the `BIAEngineAdaptor` adaptor is configured with a hospital.

Summary

This "socket based" architecture provides for a very efficient Event Handler.

The good aspects of this architecture are:

- Event transmission from source to the OVBPI Business Impact Engine is efficient and fast
- Configuring new adaptors into the network is very easy and low impact
- The `BIAEngineAdaptor` comes pre-configured with the OVBPI installation to use sockets
- If you have problems with any input (client) adaptors, you can restart them without any affect on the rest of the system

The possible down sides are:

- If the OVBPI Business Impact Engine goes down for any reason, then an exception is passed back to the client adaptors. This causes the client adaptors to terminate. See Restartable Adaptors on page 203
- If the `BIAEngineAdaptor` shuts down for any reason, then this causes all the adaptors shutdown. See Restartable Adaptors on page 203.
- It does not provide "event replay"

  That is, by default you do not have a persistent store (a "running log") of every event as they went into the OVBPI Business Impact Engine. Thus you can not ask for a reply of the events as they happened.

  This may not an issue for some sites, but some situations might desire this functionality.

Event Store Architecture

The "event store" architecture looks something like this:

Figure 24 Event Store Event Handler
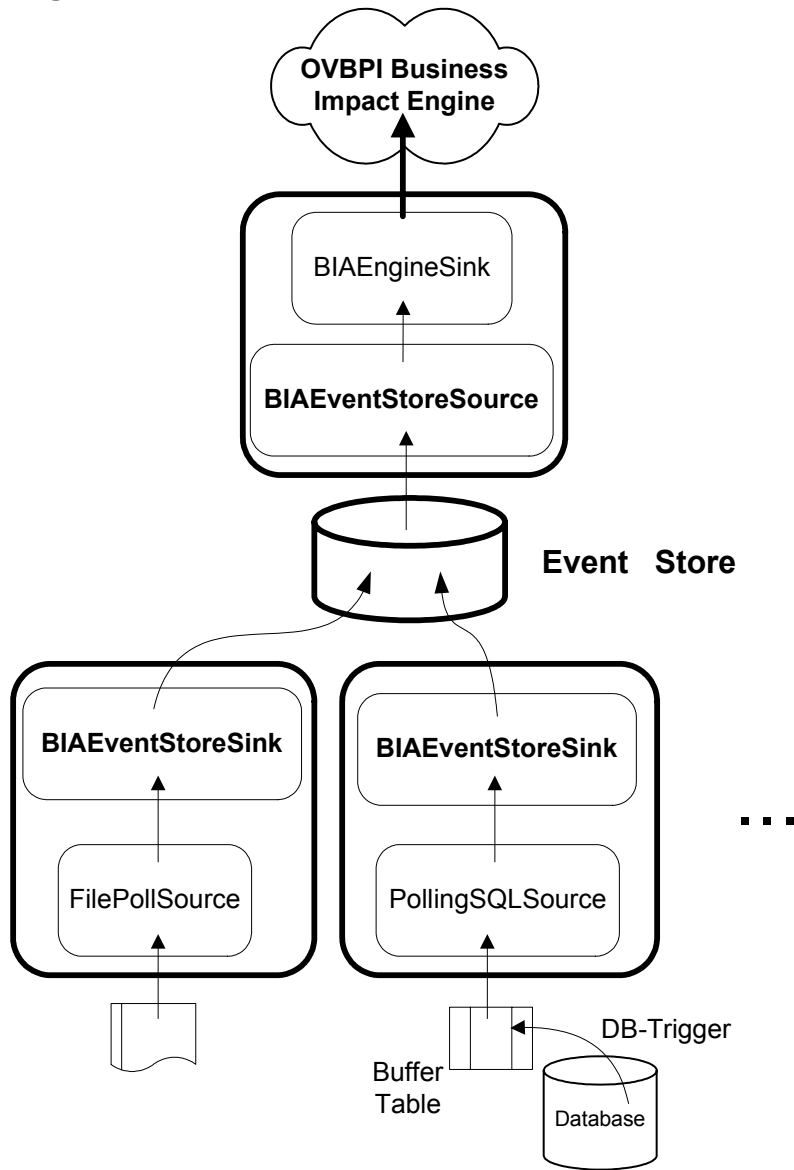
where:

- Again, the architecture relies on there being one main adaptor that talks to the OVBPI Business Impact Engine

  This adaptor reads all its input events from a central database table, using the `BIAEventStoreSource` component.

- For each input source you configure an adaptor to use whatever components are required, so long as it outputs to the `BIAEventStoreSink`

  The `BIAEventStoreSink` component writes the output event to the event store data table.

- When you need to add another input source, you simple configure a new adaptor that sends its output through to a `BIAEventStoreSink`, which then writes the events to the same event store data table

- Obviously for this architecture to work, the actual event store table needs to be created first within the database - and this is done as part of the OVBPI installation. You must also ensure that the event store data table is accessible to your network of adaptors

As you can see, this architecture allows for your source inputs to be on the local machine, or distributed across your network.

Adding new adaptors does not require any loss of up-time for your current running adaptors. The only issue is that your database must be accessible across the network for all your adaptors, and the database must allow the necessary number of connections. Obviously connection pooling is available using the appropriate JDBC driver.

Default Installation

When you install OVBPI on your system an event store table is created in your database. This table is called `EVENT_STORE`.

You would use this table if you chose to configure your Event Handler to use this architecture.

Hospitals

The adaptor reading from the event store would need a hospital to cope with error data events. Indeed, you must configure a hospital with this adaptor. If you do not, then when an error message occurs, the `BIAEngineSink` throws an error, the `BIAEventStoreSource` rolls back the transaction which leaves the data record in the database. When the `BIAEventStoreSource` then asks for the next record to process, it retrieves the same record - the record which caused the previous rollback. Thus, the adaptor loops on the same input record!

So why isn't there some logic in the `BIAEventStoreSource` such that it could cope better with errors? Because that's what hospitals are for.

So...always configure a hospital when configuring the `BIAEventStoreSource/BIAEngineSink` adaptor.

Summary

This "event store based" architecture provides for a safe and flexible Event Handler. Certainly, if you had a requirement that you needed to keep an audit trail of every event entering your system then this architecture can provide that.

The good aspects of this architecture are:

- Configuring new adaptors into the network is very easy and low impact

- The EVENT_STORE data table comes pre-configured with the OVBPI installation

- If you have problems with any input (client) adaptors, you can restart them without any affect on the rest of the system

- If the OVBPI Business Impact Engine goes down for some reason, then the only adaptor that is affected is the adaptor reading from the event store. All your other client adaptors continue to feed events into the event store as if nothing has happened

- You are able to shutdown the adaptor that is reading from the event store without affecting any other adaptors

- The BIAEventStoreSource can be configured to leave processed events in the event store and thus provide you with a complete chronological store of all the events that have come into your system. Thus providing you with an "event playback" or "auditing" facility

The possible down sides are:

- Event transmission from source to the OVBPI Business Impact Engine is perhaps not as efficient or as fast as for the socket-based architecture

- If you configured the BIAEventStoreSource to leave processed events in the event store table, then you must set up some sort of event store maintenance

  By default events are deleted from the event store after they are processed and so there is no need for any on-going maintenance concerns.

Restartable Adaptors

One of the main issues that people have with the socket-based event architecture, is that if the `BIAEngineAdaptor` adaptor goes down then the whole network of adaptors goes down. Clearly, no administrator wants to have to go out and restart all their adaptors.

This is also a valid concern if there are any network problems. That is, your `BIAEngineAdaptor` may be running just fine, however there may be networking problems that cause some of your adaptors to lose their connection and shutdown.

You need to configure your adaptor network to be robust. If an adaptor goes down for any reason, it needs to restart itself, or be monitored in some way.

The Business Event Handler Windows Service

The `BIAEngineAdaptor` (the `Business Event Handler`) is configured to run as a windows service on your OVBPI server. The Windows service name is `OVBPIEventHandler`.

By default, if this service aborts then it does **not** auto-restart.

You can configure the recovery behavior of this Windows service using the normal Windows control panel. Hence you can configure this to restart itself should it fail for any reason.

Client Adaptors

You should consider how best to set up your client adaptors such that, if they fail for any reason, they can auto restart. Thus, if they lose their connection to the Business Event Handler, they can restart and continue processing as before.

If the client adaptor is running on an OpenView Managed Node then you could use OpenView to monitor and restart the adaptor. If not running on a Managed Node then you could consider other technologies for providing adaptor restart capabilities.

One option is to set up the client adaptor as a Windows service. There is third party software available that offers this "Windows service" capability on both Windows and Unix environment. See Auto-ReStarting Adaptors on page 394 for more details and examples.

With your adaptors configured to auto-restart on failure you are able to configure a robust network of adaptors.

Sockets With Event Replay

If you like the socket-based architecture but require event replay or event auditing, then you could set up a combined approach that looks something like this:

Figure 25 Sockets With Event Replay Event Handler
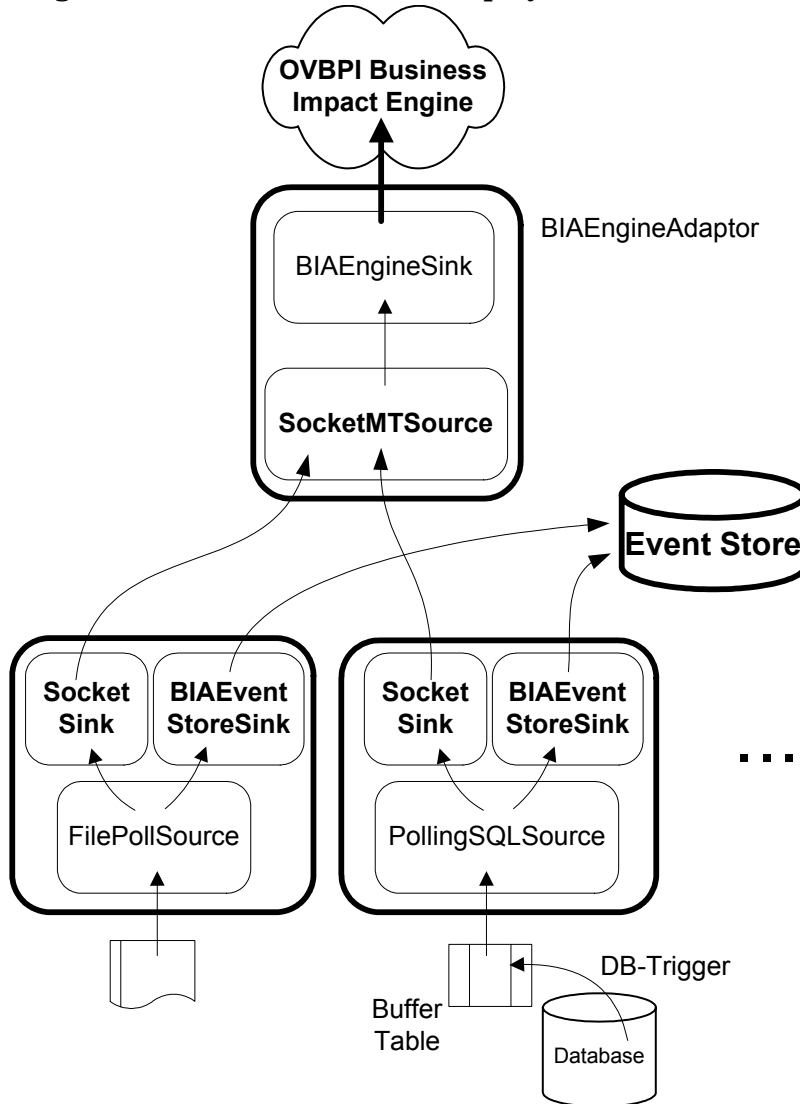
where:

- This architecture gives you the speed of the socket connections and the storage of the event store

- Obviously, if you were required to feed the event data from the event store into the Business Impact Engine you would need to configure and run a separate adaptor using the `BIAEventStoreSource` component

Alternatively, you could configure the following:

Figure 26 Sockets With Event Replay - alternative
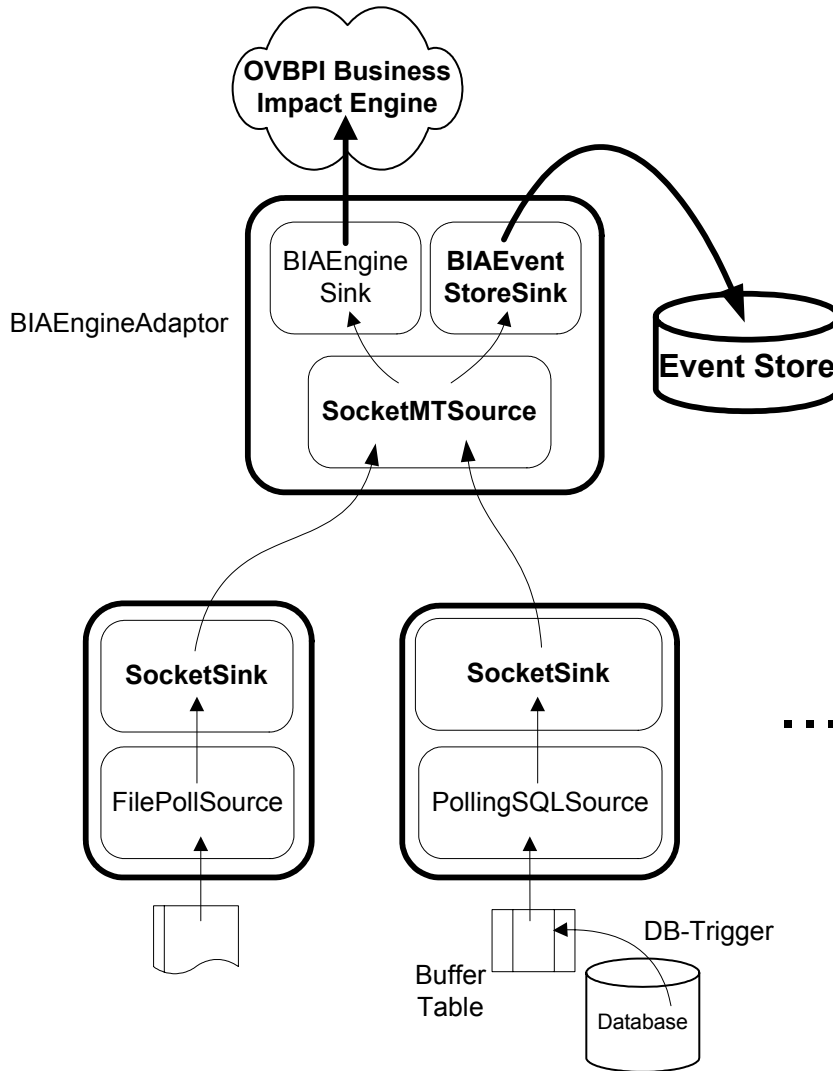
This would achieve the same result as Figure 25 on page 205 however you would only need one adaptor configured to write to the event store.

Altering The BIAEngineAdaptor

The `BIAEngineAdaptor` is started/stopped using the OVBPI Administration Console - the `Business Event Handler` component.

The configuration file for the `BIAEngineAdaptor` adaptor is found in the `OVBPI-install-dir`\data\conf\bia directory, in the file called: bia_event_engineadaptor.props

The default configuration is for a socked-based adaptor network.

My Config Changes Have Been Removed?

The `bia_event_engineadaptor.props` file is a normal openadaptor configuration file and so it might be tempting to edit this file directly. You can, and your changes take effect...however, if you later make any configuration changes (to anything) using the OVBPI Administration Console your changes to the `bia_event_engineadaptor.props` file **are lost**.

When you apply configuration changes using the OVBPI Administration Console, it applies the changes to a set of base template files and then copies these modified files into place. It does this to every configuration file regardless of what you have changed.

So, if you wish to make permanent changes to the `bia_event_engineadaptor.props` file then you actually need to make the changes to the base templates.

Config Template Files

The base template files are found in the directory:

`OVBPI-install-dir`\newconfig\DataDir\conf\bia

There are two files that you need to maintain:

bia_event_engineadaptor.mssql.props
bia_event_engineadaptor.oracle.props

The base template file used on your installation depends on which database your OVBPI installation is using. By editing both files each time you make an update you are making sure that if you ever switch databases your configurations are correct.

These template files are normal adaptor configuration files except for the fact that they contain some parameter substitution strings for some of the values used. That is, you see lines that contain strings within double brackets.

For example:

```
...
BIAEngineAdaptor.H1.Database   = {{BIA_ENGINE_MSSQL_DATABASE}}
BIAEngineAdaptor.H1.UserName   = {{BIA_ENGINE_MSSQL_USERNAME}}
BIAEngineAdaptor.H1.Password   = {{BIA_ENGINE_MSSQL_PASSWORD}}
...
```

The tags within the double brackets get replaced by the OVBPI Administration Console when you apply new configuration changes.

Making Config Changes

To make configuration changes to your Business Event Handler, you do the following:

1. Stop the Business Event Handler, using the OVBPI Administration Console

2. Make your configuration changes to the base template files (under the `newconfig` directory)

3. You need to activate these configuration changes. To do this you have two choices:

   a. Make the same changes to the live configuration file (in the `data\conf\bia` directory) - but make sure you use actual values for all the double-bracket entries. That is, make sure you manually expand each double-bracket tag to its correct result

   ...or...

   b. Run the OVBPI Administration Console and change any configuration option at all, and apply the change. This would correctly expand your new base template files and copy them into place

4. Start the Business Event Handler, using the OVBPI Administration Console

Of course, most people try out their configuration changes directly on the `bia_event_engineadaptor.props` file, and only once it is correct do they then go and update the base template files. This is OK, just so long as you do go and apply the changes to the base template files. If you do not, you will lose your configuration changes.

Alternative Test Option

If you want to try out the Event Store but you do not want to mess too much with the standard `BIAEngineAdaptor` configuration file, there is an alternative that you could try.

You could always leave the `BIAEngineAdaptor` configured as a socket-based adaptor, and configure your adaptor network as follows:

Figure 27 Event Store and Sockets Combined
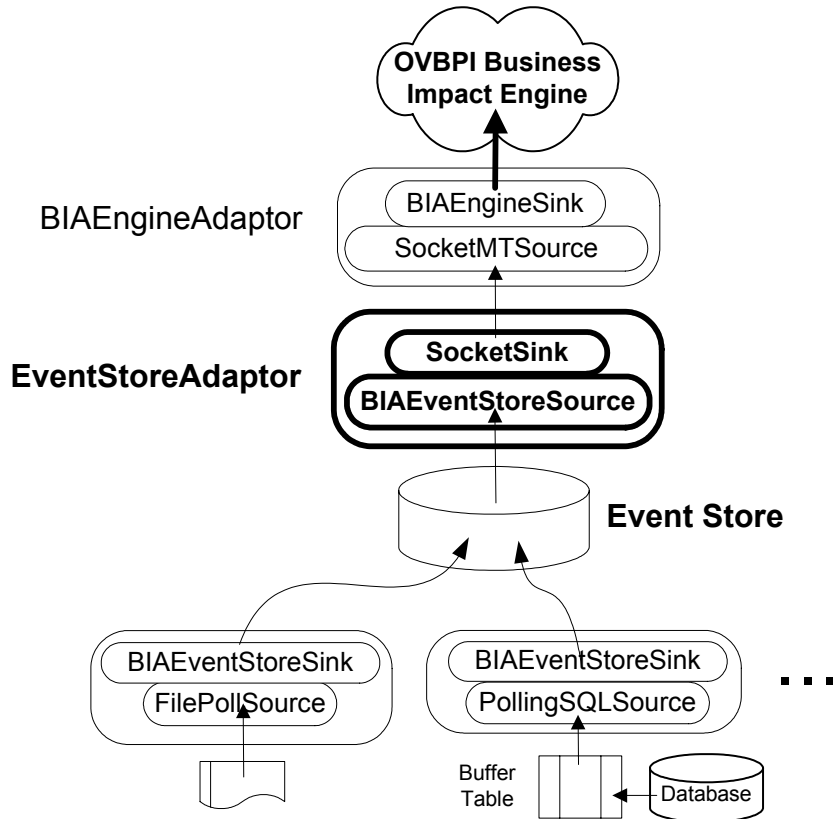
where:

- You configure your adaptor network to feed up to the event store

- You configure an additional adaptor on the OVBPI server to read from the event store and pass the events to the standard `BIAEngineAdaptor`

  This allows you to test out the use of the event store without having to reconfigure the `BIAEngineAdaptor`.

Once you are happy that you do want to use the event store you would be able to follow the steps described above in the section Making Config Changes on page 209 with confidence.

Summary

The choice of which Event Handler architecture you implement is down to you. Here are some summary thoughts that might help you to decide:

- Both the socket-based and event store-based architectures provide the ability to:

  — Add/Remove adaptors with ease

  — Fix up individual client adaptors without any effect on the others

- The event store architecture also provides:

  — Ability to take down the OVBPI Business Impact Engine whilst leaving your adaptor up and running - feeding events into the data store

  — (optionally) Event auditing/logging

- The socked-based architecture is presumably able to process events at a faster rate. It is not bound to the speed of a single database table

- When running the socket-based architecture, make sure your client adaptors are configured to auto-restart

Whichever architecture you choose, you can see the importance of having the adaptors feed up to a single adaptor which then handles the event transfer to the OVBPI Business Impact Engine. This idea also provides for your client adaptors to be placed wherever they need to be within your computing network.

If you can configure your adaptor network such that it is easy to start (restart) all adaptors, and you do not require event storage, then the pure socket-based approach may be for you. Otherwise, you should look into the event store-based architecture. The choice is your.

If you make any configuration changes to the Business Event Handler (`BIAEngineAdaptor`), make sure you also carry out these changes on the base template configuration files.

5 Events Definition and Configuration

This chapter looks at the connection between the events defined within the OVBPI Modeler and the events generated from your network of adaptors (your Business Event Handler).

This chapter looks briefly at how events are defined within the OVBPI Modeler, how they are deployed to the run-time event repository, and how the `BIAEngineSink` makes use of these.

This chapter then looks at how to actually configure your Business Event Handler to generate these defined events.

The OVBPI Modeler

Once configured and running, the OVBPI Business Impact Engine receives data events from the Event Handler (your network of adaptors). For the OVBPI Business Impact Engine to understand how to process each event it receives, you need some way to define the set of valid events and their content.

You define the set of valid events from within the **OVBPI Modeler**.

The OVBPI Modeler lets you define your flows, your service dependencies, and your data definitions. You can also define your **events** (referred to as **event definitions**).

When defining an event you are basically saying that you want the OVBPI Business Impact Engine to be able to accept an event of this name and content (referred to as `payload`). At this point you are just defining an event `definition`. You are not doing any adaptor configuration or anything like that. It is purely defining that at some point in the future you are expecting an event of this name and payload to arrive from the Event Handler, and you want the OVBPI Business Impact Engine to be able to understand it when it does arrive.

Let's look now at how to define an event in the OVBPI Modeler.

Event Name

Within the OVBPI Modeler, when you create a new event, it asks you for the **event name**. This name must be unique across all of your events.

The event name can actually be specified in a similar way to normal everyday file names. That is, the event name can be a simple unique name, or it can be a fully specified name including a path or grouping.

Some examples might help. Here are a series of valid event names:

```
myEvent
myEvent1
group1/myEvent
group2/myEvent
/group2/myEvent
/group1/group2/group3/anotherEvent
```

These are all valid and all would be treated as different, unique events.

When naming your events you need to think carefully about your naming policy and whether you wish to "group" events together, etc.

An event name consists of two main parts:

- The path name (optional)

  This is the part of the name up to the last "/" (if there is a "/" in the name).

- The actual event name (required)

  The part of the name after the last "/" character.

Event Payload

Once you have created an event and given it a name, you are then required to define the "payload" of the event. That is, the actual content of the event - the attributes that make up this event.

For each payload attribute, you can define the following fields:

- Name

  This is the name of the attribute as it appears within the data event.

- Type

  Here you select one of the following options:

  — String

    This says that the attribute contains a string value.

    If you specify a string data type then you **must** also specify the maximum length that this string can be. You specify this length in the `Constraints` column for this attribute.

  — Double

    This says that the attribute contains a double value.

  — Integer

    This says that the attribute contains an integer value.

  — Currency

    This says that the attribute contains a currency (money) value. As far the actual data inside this field, it can be any of the supported numeric types - Integer, Long or Double.

In the `Contraints` column for this attribute you **must** then specify the type of currency that this field contains. You specify the currency by typing in a valid ISO 4217 currency code. For example:

```
EUR - Euro
USD - American dollars
AUD - Australian dollars
GBP - British Pounds
etc...
```

— Date

This says that the attribute contains a date value.

Refer to Date/Time Formats for OVBPI Events on page 225 for more details.

— Long

This says that the attribute contains a long value.

— Boolean

This says that the attribute contains a string value containing the word "true" (in any case - upper, lower or mixed). Any other value is taken to represent "false".

- Constraint

    You need to specify values in this column for the data types:

    — String

    — Currency

- Description

    You can enter a text string to record your comments about this attribute.

You define all the attributes that make up this event.

You would repeat this procedure to define all the events that you need for your OVBPI system to operate.

Deploying the Event Definitions

Once you have defined the set of events that you wish you OVBPI Business Impact Engine to be able to process, you must **deploy** them.

When you issue a deploy from within the modeler, the OVBPI Business Impact Engine is updated with all the latest definitions for data definitions, events, flows, etc. and, the event definitions are written to the **run-time event repository**.

The Run-Time Event Repository

The run-time event repository is used by the `BIAEngineSink` adaptor component.

Figure 28 The Run-time Event Repository
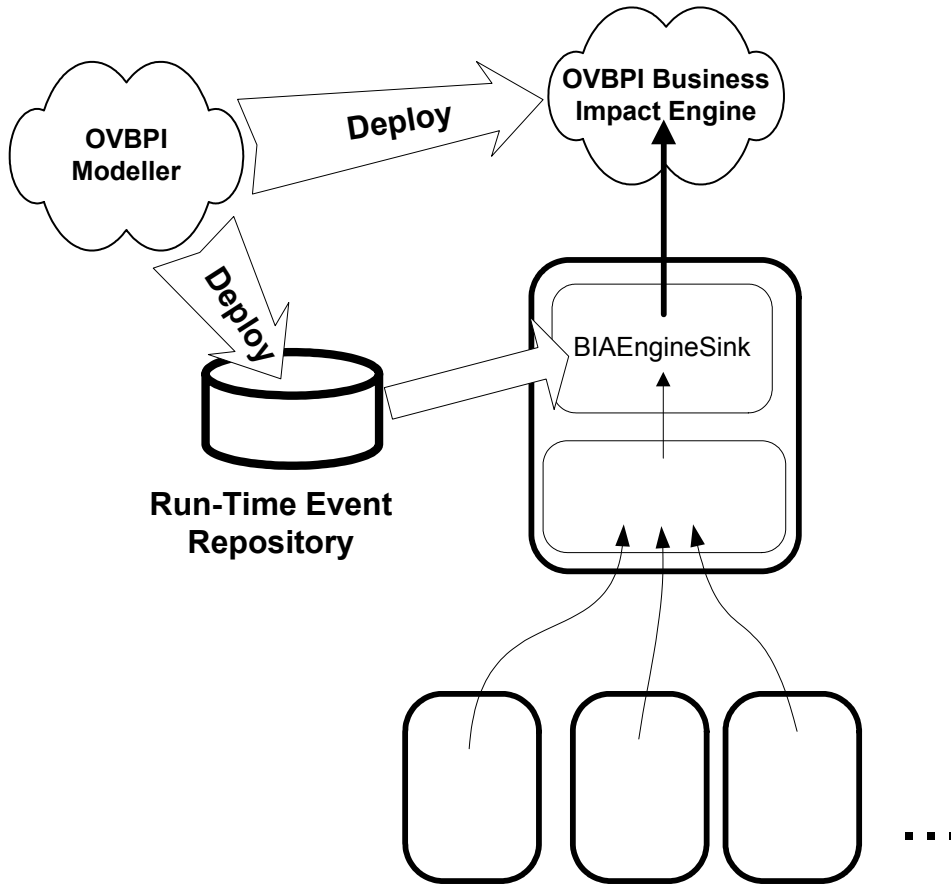
where:

- When you issue a deploy from within the OVBPI Modeler, all changes to the event definitions (additions, updates, deletes) are deployed to the run-time event repository

- The `BIAEngineSink` uses the run-time event repository to know what events it is allowed to pass through to the OVBPI Business Impact Engine

Location

The run-time event repository is located under the *OVBPI-install-dir*\data\repository\events directory, with each deployed event definition maintained within its own XML file.

Do not edit these files or you may cause corruption of your run-time event definitions.

BIAEngineSink

The BIAEngineSink is responsible for sending events to the OVBPI Business Impact Engine. The BIAEngineSink is also responsible for filtering out any stray or badly-formed events.

By having run-time access to the latest deployed event definitions, the BIAEngineSink is able to carry out this filtering process.

The BIAEngineSink caches the event definitions for efficiency. However, the cache is able to detect changes/updates within the run-time event repository with the cache being updated in real time. So a running BIAEngineSink automatically picks up all new or modified event definitions the moment the deploy is performed.

If an event arrives at the BIAEngineSink and it has an unknown event name, the event is rejected. A HOSPITAL exception is thrown, and the event is placed into the Event Hospital.

When an event arrives and the name is known within the run-time event repository, the BIAEngineSink pulls out only the attributes as defined within the run-time repository. If an attribute is missing, or has an incorrect data type, then a HOSPITAL exception is thrown. Assuming all the defined attributes are found and are of correct data type, an "Engine Event" is constructed and sent into the OVBPI Business Impact Engine. Any attributes that were in the original data event, but were not in the event definition, are simply ignored.

Configuring the Actual Events

Let's revisit a previous diagram, and consider what is defined so far:

Figure 29 What Is Defined So Far
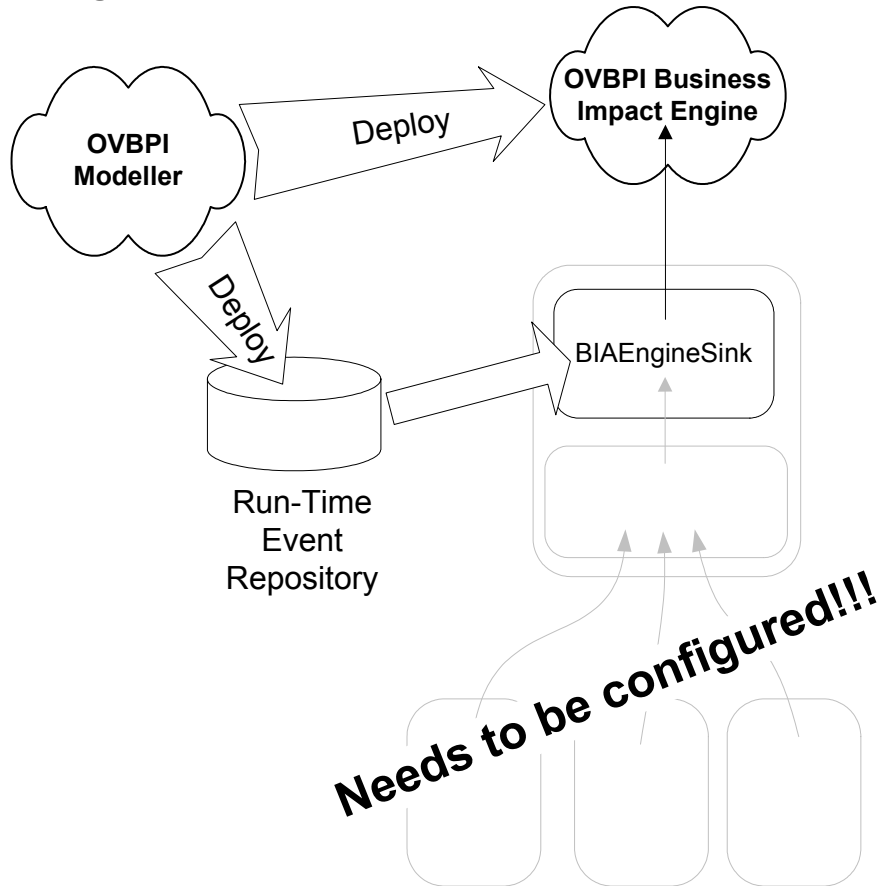
where:

- The events are defined
- The event definitions are deployed to the run-time repository
- A `BIAEngineSink` is now able to pick up these event definitions and thus filter and process the correct event
- The OVBPI Business Impact Engine is able to process valid events

The only problem is that you do not yet have any actual adaptors configured or running to feed these events up to the `BIAEngineSink`.

That is, once someone has modelled and deployed there flows, data definitions, event definitions, etc., they have described what they would like to have happen. It is then up to the person responsible for configuring and maintaining the Business Event Handler to actually configure the necessary set of adaptors to actually generate these events.

The event definitions from the OVBPI Modeler define the events that the system accepts - their names and payloads - and it is up to you to configure adaptors to access the appropriate databases, files, whatever, using whatever pipes and filters are necessary, and then send these events into the OVBPI Business Impact Engine.

Indeed, it may be that when you investigate the data sources available to you that you have trouble constructing every event exactly as defined within the OVBPI Modeler. Obviously this would require you to feed back to the person who had modelled the events in the first place, and you would then need to work together to see whether some event definitions could be modified etc. until you could both agree a set of events that would drive the modelled flow(s) and be constructed from the data sources available to you. Configuring these events may not always be as straightforward a task as you would like.

Standard Event Header

By the time an event reaches the `BIAEngineSink` it must contain certain data attributes to allow the Engine sink to identify the event.

The standard event header attributes are as follows:

- `EventName` (**required**)
- `EventGroup` (**optional**)
- `GeneratedDate` (**optional - but recommended**)

EventName/EventGroup

In the earlier section Event Name on page 214, it mentioned that you could think of the overall event name consisting of two parts - the "path" and the actual "name".

In a standard event header, the full event name can be expressed either in the attribute called `EventName`, or it can be broken into two parts with the "path" being specified in the `EventGroup` attribute and the actual name being specified in the `EventName` attribute.

Most people simply set the `EventName` to contain their complete event name.

For example:

> If the event is defined as: e_group1/e_group2/myEvent

> You could configure the event header as:

> ```
> EventGroup:   e_group1/e_group2
> EventName:    myEvent
> ```

> Or, you could simply configure the event header as:

> ```
> EventName:    e_group1/e_group2/myEvent
> ```

The `EventName` and `EventGroup` attributes can be read as normal fields from within the actual source object itself (file, database, etc.), or can be added by an adaptor (for example, by using the `AdornmentPipe`).

GeneratedDate

If you supply a timestamp in the `GeneratedDate` attribute then this timestamp becomes the "Creation Date" for this event when it is created within the OVBPI Business Impact Engine.

If this attribute is not supplied then the `BIAEngineSink` simply sets the creation date to be the current system time of when the event arrives. This can cause real problems if your events arrive out of sequence.

It is best to set the `GeneratedDate` attribute within the event at the point nearest the source of this event. This means that the creation date for this event within the OVBPI Business Impact Engine reflects the actual date/time that this event was generated and not the time that event entered the OVBPI Business Impact Engine.

The `GeneratedDate` attribute can be read as a normal field from within the actual source object itself (file, database, etc.), or can be added by the adaptor (for example, by using the `AdornmentPipe`).

➤ It is recommend that you always produce events with the `GeneratedDate` attribute set. Without the `GeneratedDate` attribute set, the OVBPI Business Impact Engine has no way to correctly handle any events if they arrive out of sequence, which can have serious implications on your flow monitoring.

Here are some example ways to set the `GeneratedDate` for an event:

- You could set the creation date to `17 June 2003 14:56:57 GMT+1` by populating the `GeneratedDate` attribute within your source (file, database, etc.) to be the value `1055858217077`. Which is the representation of the date expressed as the number of milliseconds since 1970

- If you data source has no way of containing a data field, you could use the `AdornmentPipe` to add a `__TIMESTAMP` attribute, as follows:

```
A.AP.AdornType1.AttName1   = GeneratedDate
A.AP.AdornType1.AttValue1  = __TIMESTAMP
A.AP.AdornType1.AttType1   = Long
```

This would at least mean that a timestamp was added at the time the event was read from the data source.

Refer to Date/Time Formats for OVBPI Events on page 225 for more details about how to handle date and time attributes/formats.

Normalized Events

You may hear the terms **Normalized Business Event**, or **Normalized Engine Event**. These two terms mean the same thing. They both simply refer to a data event that contains the "standard event header" - as described in Standard Event Header on page 222.

That is, for an event to be accepted by the `BIAEngineSink` it must be normalized. All this means is that each event contain the attributes `EventGroup` (optional), `EventName` (required) and `GeneratedDate` (optional) somewhere within their payload of attributes.

Case-Insensitive Event Names

When you define an event within the OVBPI Modeler, you are able to define the event name using mixed-case. At run-time, all event names are treated as case-**in**sensitive.

Case-Insensitive Attribute Names

When you define an event within the OVBPI Modeler, you are able to specify each attribute name using mixed case (lower and/or upper case). However, be aware that at run-time, all event attribute names are treated as case-**in**sensitive.

So, if an event is configured to contain the attribute: `OrderNO`, the actual event must contain this attribute, however the name could be `ORDERNO`, or `orderNo`, or any combination of lower or upper case.

Date/Time Formats for OVBPI Events

Date values must be expressed in one of the following four formats:

1. A string, or numeric, value representing the number of milliseconds since 1970

   For example: `1066060093765`

   would represent the date: `10 October 2003 16:48:13 GMT`

   See Date/Time Formats and openadaptor on page 261.

2. A string representing the date in the format:

   `YYYY-MM-DD TZN`

   where:

   — The time zone (TZN) part is optional

   For example:

   ```
   2004-11-23
   2004-06-03 GMT
   ```

   See Date/Time Formats and openadaptor on page 261.

3. A string representing the date and time in the format:

   `YYYY-MM-DDThh:mm:ss TZN`

   where:

   — The time zone (TZN) part is optional

   — The date and the time parts are separated by the character `T` as shown

   For example:

   ```
   2004-11-23T09:56:01 GMT
   2004-06-03T23:32:00
   ```

   See Date/Time Formats and openadaptor on page 261.

4. A string matching the `java.text.DateFormat` getDateTimeInstance() method, using the long date and time style

This format is dependent on the system locale setting.

For example:

`13 October 2003 16:48:13 GMT`

See Date/Time Formats and openadaptor on page 261.

This means that if your data source holds the dates in any of these exact string formats then you can import them directly into OVBPI. These strings are then converted into dates within the OVBPI Business Impact Engine.

When your date fields do not conform to these exact formats you simply need to configure your adaptor to handle this. Refer to Date/Time Formats and openadaptor on page 261 to see a number of different examples.

Choosing Your Event Handler Architecture

As discussed in Chapter 4, Event Handler Architecture, you need to decide on the architecture to use when setting up your network of adaptors. In particular, you need to decide whether to send events in using an event store or direct over a socket connection.

With your `BIAEngineAdaptor` configured and in place, your OVBPI installation looks something like this:
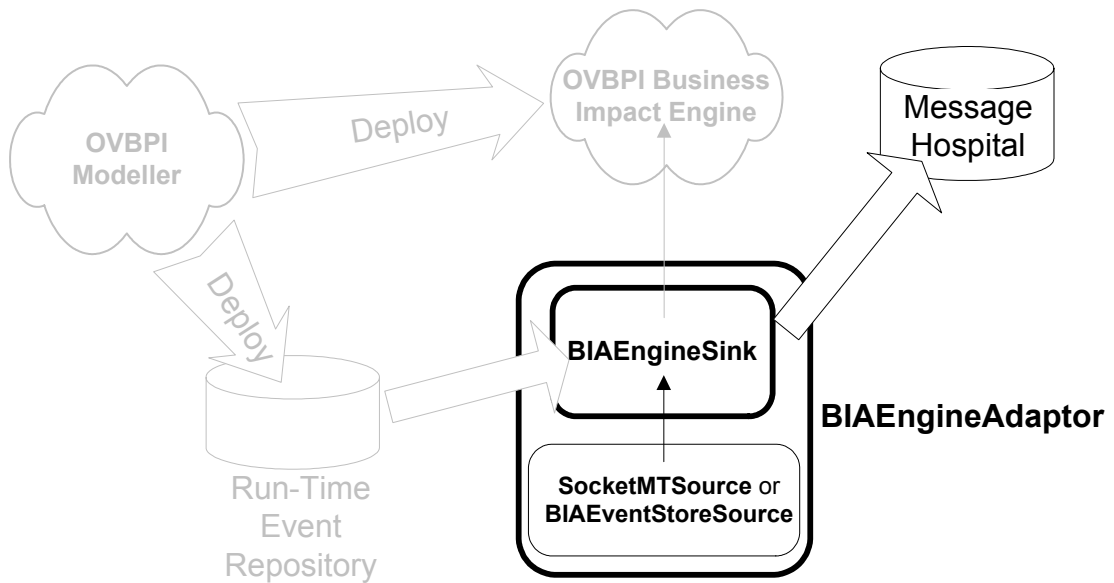
It is then a matter of configuring individual adaptors to feed events to this `BIAEngineAdaptor`.

Events From a Flat-File

Suppose an event has been defined in the OVBPI Modeler as follows:

```
Event Name: Sales/Web/NewCustomer

Payload:
    CustomerID - String(30)
    Name       - String(50)
    Address    - String(200)
    Phone      - String(50)
    Email      - String(100)
```

Your job is to actually configure an adaptor to produce this event.

After investigating your Web site sales application, you discover that as a new customer registers with your Web site, their details are written to a central database and the assigned customer ID is then logged to a flat-file.

You realize that you can configure a file source component to monitor this flat-file, and for each record it reads, use an indirection pipe to lookup this customer ID in the database and thus build the data record.

Whether you use a component such as the `FilePollSource`, or the `FileRollSource` depends on whether the Web site writes out to a single flat-file or rolls across many files.

Assuming the Web site writes out a single flat-file, you choose the `FilePollSource` - this provides you with the `CustomerID` attribute. By using the `IndirectionPipe` you add the extra attributes of `Name`, `Address`, `Phone` and `Email`. However, the database holds this data under the names: `CustName`, `Address`, `Phone` and `Email`. So you could use one of the standard openadaptor alias pipes to rename the `CustName` attribute to be called `Name` so that it matches the desired event definition. (See Attribute Aliasing on page 233 for another easier alternative.)

But...how do you build the event header? Your adaptor must be able to label these events as being of the correct name. You also need to consider how you might add the generated date attribute to each event to say when it was constructed and sent into the Event Handler.

To construct the event header you can use the `AdornmentPipe`.

Pictorially, you want to build the following adaptor network:

Figure 30 Events From a Flat File
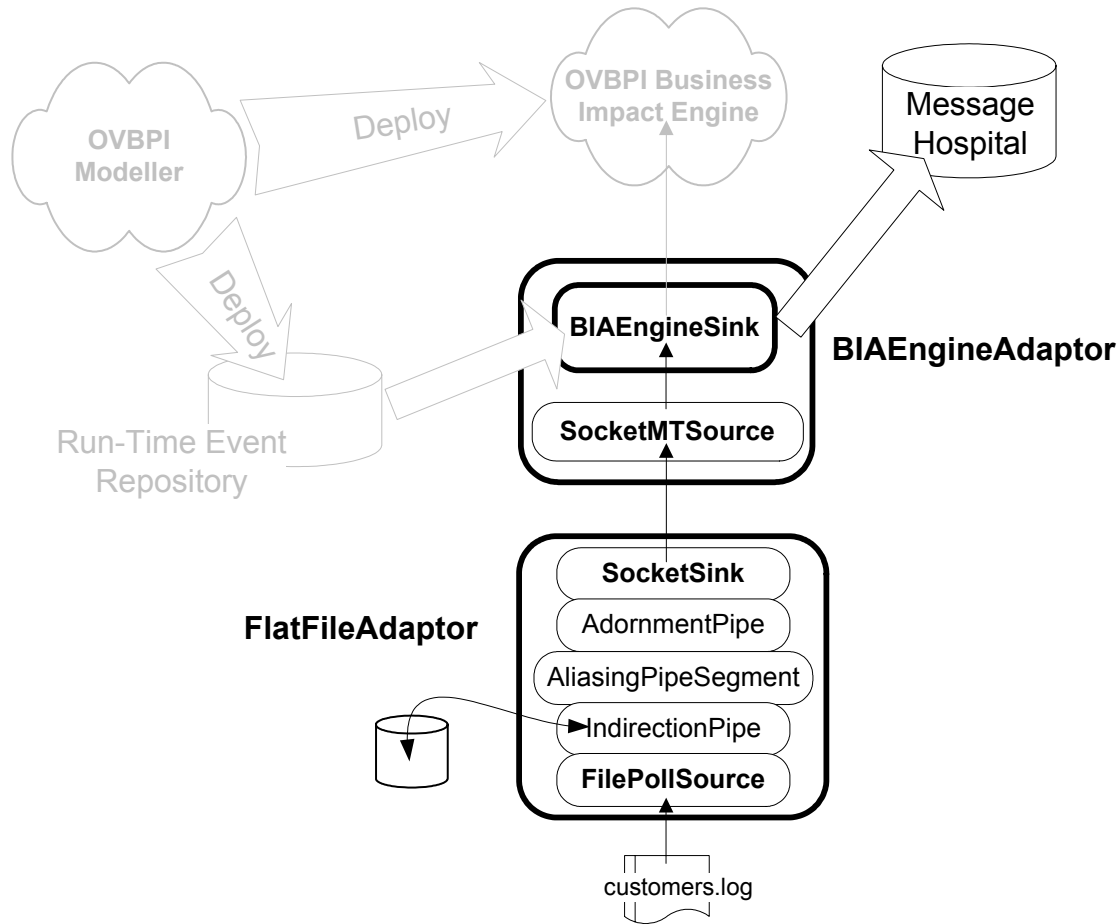
The configuration file for your `FlatFileAdaptor` might look as follows:

```
# List and name all components for this adaptor
# --------------------------------------------
A.Controller.Name = Controller
A.Logging.Name    = Logging
A.Component1.Name = FS
A.Component2.Name = Sock
A.Component3.Name = AP
A.Component4.Name = IP
A.Component5.Name = Alias


# Show how the components link together
# ------------------------------------
A.FS.LinkTo1       = IP
A.IP.LinkTo1       = Alias
A.Alias.LinkTo1    = AP
A.AP.LinkTo1       = Sock


# Source component is the FilePollSource
# --------------------------------------
A.FS.ClassName        = org.openadaptor.adaptor.standard.FilePollSource
A.FS.DOStringReader   = org.openadaptor.dostrings.DelimitedStringReader

A.FS.InputFileName    = customers.log
A.FS.NumAttributes    = 1
A.FS.AttName1         = CustomerID
A.FS.PollSource       = true
A.FS.SaveFilePosition = true


# Sink component is a SocketSink
# ------------------------------
A.Sock.ClassName  = org.openadaptor.adaptor.standard.SocketSink
A.Sock.Port       = 44005
A.Sock.HostName   = localhost


# IP is the IndirectionPipe
# -------------------------
A.IP.ClassName = org.openadaptor.adaptor.standard.IndirectionPipe
```

```
# For MSSQL
# ----=====
A.IP.Connection1.JdbcDriver       = com.inet.tds.TdsDriver
A.IP.Connection1.JdbcUrl          = jdbc:inetdae7://localhost:1433
A.IP.Connection1.UserName         = username
A.IP.Connection1.Password         = password
A.IP.Connection1.PasswordEncoding = true/false
A.IP.Connection1.Database         = databasename

# Now configure the actual SQL Select
# ------------------------==============
A.IP.Type1      = __DEFAULT

A.IP.Type1.Lookup1.SQL    = select CustName, Address, Phone, Email
                              from testcustomers where
                                    customerid = @@CustomerID@@
A.IP.Type1.Lookup1.Trim   = true


# Configure any name aliasing required
# ------------------------------------
A.Alias.ClassName         = org.openadaptor.adaptor.standard.AliasingPipeSegment

A.Alias.Type1             = Any Any
A.Alias.Type1.Alias1      = CustomerID CustomerID
A.Alias.Type1.Alias2      = CustName Name
A.Alias.Type1.Alias3      = Address Address
A.Alias.Type1.Alias4      = Phone Phone
A.Alias.Type1.Alias5      = Email Email


# AP is the AdornmentPipe
# -----------------------
# Add the event header
# --------------------
A.AP.ClassName  = org.openadaptor.adaptor.standard.AdornmentPipe

A.AP.AdornType1             = __DEFAULT

A.AP.AdornType1.AttName1  = EventName
A.AP.AdornType1.AttValue1 = Sales/Web/NewCustomer

A.AP.AdornType1.AttName2  = GeneratedDate
A.AP.AdornType1.AttValue2 = __TIMESTAMP
A.AP.AdornType1.AttType2  = Long
```

            where:

- The `FilePollSource` reads the `customer.log` text file. This provides the `CustomerID` attribute

- The `IndirectionPipe` then uses this `CustomerID` to do a database lookup and retrieve the attributes: `CustName`, `Address`, `Phone` and `Email`

- The `AliasingPipeSegment` then aliases all attributes such that they have the correct names for the event definition

- The `AdornmentPipe` then adds on the standard header information to identify this event as being the `Sales/Web/NewCustomer` event

Attribute Aliasing

The example adaptor configured in Events From a Flat-File on page 228 shows the use of the standard openadaptor pipe `AliasingPipeSegment`. This is a general purpose pipe that allows you to rename attributes within an event.

Instead of using the `AliasingPipeSegment` pipe, you can achieve this renaming/aliasing directly within the `IndirectionPipe`.

The database holds the data under the names: `CustName`, `Address`, `Phone` and `Email`, however, you need the attribute names to be `Name`, `Address`, `Phone` and `Email`. That is, the database attribute `CustName` needs to be aliased to be `Name`.

You can achieve this name-aliasing directly using the aliasing ability of the SQL SELECT statement.

For example, you can specified the indirection pipe as follows:

```
# Now configure the actual SQL Select
# -------------------=================
A.IP.Type1    = __DEFAULT

A.IP.Type1.Lookup1.SQL    = select CustName "Name", Address, Phone, Email
                            from testcustomers where
                                    customerid = @@CustomerID@@
A.IP.Type1.Lookup1.Trim  = true
```

where:

- You specify that the `CustName` attribute is to be called `Name`

  The use of double-quotes is optional. It just preserves the case-sensitivity of the aliased name. That is, you could use the select statement:

  ```
  select CustName Name, Address, Phone, ...
  ```

  This would work just as well, however on some database implementations the `CustName` column would be aliased to `NAME` and not `Name`.

- Thus you do not require the `AliasingPipeSegment` component

So the indirection pipe can handle attribute aliasing directly within the SQL statement(s) itself. If your aliasing needs are more general then you can always use the `AliasingPipeSegment` component.

Events From a Database

Suppose an event has been defined in the OVBPI Modeler as follows:

```
Event Name: Sales/Web/NewOrder

Payload:
    OrderID      - String(30)
    OrderAmount  - Currency/USD
    CustomerID   - String(40)
    OrderDate    - Date
```

When an order is placed on your Web site a record containing the above data is written to the `Orders` data table.

To access data from a database you would use the `PollingSQLSource` component. Although this comes as standard with openadaptor, you would use the version that comes with OVBPI because there are a couple of important enhancements, which are described in the section

But how do you set up an adaptor to be able to send out events as and when these orders are written to this table?

The usual issue with accessing a data table directly is that you need some way to identify the newly added data records. You could redefine the `Orders` table to include an addition "status" column and then configure the adaptor to update this column as it processes new records. Another way is to make use of a feature of SQL and define a **trigger**.

To monitor new orders you can configure a trigger such that, when a new record is added to the `Orders` table, the trigger writes a separate record to a secondary table. This secondary table is often called a `buffer` table. You configure the trigger to write out just the parts of the order that you require for your adaptor. This buffer table entry is then yours to manipulate a read using an adaptor. It is effectively a copy of the actual data and thus does not affect any current applications accessing the `orders` table.

It is a good idea to also have the trigger assign the OVBPI event header - `EventName`, `EventGroup` (optional) and `GeneratedDate`. You should definitely assign a timestamp to each entry written using this trigger as this captures the order that the records are written to the data table. This is essential if you want the OVBPI Business Impact Engine to cope with these events if they should arrive out of sequence.

The idea of setting up a trigger looks something like this:
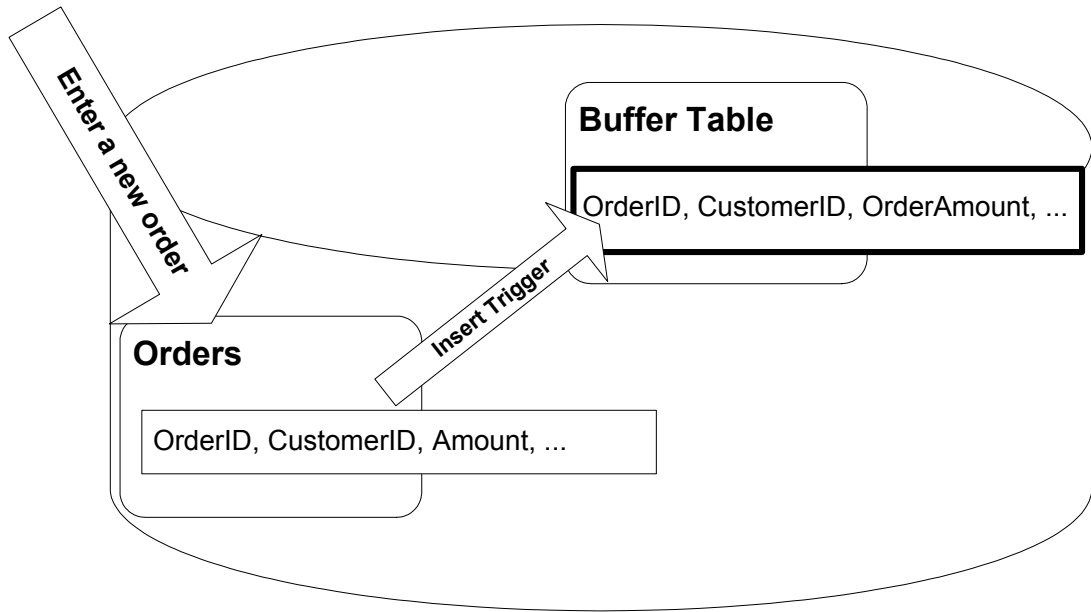
where:

- Your application(s) write a new record to the `Orders` table

- An SQL trigger is configured within the database such that every new insert into the `Orders` table generates a new record to be written to the designated buffer table

- The trigger specifies exactly what attributes are written into the buffer table

- You then configure an adaptor to process the records from this buffer table, and feed these as events up to the `BIAEngineAdaptor`

Defining the Buffer Table

Before defining the trigger, you must first create the additional buffer table. This is the table into which the trigger writes its records.

Here is the buffer table for this example (using MSSQL):

```
CREATE TABLE BIA_BUF__orders (
    EventName     varchar (50) NOT NULL,
    GeneratedDate datetime     NOT NULL,

    TheUID        uniqueidentifier,

    OrderId       varchar (20),
    CustomerID    varchar (30),
    OrderAmount   money,
    OrderDate     datetime,

    EventStatus   varchar (50),

);
```

where:

- You configure the trigger to generate the required event header and thus you define here the attributes: EventName and GeneratedDate

- You configure the trigger to assign every record that it writes to this buffer table with a unique ID. This allows you to uniquely identify this record when the adaptor processes this table

    The GeneratedDate attribute also allows the adaptor to process these records in the correct chronological order.

- You then have the attributes that make up the actual data of the record: OrderID, CustomerID, etc.

Defining the Trigger

Now you can define the trigger that writes data into this buffer table. You define this trigger to be on the Orders table, and for it to fire whenever a new record is inserted.

Here is the definition of the trigger for this example (using MSSQL):

```
CREATE TRIGGER BIA_TRIG__orders  ON Orders
AFTER INSERT
AS
BEGIN
   DECLARE
       @orderID       varchar(20),
       @customerID    varchar(30),
       @OrderAmount   money,
       @OrderDate     datetime

   SELECT
       @orderID      = OrderID,
       @customerID   = CustomerID,
       @OrderAmount  = Amount,
       @OrderDate    = OrderDate
    FROM inserted

    INSERT INTO
      BIA_BUF__orders
      (
          EventName, GeneratedDate,
          TheUID,
          OrderID, CustomerID, OrderAmount, OrderDate,
          EventStatus
      )
      VALUES
      (
          'Sales/Web/NewOrder',  getdate(),

          newid(),

          @orderID,
          @customerID,
          @OrderAmount,
          @OrderDate,

          'NEW'
      )

END;
```

where:

- This trigger is configured to fire when a new record is inserted into the `Orders` table.

- You then configure the appropriate SQL to generate the new record within the `BIA_BUF__orders` buffer table

An equivalent trigger definition for Oracle can be found in the file:

    sols/DB_trigger_myOrders.oracle

This example assumes that all the necessary data to build the event is within the actual orders data records. Obviously if this was not the case then you would build a more involved trigger, or make use of the `IndirectionPipe` within the adaptor to do additional data lookups.

Also, notice that the trigger is building the standard event header (`EventName` and `GeneratedDate` attributes). If you did not build the event header within the trigger, then you would need to build it using an `AdornmentPipe` within the adaptor (refer to Events From a Flat-File on page 228 for an example of setting this up). Either mechanism is fine, however, by building the `GeneratedDate` within the database trigger you are time stamping each event as it happens and this means that the OVBPI Business Impact Engine is able to use this timestamp if the events happen to arrive to the Business Impact Engine out of sequence (for whatever reason). Therefore it is always a good idea to **timestamp your events directly within the database trigger.**

Configuring the Adaptor

Let's now look at an adaptor configuration that processes this
`BIA_BUF__orders` buffer table and sends in new order events as and when
they arrived:

```
# List and name all components for this adaptor
# --------------------------------------------
B.Controller.Name  = Controller
B.Logging.Name     = Logging
B.Component1.Name  = DB
B.Component2.Name  = Sock


# Show how the components link together
# ------------------------------------
B.DB.LinkTo1       = Sock


# Sink component is a SocketSink
# ==============================
B.Sock.ClassName   = org.openadaptor.adaptor.standard.SocketSink
B.Sock.Port        = 44005
B.Sock.HostName    = localhost


# Configure a DB source (new Orders)
# ==================================
B.DB.ClassName     = org.openadaptor.adaptor.jdbc.PollingSQLSource

B.DB.JdbcDriver              = com.inet.tds.TdsDriver
B.DB.JdbcUrl                 = jdbc:inetdae7://localhost:1433
B.DB.UserName                = username
B.DB.Password                = password
B.DB.PasswordEncoding        = true/false
B.DB.Database                = databasename
B.DB.TransactionIsolationLevel = TRANSACTION_READ_COMMITTED


B.DB.PollPeriod      = 5000
B.DB.ExitOnError     = true

B.DB.QuoteMultipleKeys = true
B.DB.BatchSize       = 100
```

```
B.DB.UsingInClauses    = true
B.DB.PrimaryKeyRegExp  = PK
B.DB.NextPrimaryKeySQL = select TheUID from BIA_BUF__orders
                                  where EventStatus = 'NEW'
                                            order by GeneratedDate
B.DB.SelectSQL         = select * from BIA_BUF__orders
                                  where TheUID IN ('PK')
                                            order by GeneratedDate
B.DB.CommitSQL         = delete BIA_BUF__orders
                                  where TheUID IN ('PK')
```

where:

- The `PollingSQLSource` is configured to process records from the `BIA_BUF__orders` data table

- To limit the number of records retrieved with each poll of the buffer table, you can use the `BatchSize` option

  Alternatively, you could apply the limit within the actual SQL command(s) themselves. Refer to BatchSize and Limiting the Records Selected on page 245 for more details.

- This example uses "`select *`" in the `SelectSQL` property. As a general rule you should **not** use `select *` in your adaptors. You should always fully specify the columns that you wish to use

Let's consider in more detail the configuration of this `PollingSQLSource` component...

Configuring PollingSQLSource

You configure the `PollingSQLSource` component in three main parts:

- Basic connection details
- Basic setup - such as polling period
- The actual SQL used to gather the data records

Let's look at them in more detail.

Basic Connection Details

```
B.DB.JdbcDriver              = com.inet.tds.TdsDriver
B.DB.JdbcUrl                 = jdbc:inetdae7://localhost:1433
B.DB.UserName                = username
B.DB.Password                = password
B.DB.PasswordEncoding        = true/false
B.DB.Database                = databasename
B.DB.TransactionIsolationLevel = TRANSACTION_READ_COMMITTED
```

Here you simply specify the necessary parameters to connect to your database.

Basic Setup - Such as Polling Period

```
B.DB.PollPeriod      = 5000
B.DB.ExitOnError     = true
```

These are two parameters worth considering.

By default the `PollingSQLSource` polls the data table every second (1000 msecs). You might want to slow the polling down to (say) five seconds, or longer depending on your needs for data.

If an error occurs, the `PollingSQLSource` exits, and closes down. The idea is that this adaptor sends events to the OVBPI Event Handler. If any transient errors occur, such as the receiving socket being closed, or the OVBPI Business Impact Engine going down, this `PollingSQLSource` adaptor shuts down. If you have your adaptor set up to restart itself, then the adaptor restarts at a later date and resends the event.

The Actual SQL Used to Gather the Data Records

```
B.DB.QuoteMultipleKeys = true
B.DB.BatchSize         = 100

B.DB.UsingInClauses    = true
B.DB.PrimaryKeyRegExp  = PK
B.DB.NextPrimaryKeySQL = select TheUID from BIA_BUF__orders
                                where EventStatus = 'NEW'
                                  order by GeneratedDate
B.DB.SelectSQL         = select * from BIA_BUF__orders
                                where TheUID IN ('PK')
                                  order by GeneratedDate
B.DB.CommitSQL         = delete BIA_BUF__orders
                                where TheUID IN ('PK')
```

Each time the PollingSQLSource polls the data table it issues the statement you specify in the NextPrimaryKeySQL option. This statement must be such that it searches the table and return a list of unique IDs such that further SQL statements can process each of these records.

In this example, the source looks for all the records in the table that have their EventStatus attribute set to the text "NEW". The example makes use of the BatchSize configuration option to limit the number of records retrieved with each poll to 100. Alternatively, you could configure a limit into the actual SQL command(s) used. Refer to BatchSize and Limiting the Records Selected on page 245 for more details.

The SQL configured for the SelectSQL option is then called to actually process each selected record. In the SelectSQL option you specify the SQL and you substitute the primary key value(s) returned from the NextPrimaryKeySQL command by the use of the PK tag. The option PrimaryKeyRegExp has specified that you are using the tag PK for this substitution.

Because you are expecting the NextPrimaryKeySQL statement to return possibly more than one key value, you use the IN clause on the SelectSQL SQL statement. This also requires that you show this by setting the UsingInClauses option.

Because the primary key(s) returned by NextPrimaryKeySQL is/are of type string you must specify the QuoteMultipleKeys option. This is only needed for string keys and it just specifies that you want individual keys within the IN clause to be correctly quoted.

Notice that the `SelectSQL` statement selects the records for the given PK's, but it selects them ordered by `GeneratedDate`. This is essential to retrieve the records in the order they were written to the trigger table.

So, the sequence of events is as follows:

- The `PollingSQLSource` issues the `NextPrimaryKeySQL` statement and this returns a series of one or more key values identifying the records to be processed on this poll

- The `SelectSQL` statement then issue its SQL statement substituting in these key values (returned from the `NextPrimaryKeySQL` statement)

  The `SelectSQL` uses the `order by` clause to ensure that it retrieve the records in the order they actually occurred.

- The `PollingSQLSource` then processes these records

- Once the records have been processed, the `CommitSQL` statement is invoked for all the primary keys that were processed in this poll

- However, if an error has been returned, the adaptor simply shuts down, leaving the current record untouched.

And your network of adaptors (your Event Handler) now looks as follows:

OVBPI
Modeller

Deploy

OVBPI Business
Impact Engine

Message
Hospital

Deploy

Run-Time
Event
Repository

BIAEngineSink

BIAEngineAdaptor

SocketMTSource

SocketSink

AdornmentPipe

FlatFileAdaptor

AliasingPipeSegment

IndirectionPipe

FilePollSource

customers.log

SocketSink

PollingSQLSource

DBAdaptor

Orders

BIA_BUF__orders
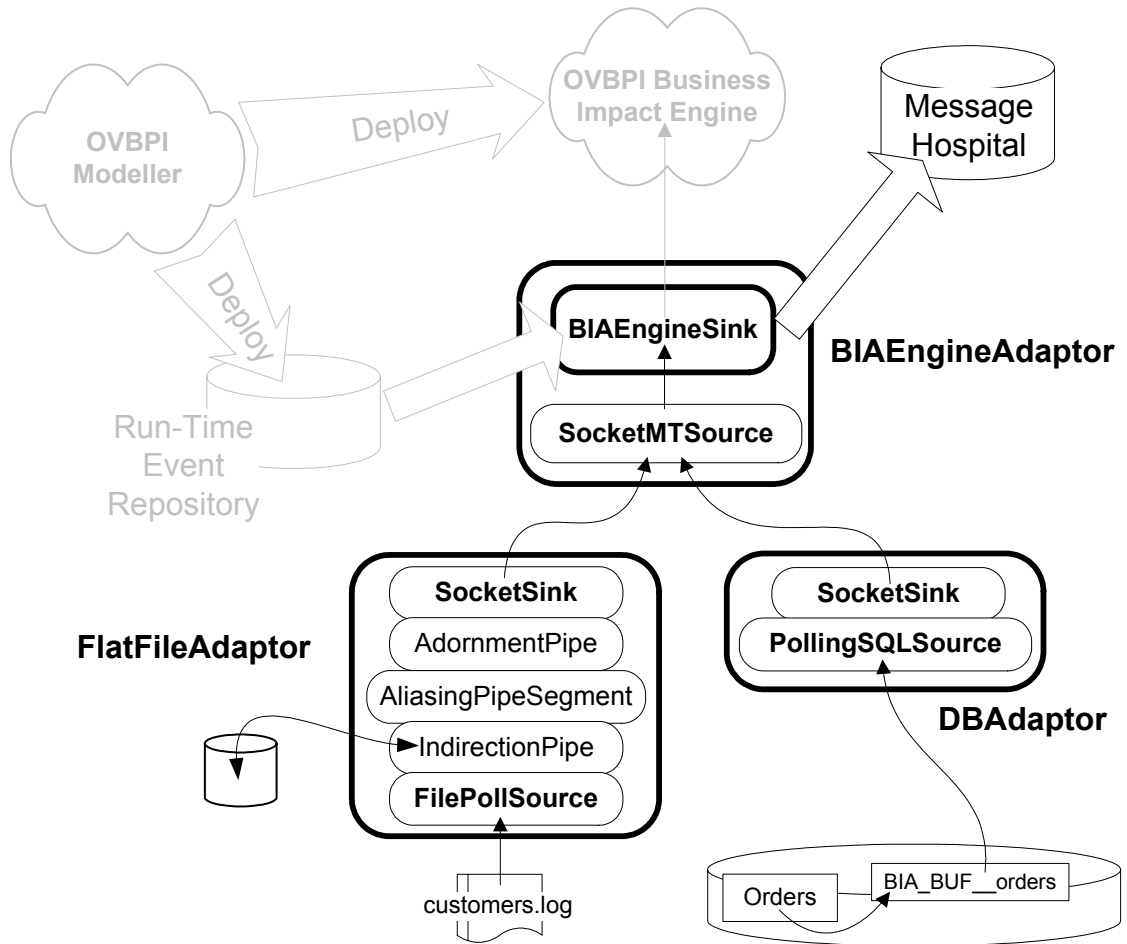
BatchSize and Limiting the Records Selected

In the above examples, you see the use of the `BatchSize` configuration option to limit the number of records retrieved with each poll.

For example:

```
B.DB.BatchSize = 100
B.DB.NextPrimaryKeySQL = select TheUID from BIA_BUF__myOrders
                                 where EventStatus = 'NEW'
                                 order by GeneratedDate
```

This `BatchSize` option means that the `NextPrimaryKeySQL` select command returns all the keys for records with an `EventStatus` of `NEW`. The adaptor then only processes the first 100 of these keys. This could be a bit inefficient if your adaptor is finding thousands of records awaiting processing at each poll.

Alternatively, you could specify the limit within the actual SQL command(s) themselves.

An example you see used within the openadaptor manual is where the SQL command sets a `rowcount` before issuing the select statement. For example:

```
(no BatchSize option is set)

B.DB.NextPrimaryKeySQL = set rowcount 100 \
                         select TheUID from BIA_BUF__myOrders
                                 where EventStatus = 'NEW'
                                 order by GeneratedDate

B.DB.SelectSQL     = select * from BIA_BUF__myOrders
                         where TheUID IN ('PK')
                             order by GeneratedDate
...etc...
```

This seems to work for MSSQL data tables, but not for Oracle data tables.

However, you need to be **very careful** if using such a mechanism. Some database implementations apply the record limiting before applying any `order by` clause. That is, in the above example you actually want the `NextPrimaryKeySQL` select statement to select all the available keys sorted by `GeneratedDate`, and then to return only the first 100 records.

Let's illustrate this further with an Oracle example:

Suppose you had the following select statement configured:

```
B.DB.NextPrimaryKeySQL = select TheUID from BIA_BUF__myOrders
                            where EventStatus = 'NEW'
                                and rownum < 101
                                    order by GeneratedDate
```

This would select the first 100 records that had an `EventStatus` of `NEW` It would then order these 100 records by their `GeneratedDate`. This would not pick up the records in chronological order...as you had probably expected.

Instead, you should configure the SQL as follows:

```
B.DB.NextPrimaryKeySQL = select TheUID from
                            (select TheUID from BIA_BUF__myOrders
                                where EventStatus = 'NEW'
                                    order by GeneratedDate)
                        where rownum < 101
```

This would select all the `NEW` records, order them by `GeneratedDate`, and then return the first 100 of these records.

So, by all means use SQL to limit the number of records retrieved by your adaptor with each poll, but make sure that it is selecting the records the way you expect.

If you are not an SQL expert then you can always just use the `BatchSize` configuration option.

Lab - Driving the Order Flow

In this lab, you:

- Configure the business data events necessary to drive an actual OVBPI flow

- Configure the OVBPI Event Handler to maintain log of the events it receives (for debugging purposes)

- Configure and use the OVBPI Event Store

Driving the OVBPI Order Flow

The Order Flow

Assuming you have completed the *OVBPI Integration Training Guide - Modeling Flows*, you should have a flow deployed called: Order Flow.

(If you do not have this flow available then you can find the zip file order_flow.zip in the events-tg\sols directory. Run the OVBPI Modeler, import this flow definition and deploy the Order Flow.)

The Order Flow is started by the event Orders/New where this event contains the following attributes:

```
Name          Type        Constraint    Unique
----          ----        ----------    ------
OrderNumber   String      20            Yes
CustomerID    String      30
Value         Currency    GBP
```

Your mission is to set up a database adaptor to generate an Orders/New event whenever a new order is entered.

The orders are to be entered into a data table called myOrders.

You are to then set up a trigger to capture each order as it is added to this myOrders table.

Let's create the myOrders data table...

The Data Source

- Create a data table called: `myOrders`
  with the following column definitions:

  ```
  Column Name         Type
  -----------         ----
  OrderNo             character (20)
  CustID              character (15)
  Amount              numeric
  Priority            character (9)
  OrderDate           date and time
  DeliveryDate        date and time
  PaymentStatus       character (9)
  ```

  Note: You need to choose the appropriate data type based on the database
  implementation that you are using.

The Buffer Table

Before creating the database trigger you need to define the buffer table into
which the trigger writes each orders record:

- Define a data table called: `BIA_BUF__myOrders`
  with the following column definitions:

  ```
  Column Name         Type
  -----------         ----
  EventName           character (50)
  GeneratedDate       date and time
  TheUID              identifier/number (see below)
  OrderNo             character (20)
  CustID              character (30)
  Value               numeric
  EventStatus         character (50)
  ```

- Depending on the database implementation you are using, you need to
  configure the column `TheUID` such that it is assigned a unique number/
  ID with each record added.

  For MSSQL, you define `TheUID` as type `uniqueidentifier`.

  For Oracle, you can define `TheUID` as type `number`, and then create an
  Oracle sequence which your trigger can assign to each record:

  — (Oracle Only) Create a sequence as follows:

```
create sequence bia_seq__myOrders
    start with 1
    increment by 1
    maxvalue 4000
    cycle
    cache 100;
```

The Trigger

- You need to create a database trigger that performs the following:

  — The trigger is to fire for every INSERT into the myOrders table

  — The trigger is to write the following data to the BIA_BUF__myOrders table:

```
Column Name          Value
-----------          -----
EventName            The string value: Orders/New
GeneratedDate        the current date/time
TheUID               The unique ID or next sequence number
OrderNo              The OrderNo from the myOrders record
CustID               The CustID from the myOrders record
Value                The Amount from the myOrders record
EventStatus          The string value: NEW
```

Insert Test Orders

To check that your trigger is working, insert some orders into the myOrders table. You should see records being written into the BIA_BUF__myOrders table as well.

- Try the following insert:

```
INSERT INTO myOrders (ORDERNO, CUSTID, AMOUNT, PRIORITY,
                      ORDERDATE, DELIVERYDATE, PAYMENTSTATUS)
  VALUES (11, 1, 199.00, 'NORMAL', '12/25/2004', Null, 'PAID');
```

Note: The date format may vary for your database implementation. You might need to use a date format of 25-Dec-2004, or whatever is appropriate for your database environment.

You should see a corresponding record written to the BIA_BUF__myOrders table. If there is a problem, go back and fix this now.

The Adaptor

- Create an adaptor called `OrderFlowAdaptor` that polls the `BIA_BUF__myOrders` table and sends in `Orders/New` events.

- Make sure that you specify the database password using encoding. That is, do not have a plain text password in the adaptor properties file

Here are some points that might help you:

- Use the `PollingSQLSource` as the source component

- Use the `SocketSink` as the sink component

- The OVBPI Business Event Handler receives its events on port `44005`

- For this lab, limit the number of records retrieved with each poll to `three(3)`

- In the `SelectSQL` option, use SQL aliasing to make sure the event attributes are all named correctly

- Refer to Password Encoding on page 152 to help you encode the database password

- Once you have configured everything, run the adaptor and see if you are able to generate `Orders/New` events into the OVBPI Business Impact Engine

    You should be able to run the OVBPI Business Process Dashboard and see these new `Order Flow` instances!

Configuring An Event Log

Often when developing a business flow and trying to generate the correct business events, it can all get a bit confusing. People start to wonder whether the OVBPI Business Impact Engine ever received the events and/or whether the events were actually sent in the correct order.

There are various debugging techniques discussed in Chapter 10, Further Topics.

For this lab, you are to follow the instructions given in Debugging Your Adaptor Network on page 386, and configure your OVBPI Business Event Handler to log every event it receives to a flat file.

Please note that this is something you should **not** do in a production environment!!! This is very useful for debugging when in development mode...but that is the only time you should set this up.

- Configure your OVBPI Business Event Handler such that it logs every event it receives to a flat file as well as sending each event into the OVBPI Engine for processing

  Remember, any changes you make to the bia_event_engineadaptor.props file are overwritten the next time anyone use the OVBPI Administration Console to make any configuration changes whatsoever.

- Send in some new orders and see the events being logged

Configuring The Event Store

This part of the lab is **optional**.

The advantage of using an Event Store architecture (as apposed to a socket-based architecture) is that it allows the OVBPI administrator to take down the OVBPI Business Event Handler without affecting any client adaptors.

In this part of the lab, you reconfigure your adaptor network to use an Event Store architecture.

You could edit the `bia_event_engineadaptor.props` file directly, however let's instead set up an interim adaptor that reads from the Event Store and then passes the events to the standard OVBPI Business Event Handler. You then alter your `OrderFlowAdaptor` adaptor (from the first part of this lab) to send the events into the Event Store.

Your architecture looks as follows:

OVBPI Business
Impact Engine

BIAEngineAdaptor

BIAEngineSink

SocketMTSource

EventStoreAdaptor

SocketSink

BIAEventStoreSource

Event Store

OrderFlowAdaptor

BIAEventStoreSink

PollingSQLSource

Buffer
Table

Database
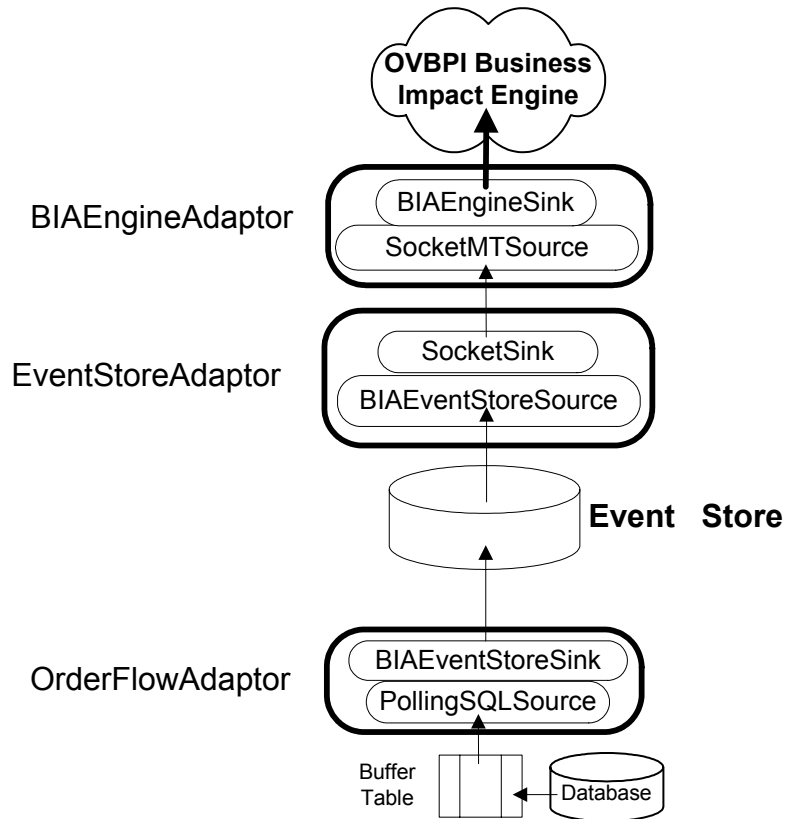
- Create a new adaptor called `EventStoreAdaptor` that reads from the Event Store and writes to the socket of the OVBPI Business Event handler

- Convert your `OrderFlowAdaptor` to send events to the Event Store

- Run the adaptor network, sending in new orders

  You are now able to turn off the `EventStoreAdaptor` and OVBPI Business Event Handler adaptors without affecting your `OrderFlowAdaptor`. You can then restart these adaptors and they simply pick up any records that have been written into the event store.

Well done! You have reached the end of the lab.

Summary

Business data events are modeled within the OVBPI Modeler and these define the events that the OVBPI Business Impact Engine accepts. Modelling these events within the Modeler however is not enough - someone has to actually configure adaptors to deliver these events from the available databases, files and applications.

Here are some key points to remember when configuring your adaptors:

- When you create or modify event definitions within the OVBPI Modeler, make sure you remember to **deploy** them - otherwise the `BIAEngineSink` does not know about these events

- The `BIAEngineSink` reads all valid event definitions from the run-time event repository. These definitions are cached for performance reasons, but all updates are effective immediately they are deployed. That is, the cache is always kept up to date

- Decide on your Event Handler architecture

  Decide whether to use the event store or socket connections. Decide whether to configure your adaptors to auto-restart in the event of failure.

- Configure your client adaptors for each source as appropriate

  — It may be that you are able to generate multiple event definitions from the one data source. This would involve the use of specifying different DO record types.

  — You may require one adaptor for each event type

  — Always set the `GeneratedDate` attribute for each event, and always set the value to be the time the event was created

  — You can use the `AdornmentPipe` to add the required event header (`EventName` and/or `GeneratedDate` attributes) if you are unable to get this information from the actual data source itself

  — The `PollingSQLSource` is available for accessing source data from databases

  — Reading data events from a database does not have to involve triggers...but it probably will

  — You can use the `IndirectionPipe` if you need to carry out additional database lookups to add/overwrite attributes to an event

- Configure your client adaptors to automatically restart if they should terminate unexpectedly

  Have your adaptor exit when it encounters an error. This allows the adaptor to send events up to OVBPI and if, for any reason, that message cannot be processed, have your client adaptor simply terminate. When your client adaptor restarts, it sends the event again.

- Defining events within the OVBPI Modeler, and creating adaptors to generate these events is an iterative process

  Usually, events are defined without any understanding of what data sources are available to fulfill these events. Your aim should be to configure your adaptors such that they deliver the required events. However, if you cannot, you need to communicate this back to the person carrying out the flow modelling and discuss what alternative events might make sense. It is an iterative process.

6 Advanced Adaptor Configuration

This chapter considers adaptor configuration in more detail, looking at examples and scenarios.

For more in-depth knowledge about configuring openadaptor components you can always refer to the openadaptor documentation and the actual code.

FileSource

Data Typing

Although the FileSource reads only character files (string data) you can configure it to convert the input fields to different data types.

The available types are:

- `String`

    This is the default type.

- `Integer`

- `Double`

- `Float`

- `Boolean`

    Any case-insensitive variant of the word `true` are converted to the boolean TRUE. Anything else is converted to boolean FALSE.

- `Date`

- `DateTime`

For example, this configuration section reads three fields and convert the string input to be of the specified data types:

```
A.C1.Type           = MyType
A.C1.NumAttributes  = 3

A.C1.AttName1 = Field1
A.C1.AttType1 = Integer

A.C1.AttName2 = Field2
A.C1.AttType2 = Double

A.C1.AttName3 = Field3
A.C1.AttType4 = Boolean
```

Handling NULL entries in files

There are two configuration options that might be of interest:

- `EmptyStringAsNull`

  This option allows you to configure the behavior of the string reader when it reads an empty field.

  For example, if the input file is as follows:

  ```
  value1,value2,,value4
  ```

  By default, the FileSource reads the third field and sets it to an empty string.

  You may want the behavior such that it sets any missing fields to a `null`. if so, set this `EmptyStringAsNull` option to true.

  For example:

  ```
  A.C1.NumAttributes  = 4

  A.C1.EmptyStringAsNull = true
  ```

- `NullString`

  This option allows you to specify that if a field in the input data consists of a special token then the FileSource should set the field value to `null`.

  For example:

  ```
  A.C1.NumAttributes  = 5

  A.C1.NullString     = MyNull
  ```

  Means that when you read the input:

  ```
  value1,MyNull,value3,,value5
  ```

  it assigns `Att2` a null value, and `Att4` defaults to an empty string.

  Obviously you can combine the `NullString` and `EmptyStringAsNull` options if you desire.

Delimited String Reader

Quotes (") Within Fields

By default the Delimited String Reader ignores the quote (") character within string fields. That is, the string

```
This is "inside quotes"
```

is read as:

```
This is inside quotes
```

To tell the Delimited String Reader not to ignore quote characters, you configure the option:

```
IgnoreQuotes = false
```

There is one bug with this option :-(It appears to not work if you have also specified the `FieldDelimiter` tag.

If you are running the OVBPI version of openadaptor then you could replace the `FieldDelimiter` tag with the tag `FieldDelimiterString` and this would then allow you to specify a delimiter (as a string) and use the `IgnoreQuotes` tag.

Date/Time Formats and openadaptor

OVBPI accept dates and times in either of two main formats:

- As string attributes matching one of four standard layouts (See Date/Time Formats for OVBPI Events on page 225 for details of these)

- As openadaptor date/time attributes

Let's consider some examples...

File Based Examples

Simple Strings

This is an example where the date/time values are held in a file as text-strings:

Suppose your data record contains the following:

```
Field1,2004-06-23T09:56:01 GMT,1066060093765
```

That is, these three fields are held in a simple file as strings.

Your adaptor configuration segment might look as follows:

```
A.C1.DOStringReader = org. ... .DelimitedStringReader

A.C1.Type           = MyType
A.C1.NumAttributes  = 3

A.C1.AttName1       = Field1
A.C1.AttName2       = Field2
A.C1.AttName3       = Field3
```

When these date fields are sent into the BIAEngineSink it interprets these strings as date/time fields and stores them correctly within the data definition within the OVBPI Business Impact Engine.

Custom Date/Time Format

What if you have a date attribute that contains the date/time as a text string that does not match one of the OVBPI acceptable formats?

The underlying `DOStringReader` component of openadaptor (which forms part of `DelimitedStringReader` and `FixedWidthStringReader`) actually allows you to specify a custom date/time format when reading data using a `FileSource`.

For example, if your data file contains records of the form:

```
Field1,23/11/2003 __ 09:13 AM GMT
```

You can configure the file reader with your date/time format using the line:

```
A.C1.DateFormat = DD/MM/YYYY __ hh:mm NN TZN
```

You then just need to specify which attribute(s) within your input file are date/time fields of this format.

Your adaptor configuration segment might look as follows:

```
A.C1.DOStringReader = org. ... .DelimitedStringReader

A.C1.DateFormat = DD/MM/YYYY __ hh:mm NN TZN

A.C1.NumAttributes  = 2
A.C1.AttName1       = Field1

A.C1.AttName2       = Field2
A.C1.AttType2       = DateTime
```

where:

- You can only specify 1 custom date format for the file

- If your date field(s) only contained dates and not times, then you could define a DateFormat that matched your dates and then specify your date attribute(s) as AttType<n> = **Date** (instead of `DateTime`)

- The format that specifies your date and time can be made up from any combination of the following (optional) elements:

    — YYYY or YY

      To specify a year in either two or four digit form.

— `MM` or `MMM`

  To specify either a two digit month or a three-letter month name.

— `DD`

  To specify a two-digit day.

— `hh`

  Specify a two-digit hour.

— `mm`

  Specify a two-digit minute.

— `ss`

  Specify a two-digit second.

— `NN`

  Specify an AM, PM option.

— `TZN`

  Specify a three-letter time zone.

For example:

  A format of: `YYYY MMM DD`

  would match a date of: `2004 Feb 23`

  (The three-letter month name must be all uppercase, or with a leading capital letter.)

  A format of: `YYYY MMM DD  hh:mm NN -- TZN`

  would match a date of: `2004 Feb 23  09:24 PM -- EST`

When you specify a date format in this way, the `DOStringReader` is able to produce dates compatible with the OVBPI Business Impact Engine. That is:

- An attribute of type `DateTime` produces a date time compatible with the format `YYYY-MM-DDThh:mm:ss TZN`

- An attribute of type `Date` produces a date compatible with the format `YYYY-MM-DD TZN`

Database Examples

When reading date/time data from a database you can either have a column contain the string representation of the date/time or, more typically, obtain the date/time directly from a column defined as a date-type within the database.

MSSQL

A column defined in MSSQL as type `datetime` (or `smalldatetime`) is correctly read as a date/time within OVBPI.

The select statement in your adaptor properties file can simply select the date field by name, and that field's date and time value is returned. For example:

```
select id, datefield, eventName, generateddate
    from table where ...
```

If this `datefield` attribute contains a date and time value then that value is retrieved and is in a format valid for passing through to the OVBPI Business Impact Engine.

Time Zone

Unfortunately, the date/time value stored in the database has no time zone information. So, when the adaptor reads the date/time value, it is assumed to be in the same time zone as the adaptor's time zone. The adaptor's time zone defaults to being the time zone of the environment running the adaptor.

You can explicitly set the adaptor's time zone by using the `TimeZone` property. For example:

- To set the adaptor's time zone to be UTC:

    `A.TimeZone = UTC`

    where `A` is the name of the adaptor.

- To set the time zone to be Melbourne, Australia:

    `A.TimeZone = Australia/Melbourne`

    where `A` is the name of the adaptor.

To retrieve the date/time in the correct time zone you need to know the time zone that the date/time value is written in, and then run the adaptor in the same time zone.

Alternatively, rather than set the time zone of the entire adaptor, you can explicitly add the time zone within the select statement itself. To do this, you use the MSSQL built-in function, `convert()`, and have the date converted into a character string in a format that is compatible with OVBPI. You can then concatenate the time zone to this string.

For example:

```
select convert(varchar, datefield, 126) + '@Europe/London' mydatefield
    from table where...
```

where:

- `convert(varchar, datefield, 126)`

  This tells the select statement to select the field called `datefield` from the table, and convert it to a string of type `varchar`, using the built-in date/time format `126`. Format `126` produces date/time strings to the `ISO8601` standard, which is the form: `yyyy-mm-ddThh:mi:ss:mmm`. This form is one of the date/time string representations understood by OVBPI.

- `+ '@Europe/London'`

  This concatenates the string `@Europe/London` to the date/time string output from the `convert()` call. This says that the time zone is `Europe/London`.

  You obviously use whatever time zone is appropriate for your date/time value.

- `mydatefield`

  This is here to assign a column name to the resultant date/time string.

For a list of valid time zone strings, refer to Valid Database Time Zones on page 268.

Oracle

A column defined in ORACLE as type `date` is able to hold a date and time value, however, the Oracle select statement requires you to also specify the format that you need when selecting the field.

Suppose you have a date field that contains the date/time value:

```
November 23rd 2005, 11:03:34
```

(It is held in whatever internal format Oracle uses to hold date/time fields.)

If you issue the select statement:

```
select datefield from table where ...
```

Oracle only returns to you the "date" part of the value! That is, you retrieve the value:

```
November 23rd 2005
```

You do not get the "time" part of the value.

TO_CHAR()

To correctly select both the date and the time part of a `date` field, you can use the `TO_CHAR()` method and specify a format to produce the date/time string in a format that is acceptable to OVBPI.

The select statement becomes:

```
select to_char(datefield, 'YYYY-MM-DD"T"HH24:MI:SS' ) from table where ...
```

This retrieves the value:

```
2005-11-23T11:03:34
```

Although this is output as a string, this string is interpreted correctly as a date/time value when it reaches the OVBPI Business Impact Engine.

CAST()

For Oracle 9 and 10, you can also use the `cast()` function to read date fields.

The select statement becomes:

```
select cast(datefield as timestamp) from table where ...
```

This creates a valid timestamp which includes both the date and the time.

Time Zone

Unfortunately, the date/time value stored in the database has no time zone information. So, when the adaptor reads the date/time value, it needs to add on the actual time zone.

How you add the time zone depends on which Oracle function call you use to read the date field.

If you use the `TO_CHAR()` function, you can add the time zone as part of the format string. For example:

- If your date/time values are in the UTC time zone, your `TO_CHAR()` format string becomes:

  `to_char(`*`datefield`*`, 'YYYY-MM-DD"T"HH24:MI:SS`**` @UTC"`**`' )`

- If your date/time values are in the time zone of `America/Phoenix`, you change the `TO_CHAR()` formatting to become:

`to_char(`*`datefield`*`, 'YYYY-MM-DD"T"HH24:MI:SS`**` @America/Phoenix"`**`' )`

If you use the `CAST()` function, you can choose to run the adaptor in the same time zone as the dates within the Oracle database you are reading. For example:

- If your date/time values are in the UTC time zone, your adaptor requires the property:

  `AdaptorName``.TimeZone=UTC`

For a list of valid time zone strings, refer to Valid Database Time Zones on page 268.

Valid Database Time Zones

When required to specify a time zone string you need to specify a time zone string that is understood by openadaptor adaptors.

The actual time zone can be any string token that represents a valid time zone. For example:

```
UTC
GMT
Europe/London
America/Phoenix
```

You can **not** specify an offset, or relative, time zone. For example you can not specify any of the following:

```
GMT+1
GMT+02:00
UTC+1
```

If you do specify an offset, or relative, time zone, as in the above examples, the adaptor reads times in the base time zone and disregards the offset.

You can also have issues where certain time zone strings do not seem to work as you would hope. The most consistent style of time zone string is the long-format time zones such as:

```
Europe/London
America/Phoenix
```

If you need to find a time zone, use an internet search engine and there are many pages that list possible time zone strings.

Inspector Sink

When initially building and testing an adaptor, it is often a good idea to output the events to a `FileSink`. This allows you to see the output of the event and you are able to check that it is as you expect.

There is another openadaptor sink that can be very useful when first building and testing your adaptors. It is the **InspectorSink**:

```
org.openadaptor.adaptor.display.InspectorSink
```

The inspector sink displays each data object in a pop-up Windows dialog. It makes it very easy to see the exact make up of the data object.

For example:

Suppose you have the following properties file:

```
A.Component1.Name    = C1
A.Component2.Name    = C2
A.C1.LinkTo1         = C2

A.C1.ClassName       = org.openadaptor.adaptor.standard.FileSource
A.C1.DOStringReader  = org.openadaptor.dostrings.DelimitedStringReader

A.C1.InputFileName   = simple.txt

A.C1.Type            = MyTestType
A.C1.NumAttributes   = 4
A.C1.FieldDelimiter  = 124
A.C1.AttName1        = Field1
A.C1.AttName2        = Field2
A.C1.AttName3        = Field3
A.C1.AttName4        = Field4
```

A.C2.ClassName   = org.openadaptor.adaptor.display.InspectorSink

This adaptor is simply reading a file (`simple.txt`) and sending the output to the `InspectorSink`.

For each output record you see a pop-up window like the following:

Figure 31 Output from Inspector Sink
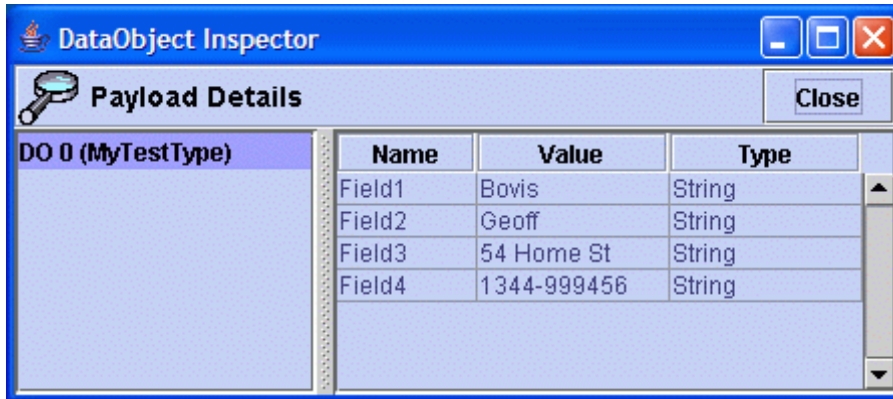
By default, the InspectorSink displays the current output record and waits for you to click Close to close down the pop-up window.

The InspectorSink can be very useful when you have events containing multiple data objects, as you are able to easily see the structure of the result.

Delaying Your Adaptor (MaxCallsPerPoll)

Let's consider the `FilePollSource` component. When the `FilePollSource` component is reading its input, it reads and processes the first record, then the next, and so on through the input file. It is only when it reaches the end-of-file (EOF) that it breaks out of its processing loop, pauses and then periodically polls the file for more data. When new data records appear, these are all processed before the source pauses again.

The `PollingSQLSource` shows a similar behavior. The `PollingSQLSource` runs a loop that issues the SQL to get the next set of primary keys (`NextPrimaryKeySQL`) and, so long as that returns some values, the `SelectSQL` statement is then issued to get and process the actual data records. Once these data records have been processed, the `PollingSQLSource` loops back and issues the `NextPrimaryKeySQL` statement. If this returns values, then the `SelectSQL` is issued and these records are processed. So, all available SQL data records are processed in one continuous loop. Only when the `NextPrimaryKeySQL` call returns no more primary key values does the `PollingSQLSource` pause and poll for more records to appear.

This "pause and poll for new data records" is specified by the `PollPeriod` property. This is where you can specify the time (in milliseconds) that a source pauses before checking for new data. The default poll period is 1000 (1 second).

What if you wanted to configure your source component to process the next batch of records, and then pause for the specified `PollPeriod`. That is, process the next batch of input records, and then regardless of how many more records are ready and waiting to be processed, your source component pauses.

You can configure this by using the `MaxCallsPerPoll` property. This property tells the source how many times to loop through processing available data records before breaking out and pausing. The `MaxCallsPerPoll` property is actually a property of **every source component**.

So, for example, if your `PollingSQLSource` adapter is configured as follows:

```
B.DB.ClassName         = org.openadaptor.adaptor.jdbc.PollingSQLSource

B.DB.PollPeriod        = 5000
B.DB.MaxCallsPerPoll   = 1
B.DB.BatchSize         = 10
B.DB.NextPrimaryKeySQL = select TheUID from buf1
                             where EventStatus = 'NEW' order by GeneratedDate

B.DB.SelectSQL    = select EventName, datefield GeneratedDate,
                        OrderNo, Value from buf1
                            where TheUID IN ('PK') order by GeneratedDate
...etc...
```

Your adaptor issues the select statement and processes the first batch of 10 records, then pauses for five seconds (5000 msecs), then issues the select to process the next batch of 10 records.

The `MaxCallsPerPoll` property is actually part of every source component, so you can apply a similar idea to a `FileSource`, as follows:

```
A.C1.ClassName         = org.openadaptor.adaptor.standard.FileSource

A.C1.InputFileName     = simple.txt

A.C1.PollPeriod        = 10000
A.C1.MaxCallsPerPoll   = 1
```

This FileSource now reads and processes the next record from the file, and then pauses for 10 seconds (10000 msecs) before reading and processing the next record.

Transforming Attributes

Have you ever wanted to be able to:

- Strip out characters from an input attribute?

- Concatenate some attributes together?

- Do some mathematical calculations on the attributes within the input record?

- Parse the input line, match certain patterns and produce new attributes?

The openadaptor pipe:

```
org.openadaptor.adaptor.value.TransformerPipe
```

is the pipe for you.

Stripping Characters

You can configure the `TransformerPipe` to apply a regular expression to a source attribute and produce a destination attribute. For example, you could use the `TransformerPipe` to strip off any leading, or trailing, brackets that might surround a data value.

When specifying the regular expression you use round-brackets `()` to mark the section that you want to pull out of the source attribute. You must also double-escape any special character within the regular expression with two back-slashes `\\`.

Let's look at some examples...

Stripping Leading and Trailing Braces {}

```
A.TRANS.ClassName     = org.openadaptor.adaptor.value.TransformerPipe
A.TRANS.Type1         = MyRecord

A.TRANS.MyRecord.Action1.Name        = Extract
A.TRANS.MyRecord.Action1.RegExp      = \\{(.*)\\}
A.TRANS.MyRecord.Action1.Source      = Att1
A.TRANS.MyRecord.Action1.Destination = Att1
```

This transformation is only applied to records of type `MyRecord`.

The action name `Extract` means that the specified regular expression is applied to the `Source` attribute, and the result is stored in the `Destination` attribute.

The `Destination` attribute can an attribute different from the `Source` attribute. However, the `Destination` attribute **must already exist** within the data object. The TransformerPipe is unable to create new attribute definitions. You can use the `AdornmentPipe` (earlier in the adaptor pipeline) to pre-create data attributes that can then be written into by the `TransformerPipe`.

If the `Destination` attribute is the same as the `Source`, then the source attribute value is overwrites by the result of the regular expression.

If the value within the `Source` attribute does not match the regular expression, then the value is written into the `Destination` attribute without change. That is, the source attribute value is passed through untouched.

The regular expression `\\{(.*)\\}` is explained as follows:

- `\\{`

  The double back-slashes are used to escape the `{` character. This simply means that the regular expression is saying that there must be a `{` character at the start of the source attribute to match this regular expression.

- `(.*)`

  The round-brackets are used to mark the part of the source data that you wish to pull-out and save into the destination attribute.

  The `.*` is used is specify none-or-more-characters.

- `\\}`

  The double back-slashes are escaping the `{` character. This is saying that the source attribute value must end in a `}` character for it to match this regular expression.

So, `\\{(.*)\\}` is saying: If the source attribute value starts with a `{` and ends with a `}`, then pull out the part in between.

Stripping Leading and Trailing Braces {} or Brackets []

```
A.TRANS.ClassName      = org.openadaptor.adaptor.value.TransformerPipe
A.TRANS.Type1          = MyRecord

A.TRANS.MyRecord.Action1.Name        = Extract
A.TRANS.MyRecord.Action1.RegExp      = [\\{\\[](.*)[\\}\\]]
A.TRANS.MyRecord.Action1.Source      = Att1
A.TRANS.MyRecord.Action1.Destination = Att1
```

This transformation is only applied to records of type `MyRecord`.

This applies the regular expression against the source attribute `Att1`, and overwrites the result output back into the attribute `Att1`.

The regular expression is as follows:

- `[\\{\\[]`

  Square brackets are used to specify groupings. So for example, `[23]` would match either the character `2` or `3`.

  This regular expression is saying: "Match either a `{` or a `[` character."

  The `\\{` is just how you escape the `{` character.

- `(.*)`

  This just specified that part of the string you wish to output.

- `[\\}\\]]`

  This says: "Match either a `}` or `]` character."

So, `[\\{\\[](.*)[\\}\\]]` says: "If the input starts with either a `{` or `[`, and ends with either a `}` or a `]`, then pull out the part in between."

Concatenate Strings

You can concatenate two attributes together with the `Concatenate` action.

Using Attributes

```
A.TRANS.ClassName       = org.openadaptor.adaptor.value.TransformerPipe
A.TRANS.Type1           = MyRecord

A.TRANS.MyRecord.Action1.Name        = Concatenate
A.TRANS.MyRecord.Action1.Source1     = Att1
A.TRANS.MyRecord.Action1.Source2     = Att2
A.TRANS.MyRecord.Action1.Destination = StringResult
```

This concatenates the value of `Att2` to the end of the value of `Att1` and saves the result in the attribute called `StringResult`.

Using Constants

You can concatenate string constants to attribute values.

For example:

```
A.TRANS.ClassName       = org.openadaptor.adaptor.value.TransformerPipe
A.TRANS.Type1           = MyRecord

A.TRANS.MyRecord.Action1.Name        = Concatenate
A.TRANS.MyRecord.Action1.Source1     = {--Pre--}
A.TRANS.MyRecord.Action1.Source2     = Att1
A.TRANS.MyRecord.Action1.Destination = Att1

A.TRANS.MyRecord.Action2.Name        = Concatenate
A.TRANS.MyRecord.Action2.Source1     = Att1
A.TRANS.MyRecord.Action2.Source2     = {--Post--}
A.TRANS.MyRecord.Action2.Destination = Att1
```

This set of actions prefixes the value of `Att1` with the string `--Pre--`, and then suffixes that result with the string `--Post--`.

Notice that you specify constants within `{}` brackets. This allows the pipe to detect the difference between an attribute name and a constant.

Mathematical Operations

You can use the `TransformerPipe` to carry out mathematical operations on attributes.

Valid operations are: + - * /

You can provide constants, for either Source, by specifying them within `{}` brackets.

Adding Two Attribute Values

```
A.TRANS.ClassName      = org.openadaptor.adaptor.value.TransformerPipe
A.TRANS.Type1          = MyRecord

A.TRANS.MyRecord.Action1.Name        = Maths
A.TRANS.MyRecord.Action1.Operator    = +
A.TRANS.MyRecord.Action1.Source1     = Att3
A.TRANS.MyRecord.Action1.Source2     = Att4
A.TRANS.MyRecord.Action1.Destination = Result
```

This example adds the values contained within `Att3` and `Att4` and saves the result in the attribute `Result`.

You can overwrite an attribute with the result if you wish. If you are going to save the value into a different attribute, that attribute must already exist within the data object. You can use the `AdornmentPipe` to pre-create any additional attributes you need.

Both attribute values used in a mathematical calculation must contain only numeric data. You can use string fields so long as they only contain numbers.

Multiple Operations

```
A.TRANS.ClassName      = org.openadaptor.adaptor.value.TransformerPipe
A.TRANS.Type1          = MyRecord

A.TRANS.MyRecord.Action1.Name        = Maths
A.TRANS.MyRecord.Action1.Operator    = +
A.TRANS.MyRecord.Action1.Source1     = Att3
A.TRANS.MyRecord.Action1.Source2     = {50}
A.TRANS.MyRecord.Action1.Destination = Result

A.TRANS.MyRecord.Action2.Name        = Maths
A.TRANS.MyRecord.Action2.Operator    = *
```

```
A.TRANS.MyRecord.Action2.Source1      = Result
A.TRANS.MyRecord.Action2.Source2      = Att4
A.TRANS.MyRecord.Action2.Destination = Result

A.TRANS.MyRecord.Action3.Name         = Maths
A.TRANS.MyRecord.Action3.Operator     = /
A.TRANS.MyRecord.Action3.Source1      = Result
A.TRANS.MyRecord.Action3.Source2      = Att5
A.TRANS.MyRecord.Action3.Destination = Result
```

This example does the following calculations:

```
Result = Att3 + 50
Result = Result * Att4
Result = Result / Att5
```

The `Result` attribute contains the final value after all these calculations.

Decimal Precision

You may wish to alter the precision of your result after carrying out a series of calculations. Or, you may just wish to alter the precision of an attribute value you are reading from a file.

For example:

```
A.TRANS.MyRecord.Action1.Name         = Maths
A.TRANS.MyRecord.Action1.Destination = Result
A.TRANS.MyRecord.Action1.Precision    = 2
```

This takes the value within the `Result` attribute and alters its precision to two decimal places.

XML Format Data

To configure your openadaptor adaptor to read XML data records (called XML documents) from a file, you need to set the `DOStringReader` property, as follows:

```
A.C1.DOStringReader = org.openadaptor.dostrings.XMLStreamReader
```

where:

- The adaptor name in this example is: `A`
- The Source component in this example is: `C1`

But how do you configure the record layout?

The `XMLStreamReader` reads XML documents from the input file. Each XML document is self-describing, and hence, the record layout is constructed from the tags and values within the XML document. However, it is important to understand how the XML document is converted into an openadaptor data object (DO) array.

The XML Data Object (DO)

The `XMLStreamReader` reads through the XML document and constructs an openadaptor data object (DO) to hold the input data.

As you might expect, the name of each XML tag becomes the name of that attribute within the record.

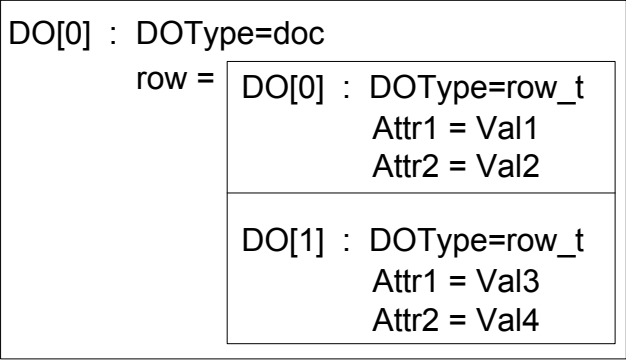But, how does it handle the hierarchy of the XML document?

Consider the following example XML document:

```
<?xml...>
<doc>
  <row>
     <attr1>val1</attr1>
     <attr2>val2</attr2>
  </row>
  <row>
     <attr1>val3</attr1>
     <attr2>val4</attr2>
  </row>
</doc>
```

When the `XMLStreamReader` processes this XML document, it constructs the following openadaptor DO tree:

Figure 32 Resultant Data Object from XML

where:

- The whole document is placed inside a single data object.

   This data object is typed as the name of the overall document tag (doc). This "type" is the DOType.

   This data object contains one attribute. This attribute is assigned the name of the XML tag. In this example, the attribute is called `row`

- The `row` attribute contains an array of data objects.

   There is a data object for each of the "rows" defined within the XML document.

   Each `row` data object has its DOType set to the name of the XML tag, with "_t" appended. So the DOType is row_t.

For the top level tag (doc) the DOType is simply the name of the tag, but the nested tag (row) has a "_t" appended to its name to produce the DOType. Why does the nested tag have a "_t" appended? It just does.

- All simple attribute values are treated as strings.

  So in this example, Val1, Val2, Val3 and Val4 are typed as String values.

So, in this example, after the XMLStreamReader has read the XML document, you end up with a single attribute (called: row), with its DOType set to doc.

Unfortunately, this is not very useful within your adaptor. For example, if your adaptor is needing to send the attributes Attr1 and Attr2 to the sink, then sending them all wrapped up within another attribute is not going to work.

So, the XMLStreamReader has an option where you can specify that you want the outer tag to be removed from the resultant DO output.

The property is DiscardRootObjectTag, and if you set this to true, it discards the root tag.

So, in this example, if you configure the XMLStreamReader as follows:

```
A.C1.DOStringReader       = org.openadaptor.dostrings.XMLStreamReader
A.C1.DiscardRootObjectTag = true
```

you end up with the following openadaptor DO:

Figure 33 Data Object without Root Tag

```
DO[0] : DOType=row_t
          Attr1 = Val1
          Attr2 = Val2

DO[1] : DOType=row_t
          Attr1 = Val3
          Attr2 = Val4
```

where:

- You have two data objects, each of type row_t.

- You are able to access the attributes within each of these data objects directly by name in the normal way.

XML Document Filter

There is one other property worth setting when using the `XMLStreamReader` and that is:

```
A.C1.UseXMLDocumentFilter = true
```

Setting this option does not seem to alter the behavior of the `XMLStreamReader` in any obvious way, however, if you do not set this property you see a warning message whenever you run your adaptor. The warning message is as follows:

```
WARN: Using old, inflexible XML Document Splitter
     (UseXMLDocumentFilter=false)
```

So, it probably makes sense to always set this property to `true` and avoid the warning message.

Example Configuration

To read XML from a file:

```
A.Component1.Name         = C1
A.Component2.Name         = C2
A.C1.LinkTo1              = C2
A.Logging.LogSetting1     = INFO

A.C1.ClassName            = org.openadaptor.adaptor.standard.FileSource

A.C1.InputFileName        = inputFile.xml

A.C1.DOStringReader       = org.openadaptor.dostrings.XMLStreamReader
A.C1.DiscardRootObjectTag = true
A.C1.UseXMLDocumentFilter = true
```

where:

- The adaptor is called: A

- The Source component is called: C1

- The outer (root) XML document tag is discarded.

- The resultant record(s) all have the same DOType. The name of this DOType is the name of the second-level XML tag, with "_t" appended.

- All attribute values within each record are be treated as strings.

So, if you have an XML document as follows:

```
<?xml...>
<rootTag>
  <tag2>
    <attr1>val1</attr1>
    <attr2>val2</attr2>
  </tag2>
  <tag2>
    <attr1>val3</attr1>
    <attr2>val4</attr2>
  </tag2>
</rootTag>
```

The resultant DO is:

Figure 34 Resultant DO

```
DO[0] : DOType=tag2_t
          Attr1 = Val1 (String)
          Attr2 = Val2 (String)

DO[1] : DOType=tag2_t
          Attr1 = Val3 (String)
          Attr2 = Val4 (String)
```

Valid XML

Although the `XMLStreamReader` is able to read generic XML, you need to think carefully about the DO it produces. In some cases the resultant DO may have nested DOs to a level that is cumbersome to work with, and in some cases the `XMLStreamReader` may not even be able to build a DO.

DiscardRootObjectTag and XML Format

The `DiscardRootObjectTag` option only works if the XML below the root tag consists of another single tag, or sets of the same tag.

In the example XML:

```
<?xml...>
<doc>
  <row>
    <attr1>val1</attr1>
    <attr2>val2</attr2>
  </row>
  <row>
    <attr1>val3</attr1>
    <attr2>val4</attr2>
  </row>
</doc>
```

the `DiscardRootObjectTag` option works because, once the `doc` tag is removed, you are left with sets of `row` tags. In other words, the remaining XML can be thought of as a series of the same tag (`row`).

If, however, you have XML like this:

```
<?xml...>
<rootTag>
  <level2>
    <attr1>val1</attr1>
    <attr2>val2</attr2>
  </level2>
  <another>
    <attr1>val3</attr1>
    <attr2>val4</attr2>
  </another>
</rootTag>
```

Once the `rootTag` is discarded, the XML has two next-level tags, `level2` and `another`, and this causes the `XMLStreamReader` **to fail**.

`XMLStreamReader` fails with the error message:

```
FATAL: Error on sourcePoll for [C1],AbstractReader C1 Failed to read
 DataObject from stream: DiscardRootObjectTag=true but document does
  not have a single root tag/attribute
```

Nested XML

If you have XML that is deeply nested, then you have to be careful in the way you access the attributes within your openadaptor adaptor configuration file.

For example, if you consider the following XML:

```
<?xml...>
<doc>
  <rowspec>
    <row>
      <attr1>val1</attr1>
      <attr2>val2</attr2>
    </row>
  </rowspec>
  <rowspec>
    <row>
      <attr1>val3</attr1>
      <attr2>val4</attr2>
    </row>
  </rowspec>
</doc>
```

With `DiscardRootObjectTag` set, the `XMLStreamReader` discards the `doc` tag, and create two DOs called `rowspec` (of DOType `rowspec_t`). Each of these DOs contain a `row` DO (DOType `row_t`) that contain the actual `attr1` and `attr2` values.

Your pipes are able to access the attributes `attr1` and `attr2`, however you need to refer to them using their full hierarchical name.

For example, if you wanted to pull out the attributes, `attr1` and `attr2`, and build them into a one dimensional DO, you could use the `AliasingPipeSegment` as follows:

```
A.ALP.ClassName      = org.openadaptor.adaptor.standard.AliasingPipeSegment

A.ALP.Type1          = rowspec_t AnOrder
A.ALP.Type1.Alias1 = row.attr1 OrderNumber
A.ALP.Type1.Alias2 = row.attr2 CustomerID
```

where:

- Each incoming record is of type `rowspec_t`.

- Each attribute is referenced by specifying its full hierarchical name.

- This builds a new DO, of DOType: `AnOrder`, for each record, containing the attributes `OrderNumber` (set to the value that was in `row.attr1`) and `CustomerID` (set to the value that was in `row.attr2`).

However, if you have the following XML document:

```xml
<?xml version="1.0" encoding="ISO-8859-1" ?>
<doc>
  <rowspec>
   <row>
     <attr1>val1</attr1>
     <attr2>val2</attr2>
   </row>
   <row>
     <attr1>val3</attr1>
     <attr2>val4</attr2>
   </row>
  </rowspec>
</doc>
```

and your adaptor configuration includes the same lines:

```
A.ALP.ClassName      = org.openadaptor.adaptor.standard.AliasingPipeSegment

A.ALP.Type1          = rowspec_t AnOrder
A.ALP.Type1.Alias1 = row.attr1 OrderNumber
A.ALP.Type1.Alias2 = row.attr2 CustomerID
```

you would only send the entry for `attr1=val1`/`attr2=val2`.

This is due to the fact that, within openadaptor, referring to an attribute using the full hierarchical name (for example: `row.attr1`) is unable to handle multiple occurrences. So you would skip over the entry for `attr1=val3/attr2=val4`.

Multiple Record Layouts

Suppose you have a situation where you have a 3rd party application writing XML format files into a directory. You need to process these files as they appear and you need to add in the `EventName` attribute to each record so that they can be sent correctly to the OVBPI Business Impact Engine as Business Data Events.

The only issue is that the XML files are not all of the same format. There is a mixture of (in this example) two types of records:

- Order records
- Customer records

How can you handle this?

The `FileRollSource` is a source component that allows you to point at a directory and process files as they appear. And as the input files are XML format files, you specify to use the `XMLStreamReader`.

The `XMLStreamReader` is able to construct each record layout according to the XML document it reads. So, unlike CDF files, you do not specify the record layout within the adaptor configuration file.

But how can you detect the different record layouts and add the appropriate `EventName` for each record? Let's consider some options available to you:

Filtering Based on the DOType

As described in The XML Data Object (DO) on page 279, when the `XMLStreamReader` builds the DO, it assigns the DOType for each record based on the surrounding XML tag.

If the different XML input documents (in this example: `Order` and `Customer`) have different XML tags for their data records, then the standard `XMLStreamReader` is able to assign different DOTypes to the `Order` and `Customer` records.

For example, if the `Order` documents are written as follows:

```
<?xml...>
<Orders>
  <Order>
     <attr1>val1</attr1>
     <attr2>val2</attr2>
  </Order>
  <Order>
     <attr1>val3</attr1>
     <attr2>val4</attr2>
  </Order>
</Orders>
```

and the `Customer` documents are written as follows:

```
<?xml...>
<rootTag>
  <Customer>
     <attr1>val1</attr1>
     <attr2>val2</attr2>
  </Customer>
  <Customer>
     <attr1>val3</attr1>
     <attr2>val4</attr2>
  </Customer>
</rootTag>
```

This allows you to configure your openadaptor adaptor to do the normal DOType-based filtering.

Here is an example adaptor configuration file segment that configures the `AdornmentPipe` to detect the two different DOTypes and append the appropriate event name attribute:

```
A.AP.ClassName            = org.openadaptor.adaptor.standard.AdornmentPipe

A.AP.AdornType1           = Order_t
A.AP.AdornType1.AttName1  = EventName
A.AP.AdornType1.AttValue1 = Order/New

A.AP.AdornType2           = Customer_t
A.AP.AdornType2.AttName1  = EventName
A.AP.AdornType2.AttValue1 = Customer/Update
```

Filtering Based on an Attribute Value

If your XML input documents have basically the same high-level document tags, then you are unable to do the standard DOType filtering and selection. In this case, you would need to look at filtering the input records by values within the data itself.

For example, if the input records had an attribute whose value could distinguish the type of input record, you could filter the records by this field. You could then use the `AdornmentPipe` to add the appropriate `EventName` attribute value.

You need to set up a filter path for each of the event types you are expecting.

The adaptor architecture looks something like this:

Figure 35 Attribute Filtering When DOType is the Same
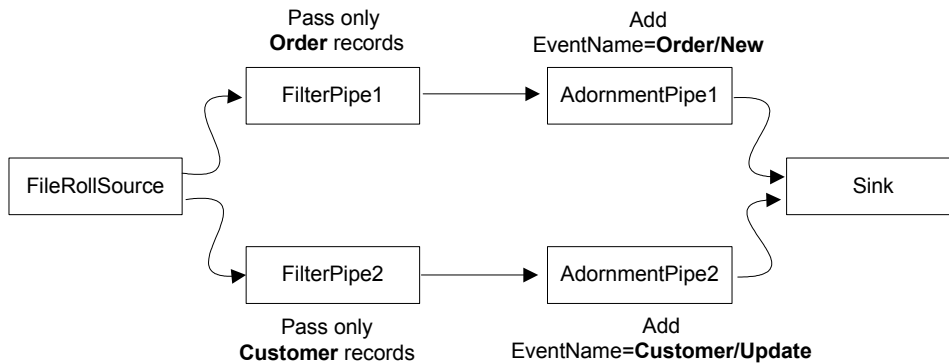
where:

- The one `FileRollSource` component reads the files from the input directory.

  Each record has the same DOType even if the actual data inside the XML document is different.

- The `FileRollSource` passes each record to **both** filters.

  `FilterPipe1` passes only those records where a particular attribute indicates that this record is an order record.

  `FilterPipe2` passes only those records where a particular attribute indicates that this record is a customer record.

- Each filter then passes the record to the appropriate `AdornmentPipe` to have the OVBPI event name appended.

- Both Adornment pipes then pass the completed record to the same sink.

- The sink then sends the record to OVBPI.

Synchronized Database Polling

Suppose you need to feed business data events from a database to drive a flow. The data comes from two data tables within the one database.

Let's draw it like this:

Figure 36 Database Adaptors
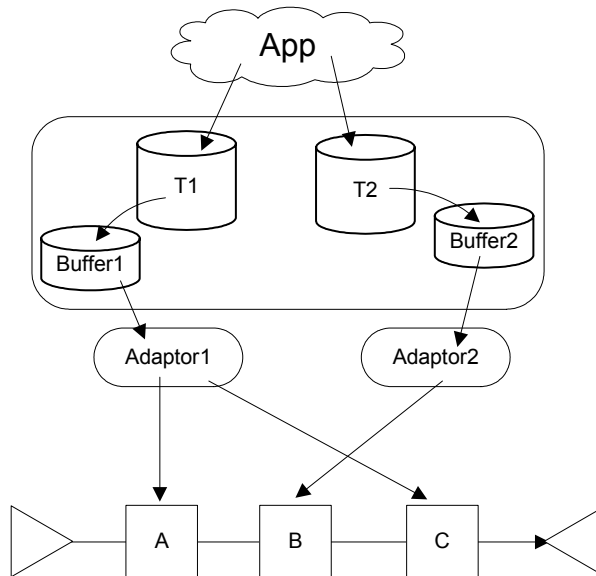
where:

- The customer's application software is doing whatever it does, and is writing records into the data tables T1 and T2.

- You set up triggers on the two data tables (T1 and T2) to write out to their respective buffer tables (Buffer1 and Buffer2)

- You set up an adaptor (Adaptor1) to monitor Buffer1, and another adaptor (Adaptor2) to monitor Buffer2.

- The events that come from Adaptor1 provide the data used in the progression rules for node A and C.

- The events that come from Adaptor2 provide the data used in the progression rules for node B.

You then set this running and everything appears to work fine. However, when you are watching the flow instances, you notice that sometimes a flow instance goes from node A to C, and at some later point then go to B. If you then look at the time line drawing for that flow instance it shows that all the nodes happened in the correct order of A, B. C. At this point you are confused? What's more, it looks very odd when you are demonstrating the dashboard to your customer as events are appearing to come in out of order...but then saying that they didn't.

It all has to do with the fact that you have multiple adaptors.

Using the `PollingSQLSource`, each adaptor is basically polling the database every-so-often. Suppose you set both adaptors to poll their respective table at 10 minute intervals. Now the customer's application software is running in real time and updating tables T1 and T2 in the correct order. Your database triggers are correctly being called, and are writing the data into the buffers tables correctly. Indeed, you have set you triggers up so that they write the actual time that they occurred into each buffer record - in the `GeneratedDate` column. So the records are being written into the respective buffer tables in the correct order and with the correct timestamps.

However, your adaptors are polling. So you could get the following sequence of events:

1. `Adaptor1` polls `Buffer1`

   Nothing there, so go to sleep for 10 minutes

2. `Adaptor2` polls `Buffer2`

   Nothing there, so go to sleep for 10 minutes

3. A new record is written to T2

   This causes a new record to be written into `Buffer2`.

4. A new record is then written to T1

   This causes a new record to be written into `Buffer1`.

5. `Adaptor1` polls `Buffer1`

   This picks up the new record that sends the flow instance into node C.

   At this point your OVBPI Dashboard shows that the flow instance has skipped node B and gone straight to node C.

6. `Adaptor2` polls `Buffer2`

This picks up the new record that sends the flow instance back into node B.

Because the GeneratedDate attribute was correctly set for each of these events, the OVBPI Business Impact Engine is able to record that fact that Node B was entered before Node C. This allows the time lime diagram to show the events happening in the correct order.

So even with a relatively simple set of database adaptors, it is possible to face issues where the events can arrive at the OVBPI Business Impact Engine out of sequence, and this may cause some issues.

Some might suggest that by setting smaller polling periods you can solve the problem. But that can put unnecessary load on your system and there is still a potential window where events may get picked up out of sequence.

The solution is described in the next section, Single Buffer Table and Adaptor on page 294

Single Buffer Table and Adaptor

A way to resolve any out-of-sequence issues is to have all the events coming from the database through one single adaptor. This can be achieved by configuring the following:

Figure 37 Single Buffer Table and Adaptor
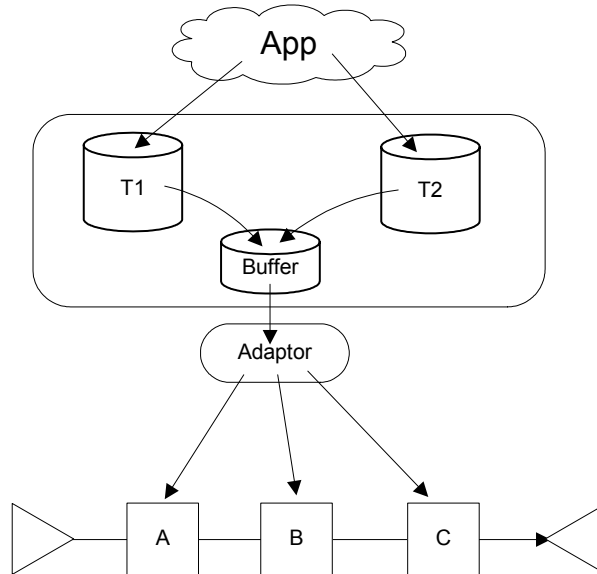
where:

- You define one buffer table

    This buffer table defines a record layout that contains all the necessary attributes from table T1 as well as the necessary attributes from table T2.

- Both triggers write into the same buffer table

    Each trigger simply writes out the attributes as it would normally. The database system writes nulls into any unspecified attributes.

- Your adaptor simply sends in each record from the buffer table - sending in all attributes

    Because each trigger writes the event name with each record, your adaptor can send the whole record to OVBPI. The Business Impact Engine looks at the EventName attribute first, and based on that, pulls out the attributes it needs. Any additional attributes are simply ignored.

Example Configuration

Suppose you are putting triggers on two data tables - `Orders` and `Customers`.

The `Order` attributes that you need to write out to the buffer table are:

```
OrderNo
CustId
OrderValue
```

The `Customer` attributes you need to write out to the buffer table are:

```
CustId
CustName
CustAddress
```

An example buffer table definition is as follows:

```
CREATE TABLE BIA_BUF__sync (

    EventName     varchar (50) NOT NULL,
    GeneratedDate datetime      NOT NULL,

    TheUID        uniqueidentifier,

    OrderNo       varchar (20),
    CustID        varchar (30),
    OrderValue    money,

    CustName      varchar (50),
    CustAddress   varchar (150),

    EventStatus   varchar (50)
);
```

where:

- The `EventName` attribute identifies each record

- The attributes `GeneratedDate`, `TheUID`, and `EventStatus` are standard attributes for a buffer table

- You then have the fields for holding the order information and for holding the customer information

- The `Order` trigger writes its records containing the attributes:

  ```
  EventName, GeneratedDate, TheUID,
  OrderNo, CustID, OrderValue, EventStatus
  ```

- The `Customer` trigger writes its records containing the attributes:

  ```
  EventName, GeneratedDate, TheUID,
  CustID, CustName, CustAddress, EventStatus
  ```

The adaptor SQL simply polls the buffer table sending in the entire record, as follows:

```
B.DB.SelectSQL   = select EventName,
                     GeneratedDate, \
                     OrderNo, \
                     CustID, \
                     OrderValue, \
                     CustName,
                     CustAddress \
               from BIA_BUF__sync  \
               where TheUID IN ('PK') order by GeneratedDate
```

Because each record in the buffer table contains the name of the event in the `EventName` attribute, the OVBPI Business Impact Engine is able to process each event.

Character Sets and Text Encoding

There is an issue with openadaptor sinks when it comes to character sets. The issue is that openadaptor sink components seem to default their character set to ISO-8859-1. That is, regardless of your system's default character set, when you send data from an openadaptor sink, the data is encoded using ISO-8859-1.

Now, for many English and American customer sites, using the ISO-8858-1 character set may be fine. Indeed, for a number of European sites this might be fine also, as the ISO-8859-1 character set does include some 8-bit characters. However, how do you handle full 8-bit and 16-bit characters?

TextEncoding

The `TextEncoding` property allows you to set the text encoding used by each component (source, pipe, and/or sink) within your adaptor.

You usually do not need to set the `TextEncoding` property for the source or pipe components.

An openadaptor source component is able to read input characters using the system default character set. For example, if your adaptor is running on a Japanese machine, where the operating system default character set is Shift-JIS (SJIS), you do not have to set the `TextEncoding` for the source component. The source component is able to read the data using the system default SJIS character set. You could explicitly set the `TextEncoding` to be `SJIS`, but as SJIS is the default for this system, you do not need to.

The problem is when your adaptor comes to pass this information out through the sink. The sink unfortunately defaults to ISO-8859-1.

To enable the sink to pass your 8/16-bit data through in a more appropriate character set, you use the `TextEncoding` property. For example:

```
A.C2.ClassName       = org.openadaptor.adaptor.standard.FileSink
A.C2.TextEncoding    = SJIS
```

This tells the `FileSink` to encode the data using the SJIS character set instead of the default ISO-8859-1.

Adaptor Network

Configuring a standalone adaptor to send your data encoded in (for example) SJIS is fine, however, you are typically configuring adaptors to send their data to a central OVBPI Event Handler. The OVBPI Event Handler may well be running on a different machine which may even be in a different country and running with a different default character set.

Suppose you have an adaptor network as follows:

Figure 38 Mixed Character Sets
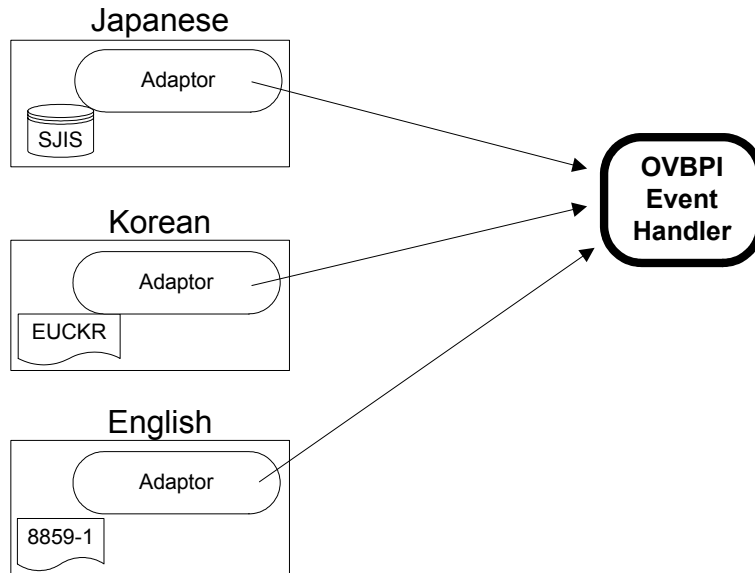
By default the OVBPI Event Handler does not have the `TextEncoding` property set for any of its components. This means that the OVBPI Event Handler expects to receive data according to the system default character set of the machine on which it is running. Unfortunately, this also means that the sink used within the OVBPI Event Handler is going to default to ISO-8859-1.

The fact that the OVBPI Event Handler source component is expecting data to come in using the system default's character set, the adaptor network as shown in Figure 38 causes problems. You are not able to configure the Japanese adaptor to send SJIS encoded data and the Korean adaptor to send EUCKR encoded data, as this causes the OVBPI Event Handler to fail.

When setting up an adaptor network to allow 8 and 16 bit data to pass correctly through into OVBPI you need to have all the adaptors send their data in a common character set, such as UTF-8.

Your adaptor network looks like this:

Figure 39 Mixed Character Sets - using UTF-8
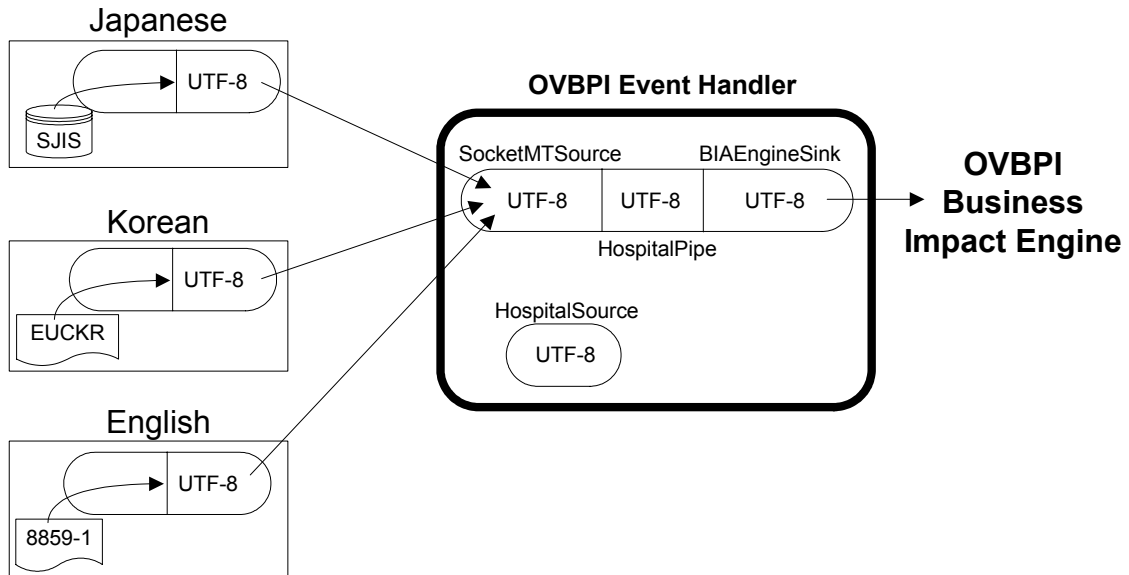
where:

- The source within each adaptor is able to read the input data using its system default character encoding
- The sink of each adaptor is set to send the data encoded using UTF-8
- The OVBPI Event Handler is configured to operate using UTF-8

  This involves setting the TextEncoding property for all components of the OVBPI Event Handler to ensure they all handle the data correctly.

OVBPI Event Handler

By default, the OVBPI Event Handler does **not** set the text encoding.

To set the text encoding for the OVBPI Event Handler you set the
`TextEncoding` property for each of the components:

```
SocketMTSource
BIAEngineSink
HospitalPipe
HospitalSource
```

You need to make the modifications to the file:

OVBPI-install-dir\data\conf\bia\bia_event_engineadaptor.props

and also to the files:

```
bia_event_engineadaptor.mssql.props
bia_event_engineadaptor.oracle.props
```

in the directory:

OVBPI-install-dir\newconfig\DataDir\conf\bia

8/16 Bit Adaptor Property Files

What if you need to enter some 8 or 16 bit characters within your adaptor property file? What character set is used when the adaptor actually reads its own property file?

By default, an adaptor reads its property file using the `ISO-8859-1` character set. If you wish to have your adaptor property file read using a different character set you can use the `PropertyFileTextEncoding` property.

For example, if your adaptor property file contains the line:

```
A.PropertyFileTextEncoding = UTF-8
```

then all lines within the property file that relate to adaptor `A` are read using the `UTF-8` character set. Thus allowing you to specify UTF-8 characters within the property file as and where you need, for adaptor `A`.

Notice that the `PropertyFileTextEncoding` property is prefixed by the name of the adaptor. This is an adaptor-wide property setting.

Multiple Unique IDs

When an event arrives at the OVBPI Business Impact Engine, for a specific data instance, the normal action is to create a new data instance if one doesn't already exist. For example, you send in a `New Order` event, for order `123`, and a new data instance is created for you and the unique id is set to `123`. All further events that arrive for this order id `123`, update this data instance.

Indeed, even if the events arrive at the OVBPI Business Impact Engine out of sequence it doesn't really matter. For example, suppose events arrive in the following sequence:

```
Order Update (orderid=123, status=shipped)
New Order    (orderid=123, value=$500)
```

The `Order Update` event arrives first and automatically creates a new data instance for order ID `123`. When the `New Order` event arrives some time later it simply updates the already created data instance for order ID `123`, and everything is fine.

However, what if your flow is tracking an order that moves across (for example) two different application systems. In the first application system the order is identified by an `OrderID`, but throughout the second application system, the same order is identified by an `OrderTracker`. Suppose the event sequence is as follows:

```
New Order           (OrderID=123, ...)
Update Order        (OrderID=123, ...)
X-Application Send  (OrderID=123, OrderTracker=abc)
Update Tracker      (OrderTracker=abc, ...)
Complete Tracker    (OrderTracker=abc, ...)
```

where:

- The first two events (`New Order` and `Update Order`) are uniquely identified by the OrderID attribute, and the value is `123`.

- When the order is sent from application system one to application system two, the event `X-Application Send` provides the mapping that OrderID `123` is now known as OrderTracker `abc`.

- The last two events provide information about the order as it moves through the second application system. Each event is identified by the OrderTracker attribute, whose value is `abc`.

If all events arrive in the correct order then everything is OK.

However, what happens if the events arrive in this order:

```
New Order           (OrderID=123, ...)
Update Order        (OrderID=123, ...)
Update Tracker      (OrderTracker=abc, ...)
X-Application Send  (OrderID=123, OrderTracker=abc)
Complete Tracker    (OrderTracker=abc, ...)
```

When the `New Order` event arrives, a new data instance is created for OrderID 123. When the `Update Order` event arrives, this updates the same data instance where OrderID = 123. When the `Update Tracker` event arrives the OVBPI Business Impact Engine tries to locate an existing data instance where OrderTracker=abc. It does not find one...so it creates a new data instance setting OrderTracker = abc. You now have two data instances - one with OrderID = 123 and another with OrderTracker = abc. Unfortunately, this is not what you wanted! Because the `X-Application Send` event had not made it through to the OVBPI Business Impact Engine in time, the Business Impact Engine had no way to know that OrderTracker abc should have been tied to the existing data instance where OrderID = 123.

Handling Out of Sequence Events

In the case where a flow is monitoring across more than one application system, and therefore the item being monitored is known by more than one unique ID, you need to configure your flow model not to automatically create new data instances all the time.

Consider the example where the events are as follows:

```
New Order           (OrderID=123, ...)
Update Order        (OrderID=123, ...)
X-Application Send  (OrderID=123, OrderTracker=abc)
Update Tracker      (OrderTracker=abc, ...)
Complete Tracker    (OrderTracker=abc, ...)
```

Within the OVBPI Modeler, you set up the data definition subscriptions for each of these events.

For the `New Order`, `Update Order` and `X-Application Send` events, you can accept the default setting:

```
Create the data definition instance if it does not exist.
```

However, for the remaining `Update Tracker` and `Complete Tracker` events, you set the creation option to be:

> `Fail if the data definition instance does not exist.`

The subscription page looks something like this:

Figure 40 Fail the Event Subscription

where:

1. The event subscription is for a specific instance of the data definition

2. You select that the event **fails** if the specific data instance does not yet exist

However, setting the option to `Fail` does **not** mean that the event fails and is then lost. When the event arrives at the OVBPI Business Impact Engine, if there is no current data instance, the event is "failed" and therefore not applied by the Engine. The OVBPI Business Impact Engine then marks this event to go into the event hospital but with the patient status already set to `discharge wait`. The OVBPI Business Event Handler then places this event in the hospital.

The Business Event Handler runs a `HospitalSource` that polls the hospital data table every three minutes looking for patients that are ready to be discharged and sent back into the OVBPI Business Impact Engine. Thus by "failing" the event and placing it into the event hospital marked for discharge, the event is re-tried every few minutes. The hope is that during this retry time the missing event that sets up the data instance will arrive.

The polling time of the `HospitalSource`, and the maximum retry count, are configurable through the OVBPI Administration Console - under the `Business Event Handler` section.

7 SOA Manager

The *OVBPI Integration Training Guide - Modeling Flows* shows you how to integrate OVBPI with SOA Manager such that your OVBPI flow definitions can receive operational status events from SOA Manager Business Services. Thus, if any of your SOA Managed Business Services become unavailable, OVBPI is able to show you the resultant impact this is having on your business.

There is a further integration that can be achieved between OVBPI and SOA Manager. You can configure an openadaptor adaptor to monitor the SOA Manager audit logs and feed these into OVBPI as business data events. For example, as a user invokes a SOA Manager Web service to place a new order, you are able to feed this event into OVBPI and start monitoring the progression of this order.

This chapter describes how to configure an openadaptor adaptor to send business events, from SOA Manager, into OVBPI.

Introduction

SOA Manager audit logging can be configured to record the message exchanges for a managed Web service. A SOA managed Web service can be configured to record just the incoming request, just the outgoing reply, or both. You can view these audit logs from within the SOA Manager Business Service Explorer (BSE); see Figure 41.

Figure 41 Web Service Audit Logging
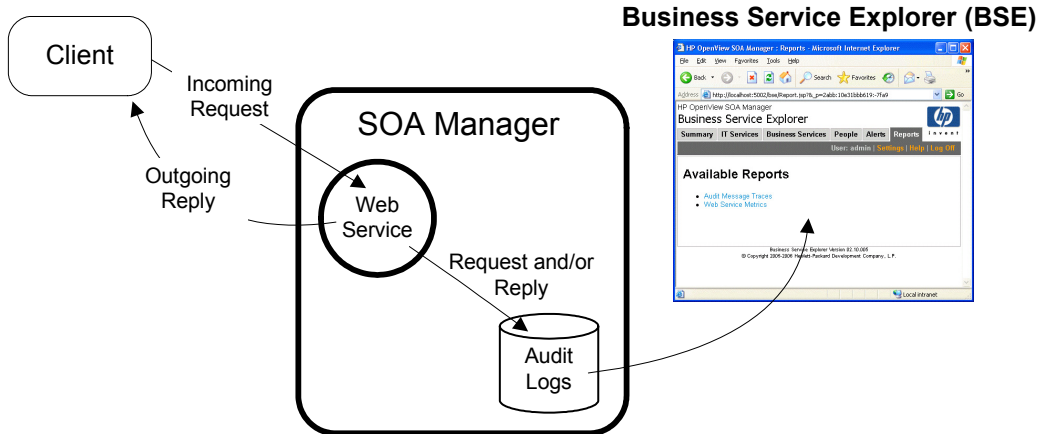
You can probably see that if you had an openadaptor source that was able to read the SOA Manager audit logs, then you would be able to send business data events into OVBPI whenever your Web service is invoked. Indeed, OVBPI provides two openadaptor components to allow just this.

OVBPI provides the following two components:

- The BIASOAMSource component.

  This openadaptor component is able to read the audit logs. As each new audit log is created, the source outputs an XML message containing this audit log.

- The XSLTransformPipe component.

  This openadaptor component is able to apply a user-supplied XSL transformation to the XML that comes from the BIASOAMSource. The output is a business event that you can send into the OVBPI Business Event Handler.

BIASOAMSource

The `BIASOAMSource` reads the SOA Manager audit logs and outputs these as XML. Access to the audit logs is provided by the SOA Manager Web service `AuditService`.

You configure the `BIASOAMSource` to make calls to the `AuditService` Web service, asking for the next batch of audit log records. As each batch of audit logs is processed, the `BIASOAMSource` records the timestamp of the last record processed. The next time the `BIASOAMSource` calls the `AuditService` Web service, it retrieves all records that have been written to the audit logs since the last record it processed.

The `BIASOAMSource` can be configured to retrieve the audit logs for more than one service. If you specify multiple services then the `BIASOAMSource` calls the audit logs for each service individually. That is, the `BIASOAMSource` gets the set of new audit logs for service one, then the set for service two, and so on. The `BIASOAMSource` stores the timestamp of the last audit log processed for each service. These timestamps are used the next time the `BIASOAMSource` polls for more data.

The `BIASOAMSource` uses the `org.openadaptor.adaptor.util.AdaptorHistory` class to save the details of the timestamp of the last record processed for each service name. (Refer to AdaptorHistory on page 117 for details about the `AdaptorHistory` class and where the adaptor history information is stored.)

Output Message Format

Once the `BIASOAMSource` has polled for the audit logs of all configured services, these logs are output as a single openadaptor message. This message is an array of JDOM data objects. (A JDOM data object is a subclass of the standard openadaptor data object.) Each JDOM data object is an XML document.

In other words, the output from the `BIASOAMSource` is an array of XML documents, and each XML document corresponds to an entry from the SOA Manager audit logs. The format of these XML documents is shown in Figure 42 on page 308.

Figure 42 XML Output Format for each Audit Log Record

<payloadXML>

   <Message serviceName = "…"
                   duration   = "…"
                   sequenceid = "…"
                   timestamp   = "…"
                   endpoint   = "…" **>**

      <Request>

          ...The XML request from the SOA Manager audit log...

      </Request>

      <Response>

          ...The XML response from the SOA Manager audit log...

      </Response>

   </Message>

</payloadXML>

where:

- `payloadXML` element

  The document root element is `payloadXML`.

- `Message` element

  The `Message` element contains the actual details of the audit log.

  Some of the properties of the audit log record are set as attributes of the `Message` element. These attributes are:

  — `serviceName`

    The name of the Web service that produced this audit log.

  — `duration`

    The duration of the Web service transaction that produced this audit log.

  — `sequenceid`

    A sequence value assigned to this audit log by SOA Manager.

— timestamp

The time that this audit log was created. This time is expressed as the number of milliseconds since January 1st, 1970.

— endpoint

This is the URL of the `AuditService` Web service.

- `Request` and `Reply` elements

    If the audit log entry contains the incoming request XML, this XML is placed within the `Request` element.

    If the audit log entry contains the outgoing response XML, this XML is placed within the `Response` element.

    When you configure audit logging for a service within SOA Manager, you configure how much of the Web service transaction you wish to save in the audit log. You can configure whether to save just the incoming request, just the outgoing reply, or both.

Configurable Properties

The class name for the `BIASOAMSource` is `org.openadaptor.adaptor.bia.BIASOAMSource`.

The following configuration options are available:

```
EndpointURL
ServiceNameN
PollPeriod
BatchSize
StartTime
InfoXML
InhibitHistory
RetryCount
RetryDelay
```

where:

- `EndpointURL`

    You specify the URL of the `AuditService` Web service that provides access to the SOA Manager audit logs.

For example:

```
A.C1.EndpointURL =
        http://hostname:5002/wsmf/services/Runtime$service=AuditService?wsdl
```

where *hostname* is your SOA Manager server.

- ServiceNameN

    This tells the `BIASOAMSource` to retrieve the audit logs for the specified service name.

    For example:

    ```
    A.C1.ServiceName1 = NewOrderServiceProxy
    ```

    If you want the `BIASOAMSource` to retrieve the audit logs for more than one service, then you specify multiple `ServiceNameN` properties, where `N` is an integer incrementing from 1.

    For example:

    ```
    A.C1.ServiceName1 = NewOrderServiceProxy
    A.C1.ServiceName2 = CustomerUpdate
    ```

    Refer to Business Service vs Web Service on page 340 for more discussion about the Web service name you specify.

- PollPeriod (optional)

    The `BIASOAMSource` makes repeated calls to the `AuditService` Web service as specified by the `EndpointURL` property. As the `BIASOAMSource` retrieves the next batch of audit logs, it outputs these as a single openadaptor message (an array of Data Objects). Once that batch of records has been sent, the `BIASOAMSource` loops back to call the `EndpointURL` for the next batch of audit logs. That is, the `BIASOAMSource` keeps calling the `EndpointURL`, and processing records, until there are no more audit logs ready for processing. Only when there are no more audit logs waiting to be processed, does the `BIASOAMSource` pause. The length of this pause is specified by the `PollPeriod` property. You specify the time, in milliseconds.

    The default `PollPeriod` value is `1000`msecs.

    If you want the `BIASOAMSource` component to pause after processing each batch of audit logs, regardless of how many audit logs are still ready to be processed, then you can specify the `MaxCallsPerPoll` property. The

`MaxCallsPerPoll` property is defined for all openadaptor source components and it might be of use if, for example, you are running a demonstration and you want to slow down the injection of the audit log records into OVBPI. Refer to Delaying Your Adaptor (MaxCallsPerPoll) on page 271 for details about the `MaxCallsPerPoll` property.

- `BatchSize` (optional)

  The `BatchSize` property sets the maximum number of log records retrieved, for each service, on a single call to the `EndpointURL`.

  For example, if you have specified two service names and a batch size of 10, then each output from the `BIASOAMSource` may contain up to a maximum of 20 audit logs; up to 10 for the first service and up to 10 for the second service. These 20 log records are contained within a single XML output message.

  By default, the batch size is not set, which means that each call to the `EndpointURL` returns all the audit logs ready for processing.

- `StartTime` (optional)

  By specifying a start time, the `BIASOAMSource` only processes audit logs that occur on or after this start time.

  The start time can be specified in one of the following formats:

  — A string value representing the number of milliseconds since 1970

    For example: `1066060093765`

    would represent the date: `10 October 2003 16:48:13 GMT`

  — A string in the format: `YYYY-MM-DDThh:mm:ss.SSS TZN`

    The milliseconds (`SSS`) part of the format is optional.

    The time zone can either be expressed as a time adjustment, or as an abbreviation.

    For example, the following start time values are acceptable:

    ```
    2006-11-23T12:56:01 MEST
    2006-11-23T12:56:01 GMT
    2006-11-23T12:56:01 +02:00
    2006-11-23T12:56:01.951 +02:00
    2006-11-23T12:56:01.951 MEST
    ```

If you do not specify a start time, the `BIASOAMSource` assumes a start time of January 1st 1970. This causes the `BIASOAMSource` to start processes the audit logs from the beginning.

- `InfoXML` - For debug use only.

  By default, when the `BIASOAMSource` has started processing audit logs, it produces little or no output to standard out. However, if you set the `InfoXML` property to `true`, then the `BIASOAMSource` displays each output XML record to standard out.

  This property defaults to `false`.

- `InhibitHistory` - For debug use only.

  Setting the `InhibitHistory` property to `true` tells the `BIASOAMSource` not to save the timestamps of the audit logs as it processes them. This means that the `BIASOAMSource` starts up, gets the first batch of audit logs, processes them, and then gets the same batch of audit logs again. You might think this sounds a bit strange. The idea of this `InhibitHistory` property is that when you are carrying out initial tests of your `BIASOAMSource`/`XSLTransformPipe` adaptor, you can run the `BIASOAMSource` with `InhibitHistory` set to `true` and test out your XSL transformations. See for more explanation.

- `RetryCount`
  `RetryDelay`

  The `RetryCount` property sets the maximum number of times the `BIASOAMSource` retries the connection to the `EndpointURL` if it encounters an error. This property defaults to `3`.

  The `RetryDelay` property sets the time delay, in milliseconds, between the retries. This property defaults to `1000`.

Example Configuration

The following example configures the `BIASOAMSource` to poll the `AuditService` Web service every 30 seconds and retrieve the audit logs for the two services `OrderService` and `FinanceServiceProxy`.

The `BIASOAMSource` retrieves the next log records for `OrderService` (up to a maximum of 10), then retrieves the next log records for `FinanceServiceProxy` (up to a maximum of 10). The `BIASOAMSource` then sends all these audit logs as a single openadaptor message. The `BIASOAMSource` then retrieves the next set of audit logs for these services. As long as there are audit logs waiting to be processed, the `BIASOAMSource` processes them. When there are no more audit logs waiting to be processed, the `BIASOAMSource` sleeps for 30 seconds, before looking to see if any more audit logs are ready to be processed.

```
A.C1.ClassName    = org.openadaptor.adaptor.bia.BIASOAMSource
A.C1.EndpointURL  = http://localhost:5002/wsmf/services/
                                   Runtime$service=AuditService?wsdl
A.C1.ServiceName1 = OrderService
A.C1.ServiceName2 = FinanceServiceProxy
A.C1.PollPeriod   = 30000
A.C1.BatchSize    = 10
```

Example Configuration with Start Time

The following example configures the `BIASOAMSource` to poll the `AuditService` Web service every 5 seconds and retrieve the audit logs for the two services `OrderService` and `FinanceServiceProxy`.

On the first call to the `AuditService` for each service, the `BIASOAMSource` specifies a starting time of `2006-11-23T12:56:01.951` MEST. Once audit logs for this time or later are processed, the `BIASOAMSource` maintains an up to date timestamp and keeps track of the records it has processed. It uses the `AdaptorHistory` class to maintain these timestamp details for each service. (Refer to AdaptorHistory on page 117 for details about where the adaptor history is stored.)

The BIASOAMSource retrieves the next log records for OrderService (up to a maximum of 50), then retrieves the next log records for FinanceServiceProxy (up to a maximum of 50), then sends all these as a single openadaptor message. When there are no more audit logs ready for processing, the BIASOAMSource sleeps for 5 seconds.

```
A.C1.ClassName    = org.openadaptor.adaptor.bia.BIASOAMSource
A.C1.EndpointURL  = http://localhost:5002/wsmf/services/
                                  Runtime$service=AuditService?wsdl
A.C1.ServiceName1 = OrderService
A.C1.ServiceName2 = FinanceServiceProxy
A.C1.PollPeriod   = 5000
A.C1.BatchSize    = 50
A.C1.StartTime    = 2006-11-23T12:56:01.951 MEST
```

XSLTransformPipe

The `XSLTransformPipe` is written to accept the XML output from the `BIASOAMSource`. The `XSLTransformPipe` lets you apply an XSL transformation file to these messages and produce business events ready to be sent to the OVBPI Business Event Handler.

The class name for the `XSLTransformPipe` is `org.openadaptor.adaptor.bia.XSLTransformPipe`.

The `XSLTransformPipe` has just one property, `XSLFileName`. You specify the name of an XSL transformation file and the `XSLTransformPipe` applies this transformation file to the incoming XML documents.

The `XSLTransformPipe` receives an array of XML documents that conform to the format shown in Figure 42 on page 308. The XSL transformations that you specify must convert each input XML document to XML that corresponds to event definitions that the OVBPI Event Handler is expecting to receive.

An XML Business Event

So how do you specify an OVBPI event definition within XML?

Suppose that within OVBPI you have an event definition called `Orders/New`, which has the attributes `OrderID` and `Amount`. To represent this within XML, your adaptor needs to build a document with the format as shown in Figure 43 on page 315.

Figure 43 Example Business Event Format in XML

```
<OADOType>
    <EventName>
        Orders/New
    </EventName>
    <OrderID>
        123
    </OrderID>
    <Amount>
        45.36
    </Amount>
</OADOType>
```

where:

- The enclosing document element `OADOType` sets the openadaptor data object type (DO type) for this document.

   This DO type may be of use if you are going to pass this on to another openadaptor pipe that offered filtering based on DO type. But for OVBPI, the actual name of the outer element has no significance.

- You specify the name of the event within the `EventName` element.

- You specify the `OrderID` and `Amount` fields as XML elements of these names.

- You could also include a `GeneratedDate` element if you wished to set the event creation time.

So the XSL transformation file for the `XSLTransformPipe` needs to transform the incoming XML, as shown in Figure 42 on page 308, to a format similar to that shown in Figure 43 on page 315.

The Incoming XML

When writing your XSL transformation file, you need to write transformations that pull data out of the incoming XML document and then place these into an outgoing XML document. So how do you find out exactly what data is coming in to your `XSLTransformPipe`?

The overall format of each incoming XML document is shown in Figure 42 on page 308. That is, the output from the `BIASOAMSource` is the input to your `XSLTransformPipe`. But when writing your XSL transformation you also need to know the format of the XML within the `Request` and `Response` elements.

There are two ways to work out the format of the XML that is going to be received by the `XSLTransformPipe`:

1. Using the Business Service Explorer (BSE)

   The BSE lets you view the contents of the SOA Manager audit logs. You can then substitute the various sections of this audit log into the format shown in Figure 42 on page 308.

2. Running `BIASOAMSource` in debug mode.

   You can configure the `BIASOAMSource` to display the complete XML for each output record.

Using the Business Service Explorer

The overall format of the XML you need to transform is shown in Figure 42 on page 308. You can use the Business Service Explorer (BSE) to view your audit logs and this lets you see the content of the Request and Reply XML. By combining the Request and Reply XML with the format as shown in Figure 42 on page 308, you can construct a full picture of the incoming XML record that you need to transform.

To view the audit logs using the BSE:

1. Log on to the BSE.

   You run a Web browser and go to the URL:

   `http://hostname:5002/bse`

   where:

   — *hostname* is the host running your BSE.

   — 5002 is the default port number for the BSE.

2. Click on the Reports tab.

   This displays the list of available reports.

3. Click on the Audit Message Traces option.

4. In the Query section, configure the following query fields:

   — Search For: Select the Success and/or the Failure check boxes.

   — Service: Use the drop-down lists to constrain the query by the service type.

   — Start Date: Use the fields to enter a specific start date and start time for the query to match.

   — End Date: Use the fields to enter a specific end date and end time for the query to match.

   — User: Enter a user in the field if you want query the audit log messages based on a specific user that made the request.

   If you simply accept the default, it selects all audit logs for today.

5. Click Query.

   The results of the query are listed in the Results section.

6. To see the actual details of any listed audit log, click on the timestamp (which shows as an HTML link) for the specific audit log.

Figure 44 An Example Audit Log

Request Details	
Sequence Id:	c9d2386b78853fc6d3c3b01b8749b043@bentong6.emea.hpqcorp.net
Timestamp:	12 October 2006 15:07:10 BST
Duration:	31 ms
User:	
Endpoint:	http://bentong6.emea.hpqcorp.net:9032/HPQfinanceServiceProxy
Incoming Message:	<?xml version="1.0" encoding="UTF-8"?> <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/" xr open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd" xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oa xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><soap:Header><wsa:Action>http://wsm.hp.com/finance/getG e34fa9b39b69</wsa:MessageID><wsa:ReplyTo><wsa:Address>http://schemas.xmlsoap.org/ws/2004/03/addressing/role/ar wsdl</wsa:To><wsse:Security><wsu:Timestamp wsu:Id="Timestamp-30d138b2-48df-41e5-8746-1897d6408280"><wsu:Crea 12T14:12:10Z</wsu:Expires></wsu:Timestamp></wsse:Security></soap:Header><soap:Body><QuoteRequest xmlns="http://wsm.hp.com/Finance/Request"><symbol>HPQ</symbol></QuoteRequest></soap:Body></soap:Envelope>
Outgoing Message:	<?xml version="1.0" encoding="UTF-8"?> <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelc xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"><soapenv:Body><QuoteResponse xmlns="http://wsm.hp.con 6.67</percentChange><price>1.75</price></getQuoteResult></QuoteResponse></soapenv:Body></soapenv:Envelope>

where the BSE shows the following details for each audit log:

- Sequence ID

  This is a unique ID assigned by SOA Manager for this audit log entry.

- Timestamp

  The timestamp of when this audit log was created.

- Duration

  How long the Web service took between receiving the request and sending back the reply.

- User

  The name of the requester.

- Endpoint

  The endpoint of the request that logged this audit log.

- Incoming Message

  If the audit log is configured to log the request information, then you see the XML of the incoming SOAP request.
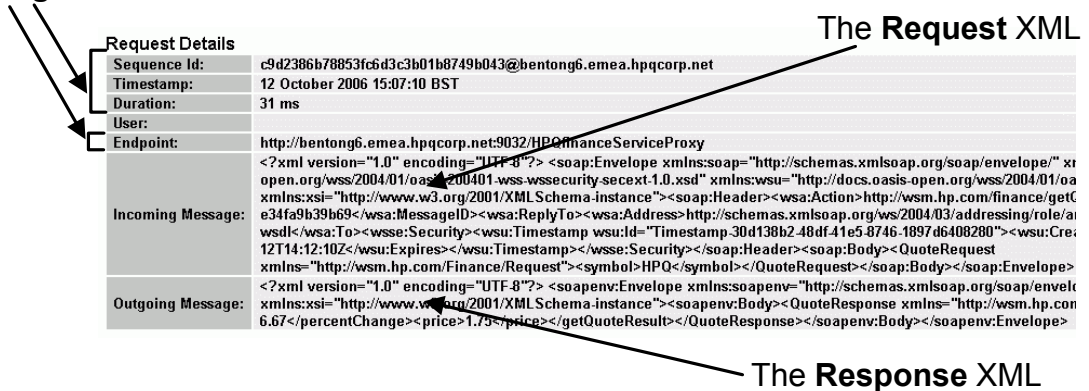
- Outgoing Message

  If the audit log is configured to log the response information, then you see the XML of the outgoing SOAP response.

Figure 45 shows where each part of the audit log appears within the XML that is output from the BIASOAMSource.

Figure 45 The Sections of the Audit Log

where:

- The fields Sequence Id, Timestamp, Duration and Endpoint are all set as attributes on the Message element.

- The User attribute is **not** included in the final XML that is output from the BIASOAMSource.

- The Incoming Message XML is placed within the Request element after removing the <?xml ... > directive.

- The Outgoing Message XML is placed within the Response element after removing the <?xml ... > directive.

Running BIASOAMSource in debug mode

You can configure the `BIASOAMSource` to run in a debug-style mode where it displays the actual XML for each record to standard out. This allows you to see the complete XML document that your XSL transformations need to manipulate.

You set the following two `BIASOAMSource` properties:

```
InhibitHistory = true
InfoXML        = true
```

Setting `InfoXML` to `true` tells the `BIASOAMSource` to display the XML of each output record to standard out. Setting `InhibitHistory` to `true` tells the `BIASOAMSource` not to save any details of the audit logs it processes.

By setting both these properties, the `BIASOAMSource` retrieves the first batch of audit logs, processes these and displays the XML for each record to standard out. Because `InhibitHistory` is set to `true`, the `BIASOAMSource` then retrieves the same set of audit logs again, and reprocesses them. This might seem a little strange, however, the idea is that you can run this adaptor as many times as you like, and test out your XSL transformations to see if they work. Once you have defined and tested your XSL transformation, you can remove both the `InfoXML` and `InhibitHistory` properties from your `BIASOAMSource` configuration, and you are ready to run your adaptor and send the events into OVBPI.

The use of the `InfoXML` property makes it very easy for you to see the XML that your `XSLTransformPipe` needs to transform.

Events from a Single Service

Let's consider an example where you are monitoring the SOA Manager audit logs for a single service. The service is called `NewOrderService` and this Web service is invoked whenever a customer places a new order.

This example shows how to transform the audit logs for the `NewOrderService` Web service into the event definition `Orders/New` with the following attributes:

- `OrderNumber` - The order number assigned by the Web service.
- `ClientId` - The ID of the customer placing this order.

Configuring the BIASOAMSource

For this example, the configuration for the `BIASOAMSource` is as follows:

```
A.C1.ClassName     = org.openadaptor.adaptor.bia.BIASOAMSource
A.C1.PollPeriod    = 5000
A.C1.BatchSize     = 10
A.C1.EndpointURL   = http://hostname:5002/wsmf/services/
                                    Runtime$service=AuditService?wsdl
A.C1.ServiceName1 = NewOrderService
```

The `BIASOAMSource` polls the audit logs looking for entries for the `NewOrderService` service. Each poll retrieves up to 10 audit logs. The `BIASOAMSource` then sends these 10 records within a single openadaptor message. This single message is an array of XML documents, where each document corresponds to an individual audit log entry. This array of XML documents is then passed to the `XSLTransformPipe`.

Configuring the XSLTransformPipe

The configuration for the `XSLTransformPipe` is as follows:

```
A.P1.ClassName     = org.openadaptor.adaptor.bia.XSLTransformPipe
A.P1.XSLFileName   = ex_newOrder.xsl
```

The `XSLTransformPipe` pipe applies the XSL transformations contained within the file `ex_newOrders.xsl`. The `XSLTransformPipe` receives the array of XML documents from the `BIASOAMSource` and the XSL transformations are applied individually to each XML document. The resultant output from the `XSLTransformPipe` is another single openadaptor message containing all the transformed XML documents. In other words, the output of the `XSLTransformPipe` is an array of data objects. each containing an XML event definition.

To write the XSL transformation file you need to know the exact format of the input your `XSLTransformPipe` is going to receive from the `BIASOAMSource`. The general format of the output from the `BIASOAMSource` is shown in the diagram XML Output Format for each Audit Log Record on page 308.

The XML to be Converted

The easiest way to see the XML that needs to be transformed is to run the `BIASOAMSource` in debug mode (See Running BIASOAMSource in debug mode on page 320 for details.)

Here is an example of the XML to be received from the `BIASOAMSource`:

```
<payloadXML>

  <Message serviceName="NewOrderService"
           duration="281"
           sequenceid="a071a73cf00382e510169ab39c30395b@server.net"
           timestamp="1160479531907"
           endpoint="http://server.net:5002/wsmf/services/
                                       Runtime$service=AuditService?wsdl">
```

```
<Request>
  <soap:Envelope
        xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
        xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
                            oasis-200401-wss-wssecurity-secext-1.0.xsd"
        xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
                            oasis-200401-wss-wssecurity-utility-1.0.xsd"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soap:Header>
       <wsa:Action>http://wsm.hp.com/orders/getOrder</wsa:Action>
       <wsa:MessageID>
          uuid:52110abf-1b8c-485a-9f7d-fe85c2582506
       </wsa:MessageID>

       <wsa:ReplyTo>
          <wsa:Address>
             http://schemas.xmlsoap.org/ws/2004/03/addressing/
                                                  role/anonymous
          </wsa:Address>
       </wsa:ReplyTo>

       <wsa:To>http://localhost:9032/NewOrderService?wsdl</wsa:To>

       <wsse:Security>
          <wsu:Timestamp wsu:Id="Timestamp-a2753d53-9532-4e9a-95dfd">
             <wsu:Created>2006-10-12T14:15:11Z</wsu:Created>
             <wsu:Expires>2006-10-12T14:20:11Z</wsu:Expires>
          </wsu:Timestamp>
       </wsse:Security>
    </soap:Header>

    <soap:Body>
       <orderRequest xmlns="http://wsm.hp.com/Orders/Request">
          <customerId>HPQ</customerId>
       </orderRequest>
    </soap:Body>
  </soap:Envelope>
</Request>

<Response>
  <soapenv:Envelope
        xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
```

```
      <soapenv:Body>
         <orderResponse xmlns="http://wsm.hp.com/Orders/Response">
            <getOrderResult>
               <orderNum>EB347-2095</orderNum>
            </getOrderResult>
         </orderResponse>
      </soapenv:Body>
   </soapenv:Envelope>
  </Response>
 </Message>
</payloadXML>
```

where:

- The main structural elements, as shown in the diagram , are marked in bold.

- The XML contained within the `Request` element contains the request that was used to run the Web service.

- The XML contained within the `Response` element contains the actual XML of the response from the Web service.

- The XML defines a number of name spaces.

  When you are writing XSL transformations for this XML you need to ensure that you correctly state the element's name and the element's name space.

The XSL Transformation File

Here is the XSL necessary to transform the above XML into the event definition, ready to send to OVBPI:

```
<xsl:stylesheet version="1.0"
          xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
          xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
          xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
          xmlns:ns1="http://wsm.hp.com/Orders/Request"
          xmlns:ns2="http://wsm.hp.com/Orders/Response"  >

  <xsl:output method="text"/>

  <xsl:template match="Message">

      <Any>

        <EventName>Orders/New</EventName>

        <GeneratedDate>
            <xsl:value-of select="@timestamp" />
        </GeneratedDate>

        <orderNumber>
            <xsl:value-of select="Response/soap:Envelope/soap:Body/
                      ns2:orderResponse/ns2:getOrderResult/ns2:orderNum" />
        </orderNumber>

        <clientId>
            <xsl:value-of select="Request/soap:Envelope/soap:Body/
                                        ns1:orderRequest/ns1:customerId" />
        </clientId>

      </Any>

  </xsl:template>

</xsl:stylesheet>
```

where:

- The `xsl:stylesheet` element defines any required name spaces.

  The name of the name space does not need to match the name used within the incoming XML document. What must match is the actual URI of the name space.

- The `EventName` is set to the hard-codes string `Orders/New`.

- The GeneratedDate is set taken from the timestamp attribute of the Message element.

- The orderNumber is set to the value of the orderNum element from within the response part of the input XML.

- The clientId is set to the value of the customerId element from within the request part of the input XML.

- The outer element Any sets the openadaptor data object type to be the text Any. This has no effect on the business event, and you can chose any string value you like. The example here just happens to use the string Any.

In this example, the above XSL is saved to the filename ex_newOrder.xsl.

Example Output after the XSL Transformation

If you take the example output XML from the BIASOAMSource and apply the XSL transformation as shown above, the resultant output XML is as follows:

```
<Any xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
     xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
     xmlns:ns1="http://wsm.hp.com/Orders/Request"
     xmlns:ns2="http://wsm.hp.com/Orders/Response">

    <EventName>Orders/New</EventName>
    <GeneratedDate>1160479531907</GeneratedDate>
    <orderNumber>EB347-2095</orderNumber>
    <clientId>HPQ</clientId>
</Any>
```

where:

- The outer element sets the DO type to be Any.

- The XML name spaces defined within the Any element will be ignored when this XML gets to the OVBPI Business Event Handler.

- The event contains the properties EventName, GeneratedDate, orderNumber, and clientId.

The Adaptor Configuration File

In this example, the `BIASOAMSource` polls the audit logs for the service `NewOrderService` and outputs arrays of XML documents, where each document corresponds to an individual audit log entry. These arrays of XML documents are passed to the `XSLTransformPipe`.

For each array of XML documents received from the `BIASOAMSource`, the `XSLTransformPipe` pipe applies the specified XSL transformations. The XSL transformations are applied individually to each XML document. The resultant output from the `XSLTransformPipe` is an array of data objects, each containing an event definition. Each event definition is contained within an `Any` element. This means that the output from the `XSLTransformPipe` is an array of data objects all of DOType `Any`.

This array of data object can then be sent to the `SocketSink`, which can then send them to the OVBPI Business Event Handler.

So, the overall adaptor configuration file is as follows:

```
A.Component1.Name = C1
A.Component2.Name = P1
A.Component3.Name = C2

A.C1.LinkTo1 = P1
A.P1.LinkTo1 = C2

A.C1.ClassName      = org.openadaptor.adaptor.bia.BIASOAMSource
A.C1.PollPeriod     = 5000
A.C1.BatchSize      = 10
A.C1.EndpointURL    = http://hostname:5002/wsmf/services/
                                        Runtime$service=AuditService?wsdl
A.C1.ServiceName1 = NewOrderService

A.P1.ClassName      = org.openadaptor.adaptor.bia.XSLTransformPipe
A.P1.XSLFileName    = ex_newOrder.xsl

A.C2.ClassName      = org.openadaptor.adaptor.standard.SocketSink
A.C2.Port           = 44005
A.C2.HostName       = localhost
```
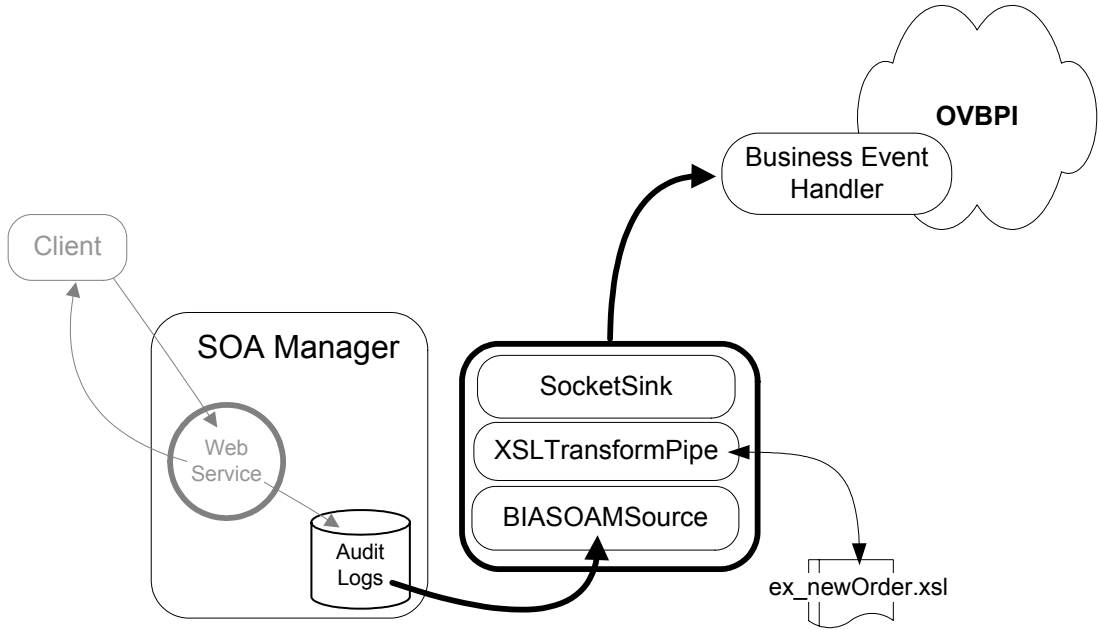
and the overall architecture for this adaptor is as shown in Figure 46 on page 328.

Figure 46 Adaptor Architecture for New Orders

Events from Multiple Services

Let's consider an example where you have an adaptor monitoring the SOA Manager audit logs for more than one service. In this example, you are monitoring the services `NewOrderService` and `FinanceService`.

This example shows how an XSL transformation file can be written to apply different transformations depending on the incoming audit log records. This example transforms the two types of audit logs as follows:

- The audit logs for the `NewOrderService` Web service are transformed into the event definition `Orders/New`, with the attributes `OrderNumber` and `ClientID`.

- The audit logs for the `FinanceService` Web service are transformed into the event definition `Finance/Quote`, with the attributes `ClientId` and `Price`.

Configuring the BIASOAMSource

For this example, the configuration for the `BIASOAMSource` is as follows:

```
A.C1.ClassName    = org.openadaptor.adaptor.bia.BIASOAMSource
A.C1.PollPeriod   = 5000
A.C1.BatchSize    = 10
A.C1.EndpointURL  = http://hostname:5002/wsmf/services/
                                      Runtime$service=AuditService?wsdl
A.C1.ServiceName1 = NewOrderService
A.C1.ServiceName2 = FinanceService
```

When the `BIASOAMSource` polls the audit logs, it polls twice. It first polls for all available audit logs for the service `NewOrderService`. The `BIASOAMSource` then polls for all available audit logs for the service `FinanceService`. Because the `BatchSize` property if set to 10, each of these polls may return up to the maximum of 10 audit logs. After it has retrieved the audit logs for both the configured services, the `BIASOAMSource` then outputs these records within a single openadaptor message. That is, a single array of up to 20 data objects. Each data object corresponds to an XML audit log. This array of XML audit logs is then passed to the `XSLTransformPipe`.

Configuring the XSLTransformPipe

The configuration for the XSLTransformPipe is as follows:

```
A.P1.ClassName    = org.openadaptor.adaptor.bia.XSLTransformPipe
A.P1.XSLFileName  = ex_multipleServices.xsl
```

The XSLTransformPipe pipe applies the XSL transformations contained within the file ex_multipleServices.xsl. The XSLTransformPipe receives the array of XML documents from the BIASOAMSource and the XSL transformations are applied individually to each XML document. The resultant output from the XSLTransformPipe is another single openadaptor message containing all the transformed XML documents. In other words, the output of the XSLTransformPipe is an array of data objects. each containing an event definition.

To write the XSL transformation file you need to know the exact format of the input your XSLTransformPipe is going to receive from the BIASOAMSource. The general format of the output from the BIASOAMSource is shown in the diagram XML Output Format for each Audit Log Record on page 308. However, in this example, the contents of the Request and Reply segments of XML are going to vary depending on whether it is an audit log for the NewOrderService or the FinanceService service.

The XML to be Converted

The easiest way to see the XML that needs to be transformed is to run the BIASOAMSource in debug mode (See Running BIASOAMSource in debug mode on page 320 for details.)

An example of the XML for the NewOrderService audit log is shown in the section The XML to be Converted on page 322.

Here is an example of the XML for the FinanceService audit log as received from the BIASOAMSource:

```
<payloadXML>

   <Message serviceName="FinanceService"
            duration="281"
            sequenceid="a071a73cf00382e510169ab39c30395b@server.net"
            timestamp="1160479531917"
            endpoint="http://server.net:5002/wsmf/services/
                                     Runtime$service=AuditService?wsdl">
```

```
<Request>
   <soap:Envelope
       xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
       xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
       xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/
                              oasis-200401-wss-wssecurity-secext-1.0.xsd"
       xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/
                              oasis-200401-wss-wssecurity-utility-1.0.xsd"
       xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <soap:Header>
     <wsa:Action>http://wsm.hp.com/finance/getQuote</wsa:Action>
     <wsa:MessageID>
         uuid:74110abf-1b8c-9f7d-485a-fe85c2582506
     </wsa:MessageID>
     <wsa:ReplyTo>
     <wsa:Address>
        http://schemas.xmlsoap.org/ws/2004/03/addressing/role/anonymous
     </wsa:Address>
     </wsa:ReplyTo>
     <wsa:To>http://localhost:9032/FinanceService?wsdl</wsa:To>
     <wsse:Security>
            <wsu:Timestamp wsu:Id="Timestamp-a2753d53-9532-4e9a-95df">
                <wsu:Created>2006-10-12T16:15:15Z</wsu:Created>
                <wsu:Expires>2006-10-12T16:32:15Z</wsu:Expires>
            </wsu:Timestamp>
     </wsse:Security>
   </soap:Header>

   <soap:Body>
     <quoteRequest xmlns="http://wsm.hp.com/finance/Request">
        <customerId>Compaq</customerId>
     </quoteRequest>
   </soap:Body>
   </soap:Envelope>
</Request>

<Response>
   <soapenv:Envelope
         xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
     <soapenv:Body>
        <quoteResponse xmlns="http://wsm.hp.com/finance/Response">
           <getQuoteResult>
              <change>-0.25</change>
              <percentChange>-0.22</percentChange>
              <price>112.265</price>
           </getQuoteResult>
        </quoteResponse>
```

```
            </soapenv:Body>
          </soapenv:Envelope>
      </Response>
    </Message>
</payloadXML>
```

where:

- The main structural elements, as shown in the diagram XML Output Format for each Audit Log Record on page 308, are marked in bold.

- The XML contained within the `Request` element contains the request that was used to run the Web service.

- The XML contained within the `Response` element contains the actual XML of the response from the Web service.

- The XML defines a number of name spaces.

  When you are writing XSL transformations for this XML you need to ensure that you correctly state the element's name (case sensitive) and the element's name space.

The XSL Transformation File

In this example, the `XSLTransformPipe` needs to be able to process two different types of audit log records and produce two different types of output. For the `NewOrderService` service, the XSL needs to produce the `Orders/New` event definition, and for the `FinanceService` service, the XSL needs to produce the `Finance/Quote` event definition.

Here is the XSL to transform the incoming XML documents into the appropriate event definitions:

```
<xsl:stylesheet version="1.0"
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
        xmlns:ns1="http://wsm.hp.com/Orders/Request"
        xmlns:ns2="http://wsm.hp.com/Orders/Response"
        xmlns:fns1="http://wsm.hp.com/finance/Request"
        xmlns:fns2="http://wsm.hp.com/finance/Response">

  <xsl:output method="text"/>

  <xsl:template match="Message">

      <xsl:choose>

<!-- ================== -->
<!-- A NewOrder document -->
<!-- ================== -->
        <xsl:when test="starts-with(@serviceName,'NewOrder')">

          <Any>
             <EventName>Orders/New</EventName>
             <GeneratedDate>
                <xsl:value-of select="@timestamp" />
             </GeneratedDate>
             <orderNumber>
                <xsl:value-of select="Response/soap:Envelope/soap:Body/
                        ns2:orderResponse/ns2:getOrderResult/ns2:orderNum" />
             </orderNumber>
             <clientId>
                <xsl:value-of select="Request/soap:Envelope/soap:Body/
                                        ns1:orderRequest/ns1:customerId" />
             </clientId>
          </Any>

        </xsl:when>
```

```
<!-- ================= -->
<!-- A Finance document -->
<!-- ================= -->
        <xsl:when test="starts-with(@serviceName,'Finance')">

            <Finance>
                <EventName>Finance/Quote</EventName>
                <GeneratedDate>
                    <xsl:value-of select="@timestamp" />
                </GeneratedDate>
                <clientId>
                    <xsl:value-of select="Request/soap:Envelope/soap:Body/
                                          fns1:quoteRequest/fns1:customerId" />
                </clientId>
                <price>
                    <xsl:value-of select="Response/soap:Envelope/soap:Body/
                                fns2:quoteResponse/fns2:getQuoteResult/fns2:price" />
                </price>
            </Finance>

        </xsl:when>

<!-- ================= -->
<!-- An unknown document -->
<!-- ================= -->
        <xsl:otherwise>
            <Unknown>
            </Unknown>
        </xsl:otherwise>

    </xsl:choose>
    </xsl:template>
</xsl:stylesheet>
```

where:

- The `xsl:stylesheet` element defines any required name spaces.

  The name of the name space does not need to match the name used within the incoming XML document. What must match is the actual URI of the name space.

- This XSL style sheet uses the `<xsl:choose>` construct to then test the value of the `serviceName` attribute.

  If the service name starts with the string `NewOrder` then the XSL transformation outputs the record as type `Any`, containing the event definition `Orders/New`.

If the service name starts with the string `Finance` then the XSL transformation outputs the record as type `Finance`, containing the event definition `Finance/Quote`.

If the service name is not one of the values expected, the XSL transformation outputs an empty record of type `Unknown`.

There are other ways that you can test values and attributes from within XSL. The XSL shown here is just one example. There are many Web sites and books that can provide more details about how to write XSL transformations.

In this example, the above XSL is saved to the filename `ex_multipleServices.xsl`.

Example Output after the XSL Transformation

The above XSL transformation file outputs two main types of XML document. For a `NewOrderService` audit log, an example output is as follows:

```
<Any xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
     xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
     xmlns:ns1="http://wsm.hp.com/Orders/Request"
     xmlns:ns2="http://wsm.hp.com/Orders/Response">

   <EventName>Orders/New</EventName>
   <GeneratedDate>1160479531900</GeneratedDate>
   <orderNumber>EB347-2095</orderNumber>
   <clientId>HPQ</clientId>
</Any>
```

where each `NewOrderService` audit log is transformed into an openadaptor data object of type `Any`.

For a `FinanceService` audit log, an example output is as follows:

```
<Finance xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
         xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/03/addressing"
         xmlns:ns1="http://wsm.hp.com/Orders/Request"
         xmlns:ns2="http://wsm.hp.com/Orders/Response"
         xmlns:fns1="http://wsm.hp.com/finance/Request"
         xmlns:fns2="http://wsm.hp.com/finance/Response">

   <EventName>Finance/Quote</EventName>
   <GeneratedDate>1160479531917</GeneratedDate>
   <clientId>Compaq</clientId>
   <price>112.265</price>
</Finance>
```

where each `FinanceService` audit log is transformed into an openadaptor data object of type `Finance`.

The Adaptor Configuration File

In this example, the `BIASOAMSource` polls the audit logs for the services `NewOrderService` and `FinanceService`. With each poll of the audit logs, the `BIASOAMSource` outputs these logs as an array of XML documents, where each document corresponds to an individual audit log entry. These arrays of XML documents are passed to the `XSLTransformPipe`.

For each array of XML documents received from the `BIASOAMSource`, the `XSLTransformPipe` pipe applies the specified XSL transformations. The XSL transformations are applied individually to each XML document. The resultant output from the `XSLTransformPipe` is an array of data objects, some of DO type `Any` and some of DO type `Finance`.

openadaptor sink components do not accept arrays of mixed DO type, so you need to send the output of the `XSLTransformPipe` to the `ArraySplitterPipe` component.

The `ArraySplitterPipe` acts as a buffer between the `XSLTransformPipe` and the sink component, passing each element of the data object array as an individual message to the sink (see ArraySplitterPipe on page 57 for further details about the `ArraySplitterPipe`.)

So, the overall adaptor configuration file is as follows:

```
A.Component1.Name = C1
A.Component2.Name = P1
A.Component3.Name = C2
A.Component4.Name = AS

A.C1.LinkTo1 = P1
A.P1.LinkTo1 = AS
A.AS.LinkTo1 = C2

A.C1.ClassName    = org.openadaptor.adaptor.bia.BIASOAMSource
A.C1.PollPeriod   = 5000
A.C1.BatchSize    = 10
A.C1.EndpointURL  = http://hostname:5002/wsmf/services/
                                    Runtime$service=AuditService?wsdl
A.C1.ServiceName1 = NewOrderService
A.C1.ServiceName2 = FinanceService

A.P1.ClassName    = org.openadaptor.adaptor.bia.XSLTransformPipe
A.P1.XSLFileName  = ex_multipleServices.xsl

A.AS.ClassName    = org.openadaptor.adaptor.standard.ArraySplitterPipe
```
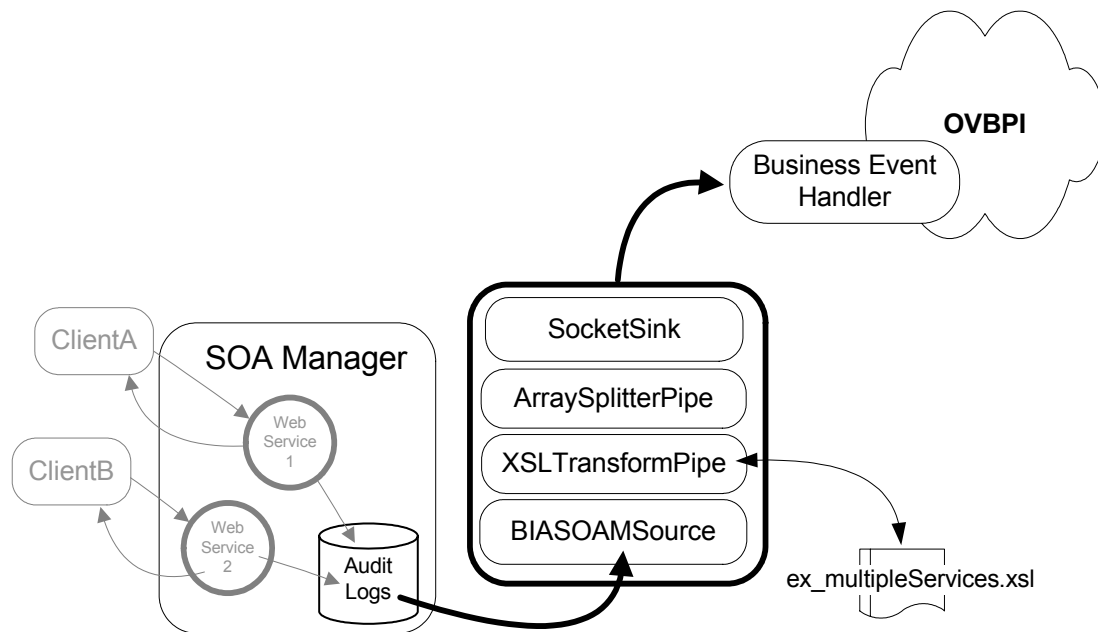
```
A.C2.ClassName    = org.openadaptor.adaptor.standard.SocketSink
A.C2.Port         = 44005
A.C2.HostName     = localhost
```

and the overall architecture for this adaptor is as shown in Figure 47 on page 338.

Figure 47 Adaptor Architecture for Multiple Services

Adaptor Architecture

You might think that to process the SOA Manager audit logs you configure a single adaptor. Indeed, this might be the case, however, you have a choice.

To configure a single adaptor to process the audit logs for multiple services, you need to be comfortable with writing XSL transformations that are able to differentiate between the various types of audit log.

You may find it easier to configure multiple adaptors, each handling either one service, or a number of related services. For example, you might configure a single adaptor that handles all the audit logs for services related to order processing, and a separate adaptor for the services related to the finance system.

Business Service vs Web Service

The previous chapters of this training guide describe how to configure the `BIASOAMSource` component to access the SOA Manager Web service audit logs. This allows you to configure an openadaptor adaptor to monitor the audit logs for a given SOA Manager Web service and send these audit logs into OVBPI as business data events. However, if you read through the *OVBPI Integration Training Guide - Modeling Flows*, is describes how to set up OVBPI to receive operational status events from SOA Manager Business Services. So, if you can configure the `BIASOAMSource` to monitor the audit logs of a SOA Manager service, and you can configure the OVBPI Modeler to monitor the operational status of a SOA Manager service, do you simply specify the same service name in both configurations? ...probably not!

When you configure the `BIASOAMSource` to monitor the audit logs of a Web service managed by SOA Manager, the service name you specify is the name of the service as configured within either a SOA Manager Web Service Broker, or Web Service Agent. For example, if you are running a SOA Manager Web Service Broker, then you use the Web service name listed within the Broker configurator console, as shown in Figure 48 on page 340.

Figure 48 Brokered Web Service List
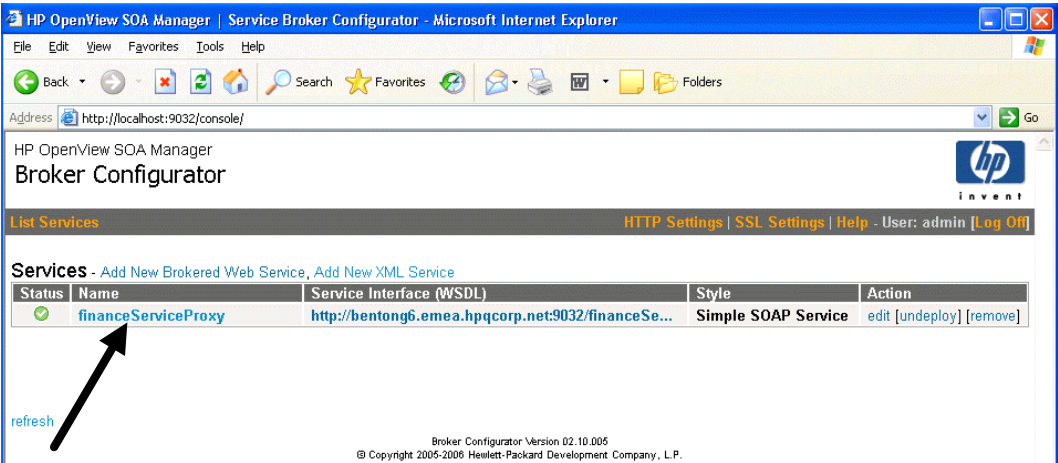
However, when you are in the OVBPI Modeler and you are linking to a SOA Manager service, you need to specify the name of a SOA Manager Business Service. That is, you need to specify the name of a Business Service that is configured within SOA Manager.

You can see the names of all configured Business Services by running the SOA Manager Business Service Explorer (BSE) and clicking on the Business Services tab, as shown in Figure 49 on page 341.

Figure 49 BSE - Business Service List
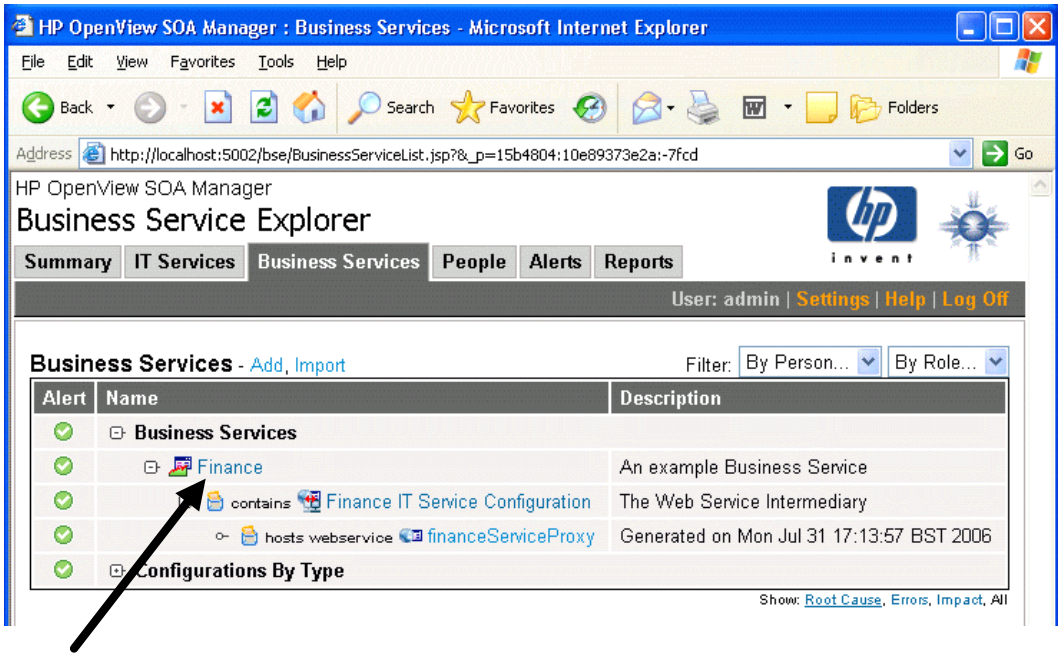
Notice that in Figure 49 on page 341, the Business Service called Finance actually corresponds to the Web service financeServiceProxy. So in the scenario as shown in Figure 48 and Figure 49, you would configure your BIASOAMSource to monitor the audit logs for the Web service financeServiceProxy, and, within the OVBPI Modeler, you would link to the SOA Manager Business Service Finance. You then have your OVBPI Flow and BIASOAMSource monitoring the same Web service.

Configuring SOA Manager Audit Logging

The way you configure audit logging is described in the *HP SOA Manager Administration Guide* and the actual mechanism depends on whether you are configuring a Web Service Management Broker or an Agent.

As an example, let's consider how to configure audit logging for a Web Service Management Broker.

You enable audit logging for each individual service using the SOA Manager Broker Configuration, as follows:

1. Log into the Broker Configurator.

   You run a Web browser and go to the URL:

   `http://`*hostname*`:9032/console`

   where:

   — *hostname* is the host running your Web Service Management Broker.

   — 9032 is the default port number for the Broker.

2. From the `Action` column, click `edit` for the brokered service you want to configure.

   The service configuration screen is now displayed.

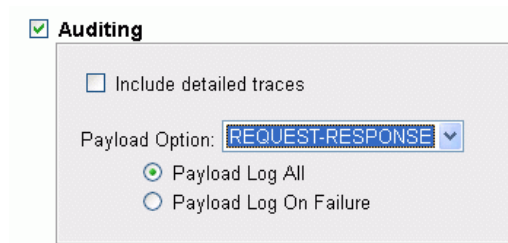3. Scroll down to the `Features` section and click the `Auditing` check box.

   You do not need to check the `Include detailed traces` check box.

   You then select how much of each message's SOAP payload you wish to log. You use the `Payload Option` drop-down list to select whether you want the payload to be collected for requests, responses, or both requests and responses. The option you choose depends what data you will need when you are building your business data event.

   You then select that you want to log the payload for all messages.

You Auditing section should look as shown in Figure 50.

Figure 50 The Auditing Settings

4. At the bottom of the screen, click `Save Changes`. The Brokered Services screen displays, and the brokered service is automatically deployed. The deployment is complete when the status changes to operational. You may need to refresh the screen to see the status change.

You can use the BSE to view audit log information (see Using the Business Service Explorer on page 317).

Audit Log Persistence

The SOA Manager audit logs are persisted to a database at regular intervals. The SOA Manager Web service `AuditService` provides access to the audit log database. Therefore, the `BIASOAMSource`, which uses the `AuditService` to gain access to the audit logs, only processes audit logs once they have been persisted to the audit log database.

The details for configuring the persistence of the audit logs is described in the *HP SOA Manager Administration Guide,* and it is different depending on whether you are configuring a Web Service Management Broker or an Agent.

As an example, let's consider how to configure the audit logging persistence for a Web Service Management Broker.

To configure the audit publisher for the Broker:

1. Stop the Broker if it is currently started.

2. Use a text editor to open the file:

   SOAM-INSTALL-DIR\conf\broker\mipServer.xml

3. Edit the audit publisher entries for `interval` and `threshold`.

   The interval value is specified in milliseconds and the threshold value is the total number of audit log messages:

   ```
   <entry name="com.hp.audit.publisher.interval">100000</entry>
   <entry name="com.hp.audit.publisher.threshold">10</entry>
   ```

   An `interval` value of `10000` means that every 10 seconds the audit logs are persisted to the database.

   A `threshold` of `10` means that the audit logs are persisted whenever there are 10 or more audit logs waiting to be flushed (see GeneratedDate on page 345 for more details.)

   The audit logs are persisted to the database when either the `interval` or `threshold` values are reached.

4. Save and close the file.

5. Restart the Broker for the changes to take effect.

GeneratedDate

When configuring the audit log persistence for a Web Services Management Broker the `threshold` setting (within the `mipServer.xml` file) actually sets the number of audit logs on an individual service basis.

Let's consider an example...

Suppose you set the `interval` value to be a large time interval, such as 3600000 (1 hour), and you set the `threshold` value to be 3. In other words, you set up a situation where you can generate audit logs within an hour and know that they will only be persisted each time you reach the `threshold` limit.

You have configured an adaptor to retrieve the audit logs for the services `WS-A` and `WS-B`.

You then run two Web service clients, one to invoke `WS-A` and the other to invoke `WS-B`.

As you invoke these Web services, audit logs are created in memory, and you would expect them to be flushed to the audit log database after every three records. But the logs are actually flushed individually for each service. That is, after you have created three audit logs for `WS-A`, this triggers these three `WS-A` audit logs to be flushed to the audit log database. Any audit logs for `WS-B` are not flushed at this time.

So be aware that the audit logs may not be persisted in the order you expect. However, the timestamp that is recorded with each audit log is set to the actual time the audit log was created in memory, not the time it was persisted.

For example, if you had the following set of audit log entries created at the times shown:

```
3:15 WS-A
3:16 WS-A
3:17 WS-B
3:18 WS-B
3:19 WS-A
```

None of these audit logs is persisted to the audit database until the `WS-A` log is created at `3:19`. At `3:19`, the following log records are persisted to the audit database:

```
3:15 WS-A
3:16 WS-A
3:19 WS-A
```

Once these logs are persisted to the audit database they are available for your
BIASOAMSource to process.

You can see now why it is important for your XSL transformation file to
always set the GeneratedDate attribute within the resultant event XML.

You can set the GeneratedDate to the timestamp attribute of the Message
element. If you do not set the GeneratedDate from the time stamp of the
audit log entry, your events appear to be out of sequence within OVBPI.

8 iWay Integration

In order to provide additional connectivity to business applications for event data, OVBPI includes iWay integration, in addition to openadaptor integration. The integration with iWay enables you to obtain business event data from applications such as SAP, PeopleSoft and JD Edwards for your business flows.

This chapter provides an introduction to iWay and its capabilities.

iWay

iWay is a framework that provides over 280 adaptors to applications, databases, etc.. Its full title is the **iWay Adaptive Framework**.

The iWay Adaptive Framework was originally developed by the company **Information Builders**. In 2001, Information Builders decided to spin-off the iWay Adaptive Framework product as a subsidiary company - called **iWay Software**.

The two companies, iWay Software and Information Builders, remain closely linked. You tend to find that the iWay consultants and trainers work for Information Builders.

Web Sites

You can read more about iWay Software and Information Builders at the following Web sites:

www.iwaysoftware.com

www.informationbuilders.com

Available Adaptors

You can see the list of available adaptors by visiting the iWay Software Web site. The direct URL is:

www.iwaysoftware.com/products/iWay_adapter_List.html

iWay Version

OVBPI has been tested against iWay version 5.5.

Given the way OVBPI integrates with iWay, it should work with any iWay version that supports the output of data to files.

How does iWay compare to openadaptor?

The iWay Adaptive Framework (iWay) is essentially another openadaptor. Both products address the same requirement, which is to enable developers to quickly and easily plug components together to allow data to flow between heterogeneous data sources.

Both iWay and openadaptor provide the ability to read/write databases and files. However, iWay can provide many more adaptors for integrating the major business applications. iWay has off-the-shelf adaptors for applications such as SAP, PeopleSoft, JD Edwards, and many more.

One significant difference between iWay and openadaptor, is that iWay costs money. If the customer wants an adaptor to SAP they have to buy it.

For many large SAP sites, this may not be a problem as they may already have iWay.

## Is iWay a Replacement for openadaptor?

No, the OVBPI event handler is still built around openadaptor, and openadaptor remains as the underlying mechanism for receiving business data events.

By offering an integration with iWay, this allows you to integrate business events from a wider set of data sources and applications. But these events still come into OVBPI through openadaptor, as shown in

Overall Architecture

The iWay integration is achieved by OVBPI offering an iWay adaptor for openadaptor, as follows:

Figure 51 OVBPI/iWay Integration
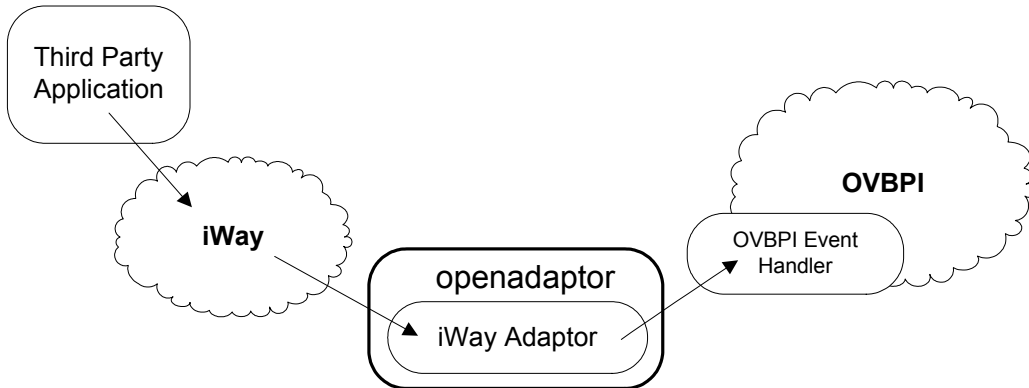

where:

- Business events come from the third party application(s) and pass into iWay.

- iWay takes these events and passes them to openadaptor

- openadaptor then passes the events into OVBPI through the OVBPI Event Handler

iWay Knowledge

You are probably wondering how much you need to know about iWay? The answer is...virtually nothing.

If you think about the overall architecture diagram, there is a clean division between OVBPI/openadaptor knowledge and iWay knowledge.

Figure 52 The iWay/OVBPI Knowledge Split
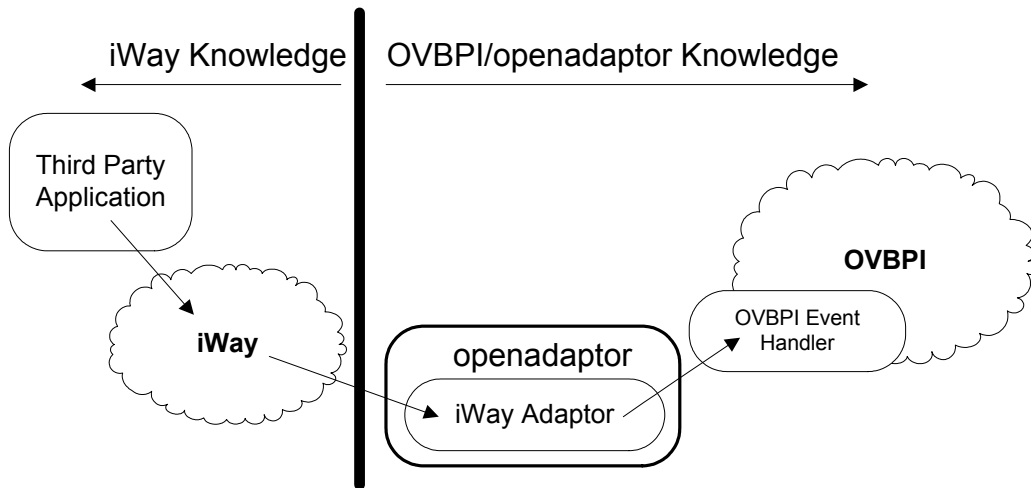
Configuring the iWay system to capture the third party application events requires iWay knowledge, and possibly some amount of knowledge about the third party application involved. This is clearly the job of the iWay consultant, or someone trained in iWay configuration.

Configuring the iWay adaptor within openadaptor to capture the events from iWay and send these into OVBPI is the job of the OVBPI consultant. You just need to understand how the event is passed from iWay to your adaptor - and this is discussed in detail in Architecture on page 355.

iWay Basics

Although you do not need in-depth knowledge of iWay, this section introduces the main iWay tools, and some of the basic iWay terminology.

iWay Tools

The iWay product consists of the following set of tools:

- iWay Adaptor Manager

  You can think of this as the iWay engine. This needs to be running for iWay to work.

- iWay Adaptor Console

  The iWay Adaptor Console is a Web console that runs within a Web browser. The iWay Adaptor Console allows the iWay developer/ administrator to configure adaptors, start and stop adaptors, monitor adaptors, troubleshoot adaptors, and more.

- iWay Adaptor Transformer

  Within an adaptor, you may want to transform the data in some way. For example, you might want to select certain attributes from the data record, or maybe perform some mathematical calculations. The iWay Adaptor Transformer lets the iWay developer visually build the transformations and make them available for the adaptors to use.

- iWay Adaptor Designer

  The iWay Adaptor Designer is a GUI tool that allows the iWay developer to piece together complex adaptors. It allows the iWay developer to visualize the steps within the adaptor. The Adaptor Designer is a bit like the openadaptor AFEditor.

- iWay Application Explorer

  The iWay Application Explorer is a GUI tool that allows the iWay developer to explore the meta-data within the customers application software. For example, an iWay developer can use the Application Explorer to determine what features and functions are available from an SAP installation.

Terminology

Although configuring iWay is a job for an iWay consultant, it is useful to have an appreciation of some of the concepts. iWay is a adaptor framework, so the concepts are fairly similar to the concepts of openadaptor. However, as you might expect, iWay have their own terminology.

Let's take a look at some of the basic iWay terms, and relate these to what you know from openadaptor.

Listener

In openadaptor, there is the concept of a Source. In iWay, they call it a Listener. A Listener is configured to listen to a data source.

Agent/Transformer/…

Having configured a Listener, you may want to pass each record through some sort of data transformation - maybe perform a database lookup to read in additional data attributes, or carry out some mathematical calculations on the data attributes. In openadaptor terminology, you pass the data through a Pipe.

In iWay, you have the concept of a Pipe, however, iWay does not call them Pipes. Instead, iWay divide Pipes into a number of different categories, such as: Agent, Preparser, Reviewer, Input Validator, Input Transformer, Output Transformer, and Output Validator. So, for example, the iWay developer might configure a Database Listener to pass its data to a Transformer that is configured to transform the data.

Where it gets a little confusing for the first-time user, is that if you wish to perform something like a transformation on some data you receive from a Listener, there is no rule in iWay to say that you must do this transformation within a Transformer. The iWay developer could, for example, configure the transformation within an Agent, and achieve the same result.

But, whether it is an Agent, a Transformer, Validator, etc. you can think of it as a Pipe - it is performing some kind of action on the data.

Emitters

iWay has the concept of an Emitter, which maps nicely to the openadaptor Sink.

However, within iWay you often do not need to configure an Emitter. This is because the Pipes (Agents, Transformers, ...) can be configured to output the data record directly. In other words, an Agent (Transformer, ...) can act both as a Pipe and a Sink.

A Listener can also output its data records directly. That is, a Listener can be both a Source and a Sink.

So, the concepts of iWay and openadaptor are similar, but within iWay the division of tasks is perhaps not always as strict.

Summary

The iWay Adaptive Framework enables developers to quickly and easily plug components (Listeners, Agents, Transformers, Emitters, ...) together to build adaptors that allow data to flow between heterogeneous data sources.

OVBPI provides an adaptor that enables you to connect iWay to openadaptor, and hence, to OVBPI. Thus allowing you to receive business data events from any of the data sources supported by iWay.

Architecture

So how is iWay passing the business data, and which openadaptor component is handling this data?

File Level Integration

iWay offers adaptors to a wide range of applications and data formats. One of these adaptors is the File Adaptor.

When configuring the iWay File Adaptor to output data, you specify an output directory. As data comes from the third party application, the iWay File Adaptor writes each batch of data records as a new file within this output directory. In other words, the adaptor produces multiple output files within a single directory.

openadaptor is able to read data from a file, using the `FileSource` component. However, this component is only able to read data from a single input file.

OVBPI provides a set of enhancements to openadaptor. One of these enhancements is the Source component called `FileRollSource`. The `FileRollSource` component can be given a directory name and it reads the files as and when they appear in that directory.

The more detailed OVBPI/iWay architecture diagram looks as follows:

Figure 53 OVBPI/iWay Integration
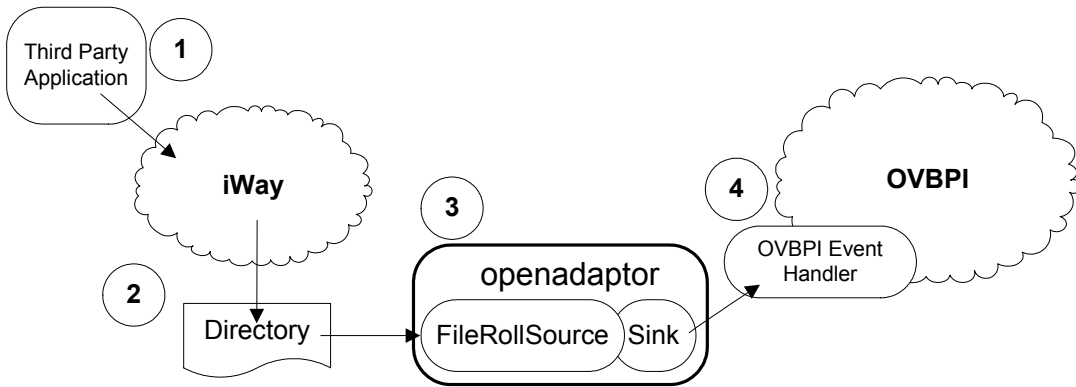
where:

1. The external application (for example, SAP) has a number of data events that it can emit. The iWay consultant sets up an iWay adaptor to listen for the particular event that you require.

2. When one of these events come from the third party application, the iWay File Adaptor writes this event out as a file in the configured output directory.

3. An openadaptor adaptor is polling this directory (using the FileRollSource component).

   As and when new files appear, the FileRollSource component processes these files and sends the event(s) into the OVBPI Event Handler.

4. The events go into the OVBPI Engine and progresses the flow(s).

To configure an iWay/OVBPI integration, you discuss with the iWay Consultant the event(s) that you are interested in receiving, and they configure their iWay File Adaptor to write those events as files in a known directory. You configure an openadaptor adaptor, using the FileRollSource component, to read and process these files as and when they appear in the iWay output directory. Your adaptor then sends the data into the OVBPI Engine as business data events.

Multiple Event Definitions

When you have a number of different types of event coming from your external application, and you wish to send these into OVBPI, you have a few options available. You either configure separate iWay and openadaptor adaptors for each event, or combine these in some way to handle all the multiple event types. It really depends on how you, and the iWay consultant, choose to set things up.

Let's look at a few possible scenarios.

Scenario 1
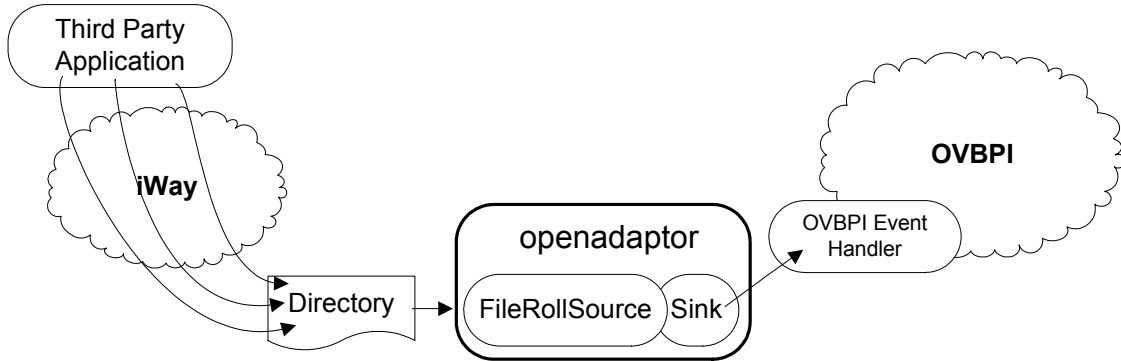
Figure 54 Multiple Adaptors
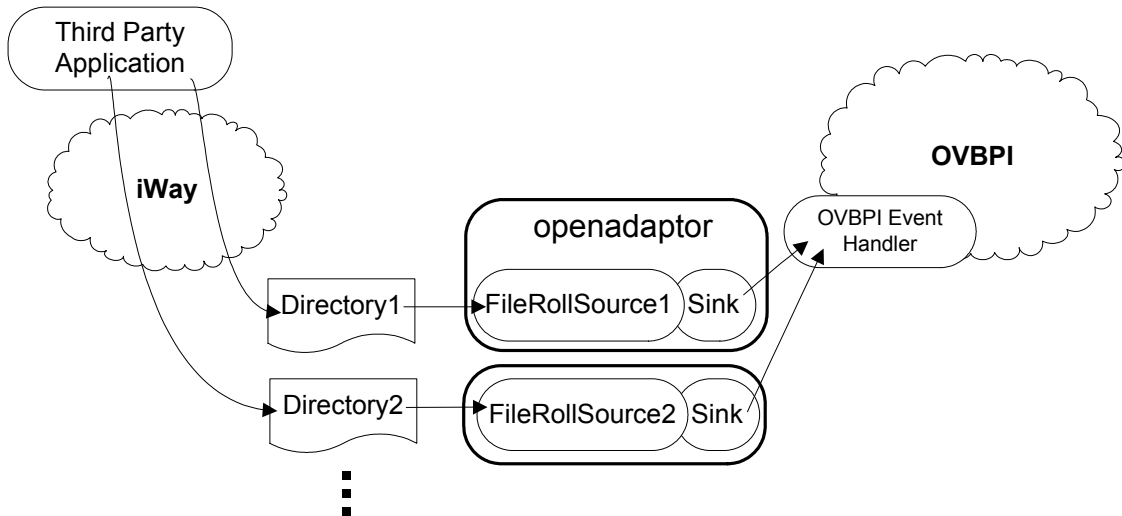
where:

- For each type of application event, the iWay consultant configures a separate iWay adaptor to write the events to a unique directory.

- You configure a separate openadaptor adaptor (using the FileRollSource component) for each of these directories.

This scenario is probably the easiest way to set things up. It can feel like you are creating too many adaptors, but it is actually quite normal and a very reasonable architecture to use. You have an iWay/openadaptor pair of adaptors for each event type.

Scenario 2

**Figure 55 Multiple iWay Adaptors**

where:

- For each type of application event, the iWay consultant configures a separate iWay adaptor and has them all writing their events to the same output directory.

  You just need to ensure that each event type is easily identifiable. The exact way you would do this depends on the format of the data being output by the iWay adaptor. The output might be using Comma Delimited File (CDF) format or XML format.

- You configure one openadaptor adaptor (using the FileRollSource component), and have it send in the events according to their type.

Scenario 3

Figure 56 Single iWay Adaptor
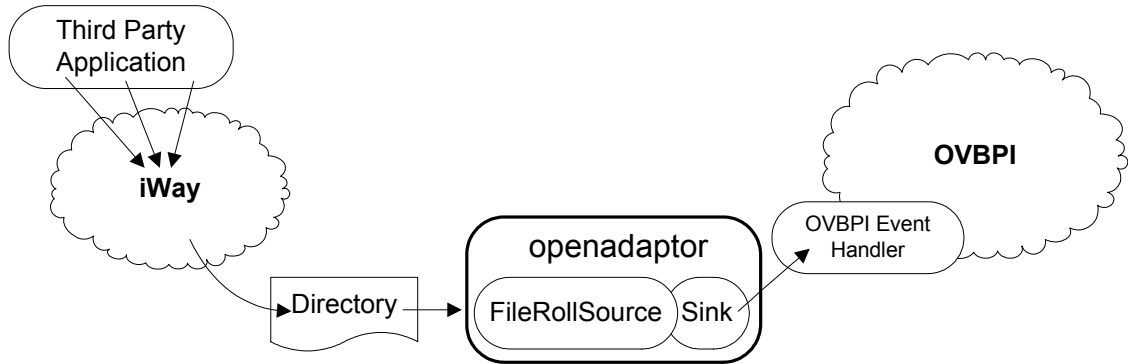
where:

- This scenario is similar to Scenario 2 on page 358. The difference is that here the iWay consultant may chose to handle all events within the one iWay adaptor.

  This is obviously a choice that the iWay consultant would make.

  You would still need to ensure that each event type was easily determined within the iWay event record - just as you do for Scenario 2 on page 358.

Choosing a Scenario

As far as you are concerned, it does not really matter how the iWay consultant does it all, just so long as you know where the output data is. Whether it is in separate directories, or a single directory, it is easy to configure openadaptor adaptors, using the `FileRollSource` component, to handle the data and generate the necessary OVBPI events.

Configuring the Events

So how does it all work?

Suppose you are wanting to define a business flow, and part, or all, of the flow requires data feeds from an external application (say for example, SAP).

Defining The OVBPI Events

You need to talk with the iWay consultant and ask them what events they are able to get for you and what data these events may contain. The iWay consultant may be able to use tools such as the iWay Application Explorer to help them work out what events the external application is capable of emitting. The iWay consultant may also work directly with the customer's IT applications people to understand more about the data.

In other words, if you need a "New Order" event, you need to talk with the iWay consultant and see if they can set this up for you, and what data attributes the event are to contain.

Once you agree on the set of events that can be produced using iWay, and their data attributes, you are able to go into the OVBPI Modeler, model your flow, and define these as business events.

iWay File Adaptor Data Format

You need to talk with the iWay consultant and have them produce all event data in flat files. The question then is, what format should the data be within these files?

By default, the iWay File Adaptor outputs data in XML format. This is ok because you can configure the openadaptor `FileRollSource` component to read XML (refer to XML Format Data on page 279). However, you might feel more comfortable working with files that contain their data in comma delimited format (CDF). The iWay File Adaptor can be configured to produce its output in CDF data format.

Accessing the Directory

The iWay File Adaptor writes its output as files within a configured directory. You need access to this directory (or directories, if there are multiple iWay adaptors involved) so you obviously need to agree with the iWay consultant where this directory is on your network.

If you are running OVBPI and iWay on the same server, then it is easy for your `FileRollSource` component to access the iWay output directory. However it is more likely that OVBPI and iWay are on different servers. So, the question is, where are you running the openadaptor adaptor relative to these iWay output files?
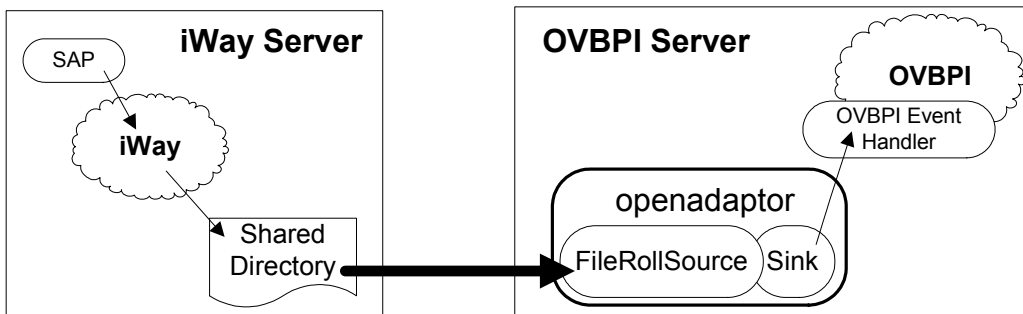
FileRollSource running on the OVBPI server

If you are running the openadaptor adaptor (the adaptor using `FileRollSource`) on the OVBPI server then you need access to the directory containing the files that are being produced by iWay.

You could achieve this by simply sharing the directory across your network.

For example:

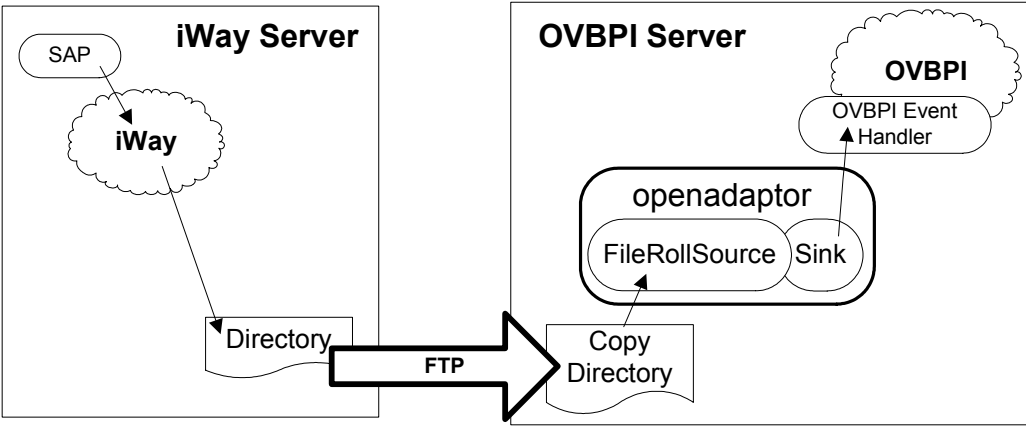**Figure 57 Sharing Directories - using a Network Share**

Another possibility is that you use something like ftp to transfer the files across to a directory on the OVBPI server and read them from there.

For example:

Figure 58 Sharing Directories - using FTP

It really depends on the customer environment as to how you choose to set this up. Certainly, the "network share" approach is the easiest.

FileRollSource running on the iWay server

If you are running the openadaptor adaptor (the adaptor using `FileRollSource`) on the iWay server, you need to install openadaptor on this iWay server (see Installing openadaptor Standalone on page 417).

Once you have the necessary openadaptor software and components installed on your iWay server, you configure and run your adaptor (the one using the `FileRollSource` component) and have it send the event(s) across the network to the OVBPI Event Handler.

For example:

Figure 59 Running openadaptor on the iWay Server
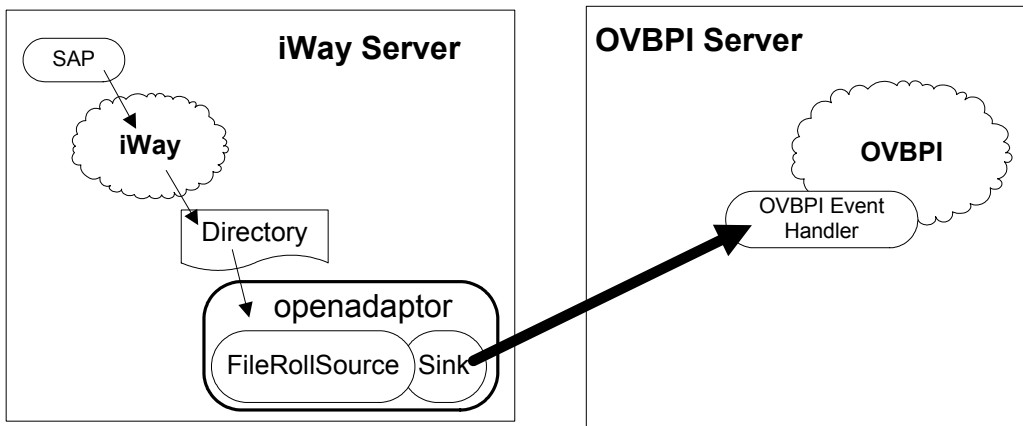
9 SAP Integration

This chapter gives a brief overview of SAP and SAP terminology. The chapter then looks at the different ways you can integrate OVBPI with SAP.

To implement an integration with SAP requires that you work with an SAP consultant or partner. The SAP consultant/partner is needed to understand and implement the integration on the SAP side of things. This chapter aims to help you understand the numerous terms and techniques that an SAP consultant may use, and give you an insight into the world of SAP.

A Brief History of SAP

In 1972, five former IBM employees founded the German company SAP (Systems, Applications and Products in Data Processing). Their vision was to develop and market enterprise software which would integrate all business processes.

SAP's first product offering consisted of two application packages - a financial package and a materials management package. This product was called System/R.

The next version of System/R was called R/2. R/2 introduced some more application packages, but still ran primarily on IBM mainframes.

With the R/2 release, SAP developed their own report writing language. This language was called "Allgemeiner Berichtsaufbereitungsprozessor" (**ABAP**) - which is German for "generic report preparation processor". ABAP grew into a complete programming language and was later renamed Advanced Business Application Programming. ABAP, with its object oriented extensions, is a very powerful language, with a syntax that is somewhat similar to Cobol. Indeed, ABAP soon became the language of choice for the SAP developers and nowadays most SAP applications are written in ABAP.

The third release of System/R, **R/3**, was a very significant release for SAP. The R/3 release implemented a client/server architecture. SAP applications could now extend to the emerging client/server market and were no longer restricted to mainframes. SAP introduced a number of new applications and their R/3 release became their Enterprise Resource Planning (**ERP**) solution, addressing areas such as manufacturing, supply-chain, distribution, financial, customer management, human resources and more.

With customers installing more of the SAP suite of applications, the need for application integration became more important. Generally speaking, these integrations were carried out on a point-to-point basis. That is, each integration required custom development to enable the applications to share and communicate data.

One of the earliest mechanisms introduced by SAP to help with integration of SAP applications was the Intermediate Document (**IDoc**). An IDoc is a data container with a format defined and documented by SAP. An IDoc can be held in a database and/or sent as a text file. SAP introduced the idea that one SAP application could be configured to write out information to an IDoc, and

another application could be configured to read this IDoc - hence information can be transferred from one application to another. IDocs are still very much a part of SAP and used extensively.

In release 3.0 of R/3, SAP introduced Application Link Enabling (**ALE**). In many ways, ALE was SAP's first attempt at providing a middleware technology that would make it easy to configure applications to pass data between themselves, and to third party systems. ALE allows you to configure SAP to monitor a business object and send out notifications as and when the business object changes. ALE sends this data within an IDoc, and you are able to configure transformations to this data. So ALE is a powerful integration tool.

SAP then looked at moving their applications to run within an application server. They developed the SAP Web Application Server. Originally this app. server only ran ABAP programs. The app. server was later made J2EE compliant - thus adding support for Java.

SAP then decided to introduce their own middleware bus for integration between their applications, and to third party applications. This middleware bus is called the Exchange Infrastructure (**XI**). SAP then wrote the Business Process Management (**BPM**) product to orchestrate the messages going across the XI bus.

At this point SAP decided to reposition its offerings under the banner **NetWeaver**. NetWeaver is SAP's new integration platform; effectively they've re-branded their J2EE app. server as the NetWeaver platform.

NetWeaver is a suite of SAP products which are provided under a single license to an Enterprise. Except for SAP's R/3 ERP, SAP's flagship product, all the other major business applications from SAP are included in NetWeaver.

The release of NetWeaver is considered as a strategic move by SAP for driving enterprises to run their business in a single platform, which is the SAP App server.

SAP Terminology

Here are some important terms you may hear when talking to anyone familiar with SAP.

IDoc (Intermediate Document)

An IDoc is a data container with a format defined and documented by SAP. An IDoc can be held in a database and/or sent as a text file. IDocs are one of the original mechanisms used for exchanging data asynchronously between two or more SAP applications.

Basically, if one application can write out information to an IDoc, a second application can then read in this IDoc.

Typically an IDoc contains the multi-line data for a single transaction.

An IDoc can be generated at any point within an SAP transaction (or work flow; see Work Flow on page 370). For example, during a shipping transaction, an IDoc may be generated that includes the data fields required to print a shipping manifest.

After a user performs an SAP transaction, one or more IDocs may be generated and made available for passing to other applications.

Application Link Enabling (ALE)

ALE provides SAP customers with a program distribution model and technology that enables them to interconnect programs across various platforms and systems. ALE is available in R/3 Version 3.0 or greater.

You can configure ALE to watch for certain events to occur within SAP. As these events occur, ALE can send the event details (typically as an IDoc) to any configured output - such as a file, database, socket, another SAP system, or to a third party external system. You can also configure transformations for the data content.

ABAP (Advanced Business Application Programming)

ABAP is a programming language created by SAP. The syntax of ABAP is somewhat similar to COBOL.

Most of the application code of SAP is written in ABAP, and so any user extensions are typically written using ABAP.

NetWeaver

NetWeaver is SAP's new integration platform; effectively they've re-branded their J2EE app. server as the NetWeaver platform. The NetWeaver app. server supports both ABAP and Java application code.

NetWeaver is a suite of SAP products which are provided under a single license to an Enterprise. NetWeaver does not include R/3 and so this may be purchased separately.

XI (Exchange Infrastructure)

XI is SAP's middleware offering. It is SAP's mechanism for allowing integration between SAP applications and with third party applications.

BAPI (Business Application Programming Interface)

SAP applications are written in ABAP using object-oriented programming techniques. For each application, SAP has defined a set of central Business Object Types (similar to Java class definitions) and for each of these types, a set of methods. These methods are called BAPIs.

External access to a Business Object's data is only possible through the defined BAPIs. For example, the functionality that is implemented with the SAP Business Object Type `Material` includes a check for the material's availability. Thus, the Business Object Type `Material` offers a BAPI called `Material.CheckAvailability`.

SAP provided the first BAPIs for customers, and external providers, in R/3 Release 3.1, enabling them to integrate their software components. The number of BAPIs is increasing with each SAP Release.

RFC (Remote Function Call)

When people talk about BAPIs they often mention the term RFC (Remote Function Call). This is because BAPIs are implemented as SAP remote function call (RFC) modules.

Work Flow

When an R/3 system is implemented for a customer, the SAP consultant tailors the functionality of the application to meet the customer's needs. So for example, the process of "creating a new sales order" can be tailored for your company. The SAP consultant is able to chain together the necessary screens, in the right order, to lead the user through the steps for creating a new sales order. This "series of steps" is called a work flow.

So when an SAP consultant sets up a new system for a customer, they set up all the work flows, tailored for that customer.

User Exit

Work flows have the ability to make calls out to user provided code. These calls-outs are called user exits. Hence, a developer can add custom code into virtually any work flow of the SAP system.

For example, you might provide a user exit that writes out the order number to a file whenever a user creates a new sales order.

To install a user exit, you need to know the SAP code module that is to call your code. You must then ask the SAP administrator for the software access key to unlock this module. You can then embed your user exit into this module.

Custom user exit code is usually written in ABAP.

Basis Administrator

When people talk about the SAP Administrator they typically refer to them as the `Basis Administrator`. The term `basis` is used to refer to the central core of SAP R/3. Things such as the keys to unlock SAP code modules, is part of the SAP basis.

Integrating with SAP

When you consider how to integrate SAP with OVBPI, there are three main questions that need to be considered. These being:

1. How to get the data out of SAP?
2. Do you need to transform this data in any way?
3. How best to send this data into OVBPI?

But question 1 in this list can take some time to sort out. Any decision about how to get data out of the SAP system requires discussion and agreement between you, an SAP consultant and (of course) the customer.

Business Monitoring not BPM (Push vs Pull)

When you first start talking to SAP consultants about accessing data within SAP, they are typically used to addressing this need within a BPM (Business Process Management) type of scenario. That is, they are quite used to an integration model where a step within a business process, or application, needs to send some data in to SAP, or request some data from SAP. In these cases, the application can typically make the appropriate BAPI/RFC call to (for example) "Raise a Purchase Order", or "Get Order Details".

So, an SAP consultant can often start by seeing the OVBPI integration as simply a call to a BAPI to "pull" the data out of SAP. However, an OVBPI integration has different requirements. OVBPI is "monitoring" the SAP business process, not orchestrating it or taking part in it. So OVBPI is typically not wanting to make synchronous calls to run specific functions within SAP, instead, OVBPI needs to be told as and when certain events happen within SAP. For example, if OVBPI is monitoring the raising of Purchase Orders, SAP needs to send the appropriate purchase order details out to OVBPI as and when each new purchase order is raised. That is, SAP needs to "push" out the details as they happen.

If you were to use the BAPI/RFC approach to monitor the raising of Purchase Orders, you would need to find a BAPI call that was able to give you something like "The set of Purchase Orders that have been raised since the last time you asked for them". You would then need to periodically call this BAPI and "pull" the information.

Integration Options

Let's go back to the three main questions that need to be considered when integrating SAP with OVBPI. These being:

1. How to get the data out of SAP?

2. Do you need to transform this data in any way?

3. How best to send this data into OVBPI?

Figure 60 shows these three main areas of integration - SAP, Data Transformation and OVBPI - and lists some of the possible options available to you.

Figure 60 Possible Options for the Areas of Integration
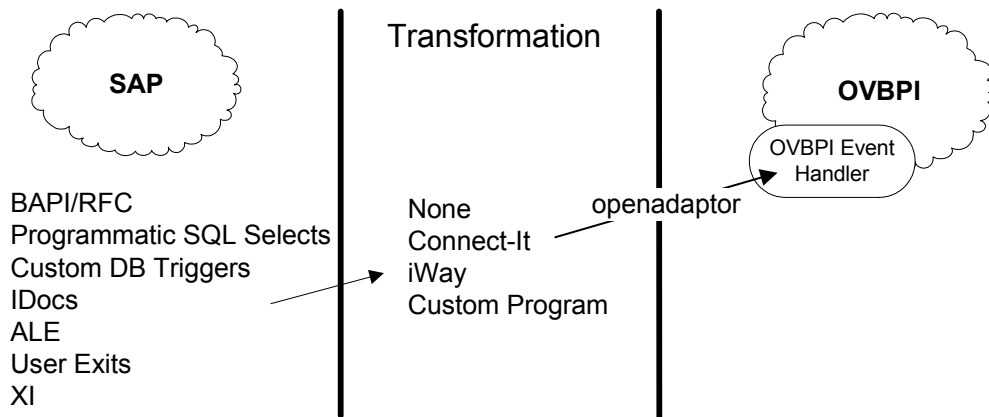
# SAP Product Knowledge

A customer's SAP installation can be a complex beast, and you need to work with a qualified SAP consultant or partner. The SAP consultant is the one who can make the decisions about the appropriate mechanism for retrieving the business data, and it is the SAP consultant that is able to then implement that part of the integration.

So Figure 60 on page 372 can be redrawn as shown in Figure 61 on page 373:

Figure 61 SAP Expertise Required
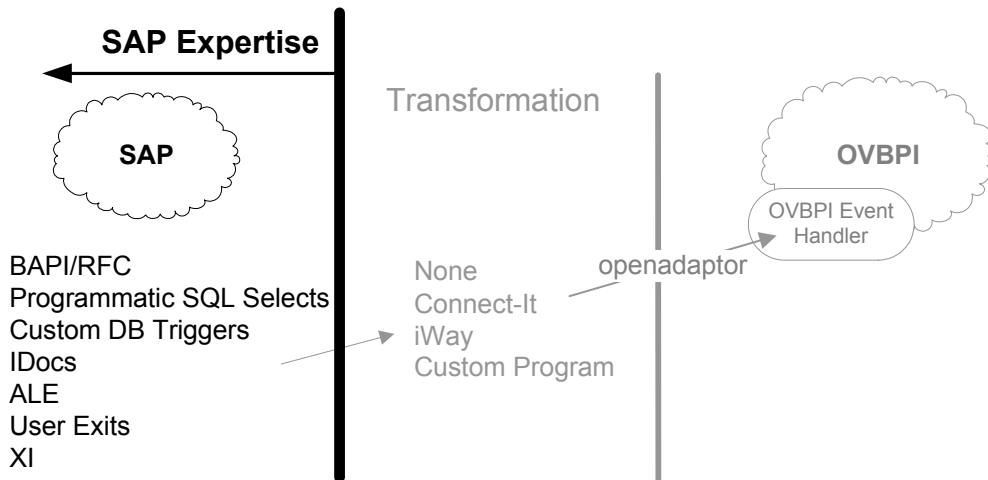
You need to partner with an SAP consultant, and have them decide and implement the necessary mechanism for generating the business data from the SAP system.

Once you have the business data coming out of SAP, you can then decide how to get that data into OVBPI. If the data needs transforming you could either use Connect-It, involve an iWay consultant, or you might decide to handle the work yourself by writing a custom transformation program. Once the data has been transformed, you are then able to feed this data into OVBPI using openadaptor and the OVBPI Event Handler.

Which is the Correct Mechanism?

When you look at Figure 60 on page 372 you may start to wonder whether any particular mechanism or combination of mechanisms is going to be the best. The answer is.....it depends. All the options available for getting data out of SAP (IDocs, ALE, User Exits, etc.) can provide the information you require, however, the SAP Basis administrator may have a preference for the kind of mechanism they use. For example, they may prefer to use User Exits, or they may already use ALE and see that as their integration mechanism.

The mechanism you choose also depends on the exact make up of the customer's SAP installation - which applications they have installed, which version of the applications, etc.

There is no right answer. This is why you need to partner with an SAP consultant and work with them, and with the customer, to decide which strategy to use.

The following sections discuss the various integration mechanisms in more detail, grouping them as either "push" or "pull" integrations.

Pull Integrations

Implementing a pull mechanism can work, however you need to make sure that you do not miss any events, or unnecessarily send events more than once.

Let's consider the two main pull mechanisms available.

BAPI/RFC

Using BAPIs (RFCs) is where the SAP consultant typically writes an ABAP program to periodically call a BAPI to get, for example, all the new Purchase Order records. This might look as shown in Figure 62.

**Figure 62 Using BAPI/RFC**
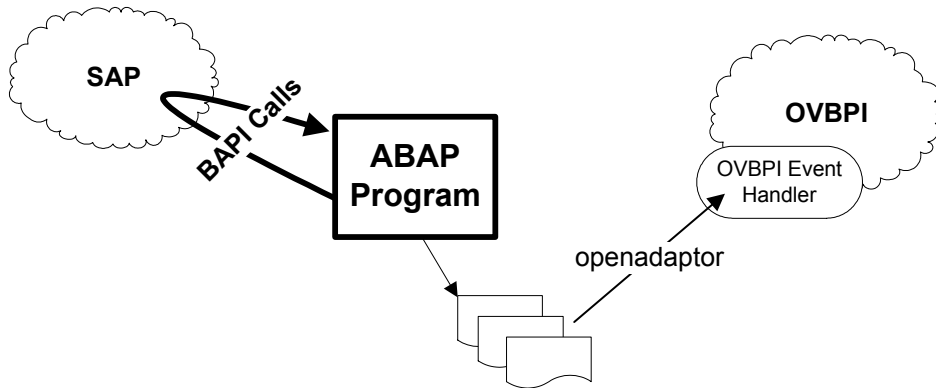
This mechanism can be difficult to get right. You need to make sure that your ABAP program does not miss any Purchase Orders, and you typically need to add program logic to ensure that you don't keep getting back the same Purchase Order with each BAPI call.

Using BAPIs gives you a "pull" integration, where the program is pulling the data from SAP at regular intervals.

Programmatic SQL Selects

An SAP consultant might suggest that they write a custom program (probably in ABAP) that polls one or more tables in the SAP database looking for new and/or updated data records. As new data is written or updated in these tables, the program selects that data and writes it out to a file or somewhere else. This might look as shown in Figure 63.

Figure 63 Using SQL Selects
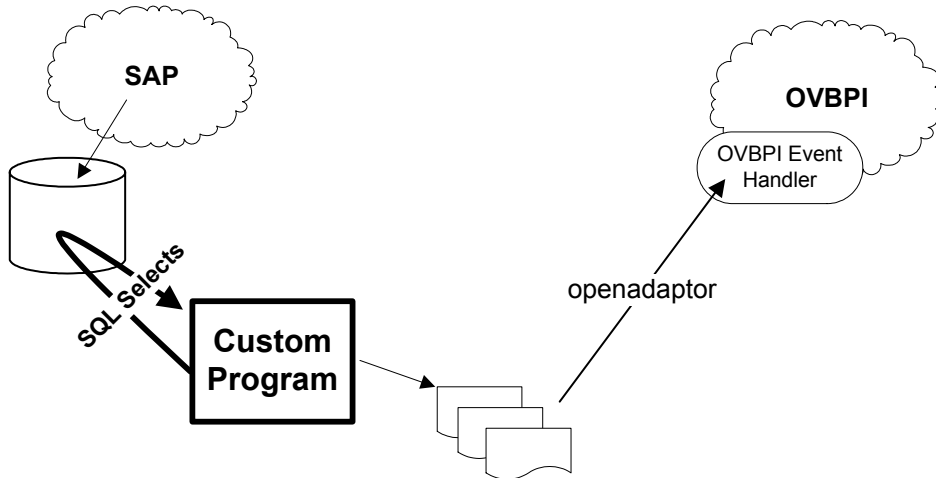
This mechanism can be quite difficult to get right. The select statement has to poll the table asking for any new records since the last time it polled. You would need to include some logic to ensure that you never missed any records.

If the table you are monitoring has records that are being updated, then your code needs to be able to detect this and determine what attributes have changed. This can be difficult to implement.

The other potential issue with this option is that the SAP database has many thousands of tables and so finding the exact table(s) for your select statement(s) may be difficult.

There is also the issue that this is a "pull" mechanism, where the program is trying to pull the data from the database at each polling interval.

Push Integrations

Implementing a push mechanism can tend to increase the initial effort required to configure the SAP side of things, but it does produce a very clean and efficient integration.

Let's consider the main push integration mechanisms available.

Custom DB Triggers

By placing database triggers on the underlying SAP database, you are able to capture the changes to the data and have the trigger write these out to a secondary buffer table. You can then configure the standard openadaptor database source to monitor this buffer table and send the business data events into OVBPI.

This trigger mechanism works as shown in Figure 64.

Figure 64 Using Database Triggers
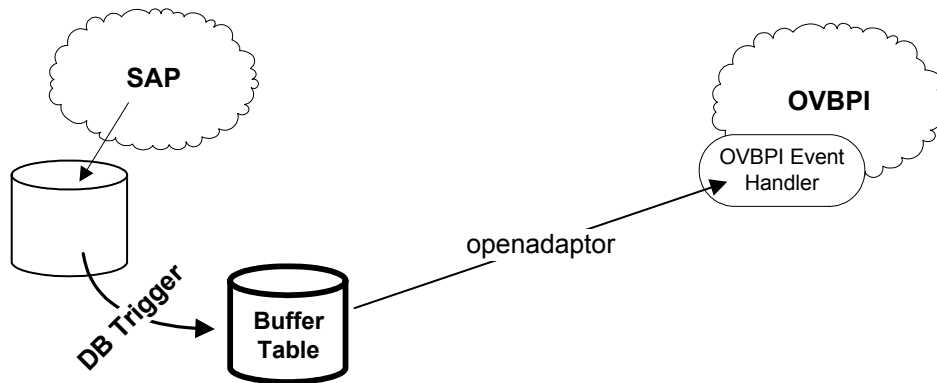
Although this mechanism works very well, it is fairly unlikely that any SAP Basis administrator is going to allow you to add triggers to their SAP database. Typically, the SAP Basis administrator will not let you alter the SAP database for fear of upsetting their SAP support contract.

The other potential issue with this option is that the SAP database has many thousands of tables and so finding the exact table(s) on which to place your trigger(s) can be difficult.

IDocs

The various work flows within SAP can be configured to write out IDocs as various transactions occur. Indeed, it may be that the information you need to monitor a flow is already being written out as a series of IDocs. If this is the case then you just need to find where they are.

The chances are that the IDocs being written already by the SAP system are not catching the exact information you require for monitoring a flow. In this case, the SAP consultant may be able to configure the applications to produce the necessary IDocs.

An IDoc typically contains the data from a single transaction and writes this as a multi-line file. You need some mechanism to transform this IDoc into one, or more, business data events. This is where you could use products such a Connect-It or iWay.

Both Connect-It and iWay have tools that allow you to interrogate the IDoc and pull out the necessary data, formatting this in a way suitable for passing to openadaptor. Alternatively, you might decide that the format of the IDoc is such that you can write a custom program to extract the necessary data and pass this into openadaptor.

This mechanism might look as shown in Figure 65.

Figure 65 Using IDocs with Connect-It, iWay or Custom Code
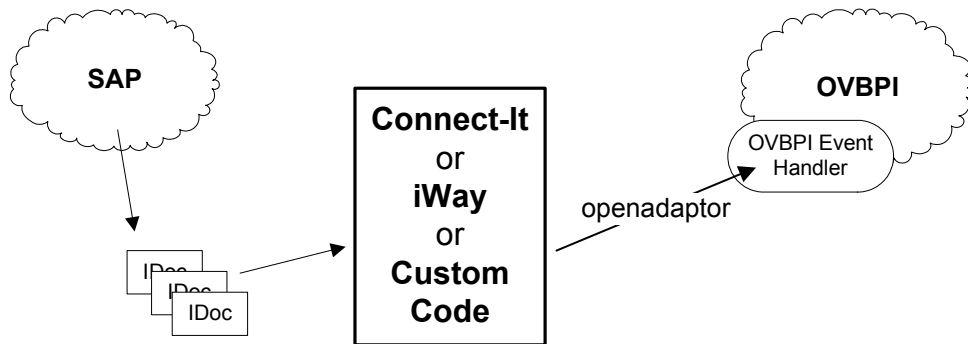
ALE

Application Link Enabling allows the SAP Basis administrator to monitor one or more SAP business objects. When a new instance of a business object is created, and/or an instance is updated, ALE can be configured to send an IDoc to a destination PORT. In ALE, a destination PORT can be any output device, such as a file or a socket. It is also possible to configure ALE to carry data transformations before writing out the IDoc to this PORT.

Your ALE configuration might look as shown in Figure 66:

Figure 66 Using ALE
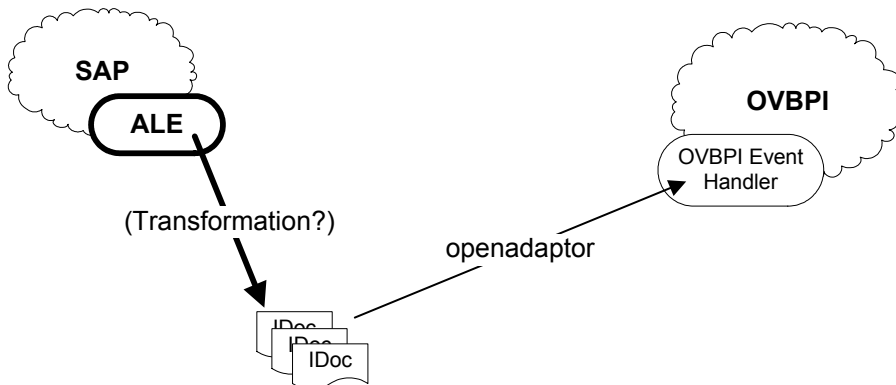
You may be able to configure the necessary data transformation within ALE such that the resultant IDocs are in a form readable by openadaptor.

Alternatively, you might decide to have ALE simply produce the IDocs and then chose to use something like Connect-It, or iWay, to do the necessary transformations, before then passing the data to openadaptor. This architecture might look as shown in Figure 67 on page 380:

Figure 67 Using ALE with Connect-It or iWay
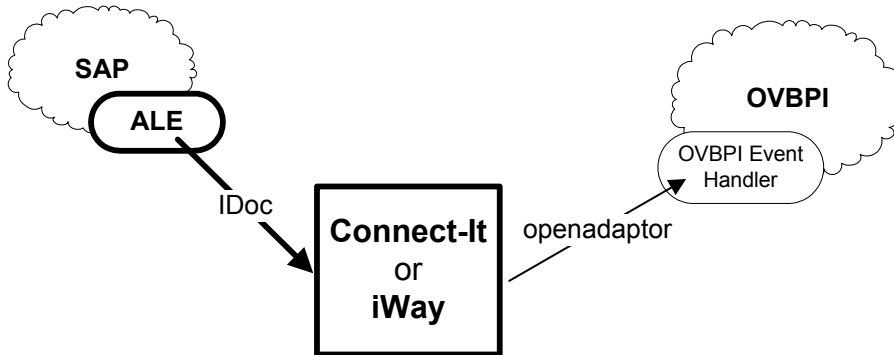
Another alternative is to use ALE to send the data directly to a socket on the OVBPI machine. You could then have a custom program, listening on the socket, that does any final transformations before invoking openadaptor to pass the data into OVBPI. This architecture might look as shown in Figure 68:

Figure 68 ALE Sending to a Socket on the OVBPI Machine
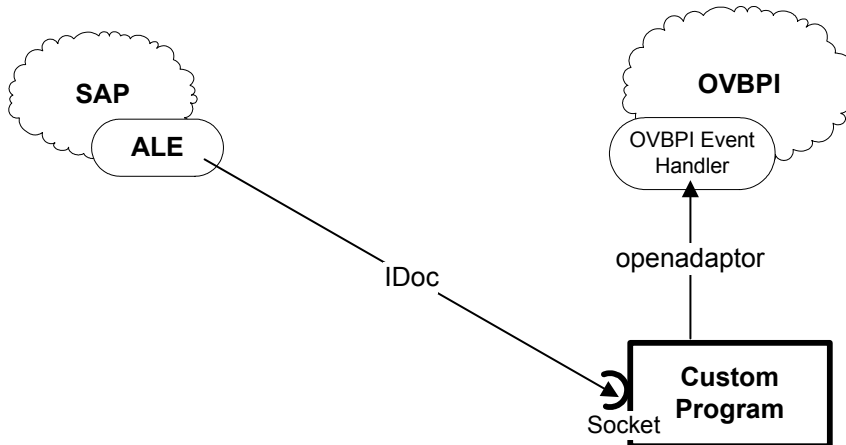
where:

- You may choose to do some data transformation within ALE.

- Your custom program on the OVBPI machine can obviously do any transformations you require.

- Your custom program invokes openadaptor and acts as a `LocalSource`.

  This allows your custom program to pass the input data record directly to the `SocketSink` and hence directly to the OVBPI Event Handler. If necessary your adaptor can carry out further filtering or data transformations.

Using ALE can provide for a very efficient integration.

User Exits

User exits allows you to embed your own code (typically written in ABAP) within a user's SAP work flow. For example, you might wish to embed a user exit within the "Create new sales order" flow, and capture the new sales order id, the customer id and total order value, whenever a user creates a new sales order.

Your user exit code can output the data as best suits your needs. Figure 69 shows the architecture where the user exit code writes out each new sales order as a separate comma delimited file.

Figure 69 User Exit Code Writing to Files
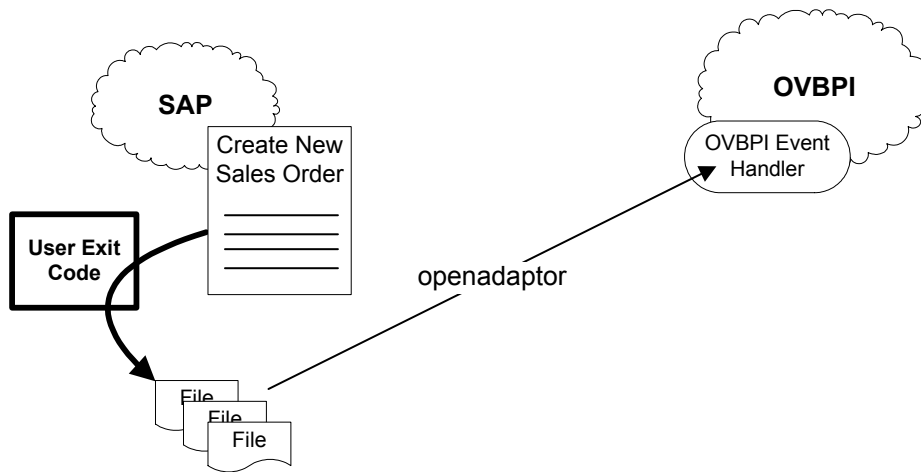
You can have more than one user exit for a work flow so that you can capture whatever data you require. You can output the data to any location you need, using whatever format is appropriate.

In Figure 69, you could configure the `FileRollSource` and point it at the directory in which your user exit code is writing its files. You would just need to make sure that the user exit code creates the file without any filename extension, and when the file is complete, renames this file to have a file extension (for example, `.txt`). Your `FileRollSource` just needs to be configured to only process files ending in `.txt`.

To write and embed user exit code, the SAP consultant/developer needs to be able to write the ABAP code and identify the place(s) within the SAP work flow to insert this code. The SAP Basis administrator then needs to provide the necessary software unlock keys to allow the embedding of these user exits.

XI

If your customer has SAP XI, or NetWeaver (including XI), then the SAP consultant may choose to implement the sending of the SAP data across the XI bus. This involves the sending and receiving of XML data across the XI bus.

XI is SAP's middleware bus for exchanging messages, and you can think of XI as being something like a TIBCO, WebMethods or BizTalk message bus.

The SAP consultant may write a specific client to accept messages destined for OVBPI, or possibly use a file-based XI client to accept the messages destined for OVBPI and simply write these as files into a directory. You could then configure the `FileRollSource` to monitor this directory and send the data into the OVBPI Event handler.

10 Further Topics

This chapter looks at additional topics to do with the Business Event Handler, including some debugging tips.

Debugging Your Adaptor Network

When people first start trying to feed in events to drive their flow, you hear some common questions:

> "How can I see whether the event actually went in to OVBPI?"
> "How can I see what order the events arrived in?"
> "Is there any way to list the events that the Business Impact Engine received?"

The answer to all of these questions is:

> You can configure the `BIAEngineAdaptor` to log all events.

▶ This is something you might want to consider for when you are developing the flow, but you would **not** want to do this on a production system as it would adversely affect the performance.

When you are in **development mode** you might want to configure the BIAEngineAdaptor to log all received events to a flat file. This would allow you to send in events, see if the Business Impact Engine processed them, and if there were some problems, you would be able to look through the log file and see all the events that had been processed and the order in which they had arrived.

By configuring the BIAEngineAdaptor to log all events to a log file, you are able to capture every event that comes through your adaptor network.

All you need to do is configure the BIAEngineAdaptor to have a second sink and configure this as a `FileSink` which writes its output as a comma-delimited file.

Your adaptor network would look something like this:

Figure 70 Logging All Events To A File
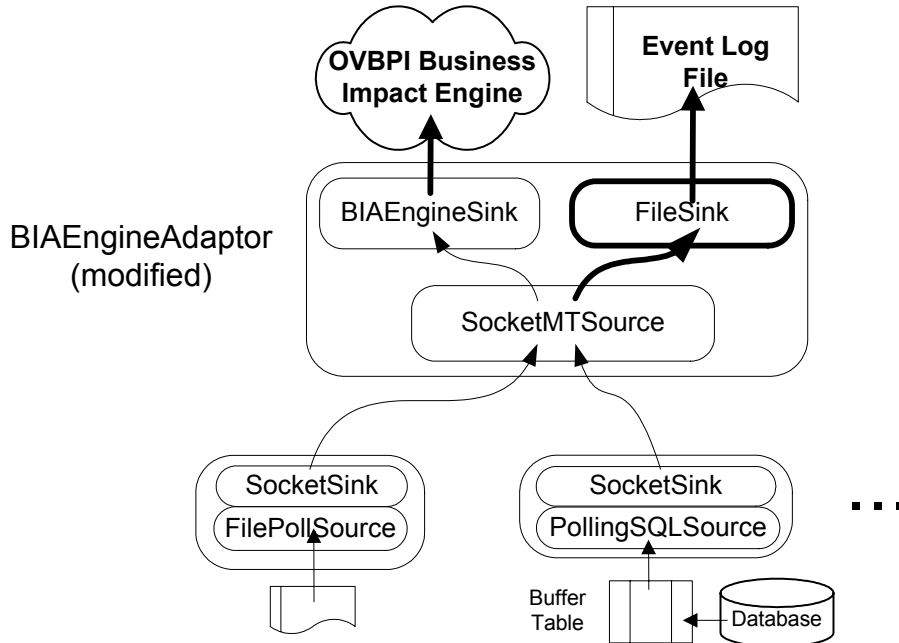
This idea works just the same if you are using the Event Store architecture.

You would need to decide whether you were going to do this to the bia_event_engineadaptor.props file directly and potentially lose the configuration the next time you (or someone else) makes a configuration change using the OVBPI Administration Console, or whether you were going to do the change within the template files, or using some other mechanism.

Here are the configuration changes that you need to make to the bia_event_engineadaptor.props file to configure the BIAEngineAdaptor to make a log of all events as the Business Impact Engine receives them. (This example assumes that the BIAEngineAdaptor is currently configured to use the SocketMTSource component.)

- **Add the new SaveSink component**

  You simply need to add a new component, which you call SaveSink, and assign it the next sequential component number.

For example, if the file contains the components:

```
BIAEngineAdaptor.Controller.Name = Controller
BIAEngineAdaptor.Component1.Name = SocketMTSource
BIAEngineAdaptor.Component2.Name = HospitalPipe
BIAEngineAdaptor.Component3.Name = BIAEngineSink
BIAEngineAdaptor.Component4.Name = HospitalSource
```

Then you would add your `SaveSink` component as component number 5, as follows (new lines shown in **bold** type):

```
BIAEngineAdaptor.Controller.Name = Controller
BIAEngineAdaptor.Component1.Name = SocketMTSource
BIAEngineAdaptor.Component2.Name = HospitalPipe
BIAEngineAdaptor.Component3.Name = BIAEngineSink
BIAEngineAdaptor.Component4.Name = HospitalSource
BIAEngineAdaptor.Component5.Name = SaveSink
```

- **Link in this new SaveSink component**

  You need to configure the `HospitalPipe` component to send its output to both the `SocketMTSource` and the new `SaveSink` components.

  You have two choices here. The order that you link them is important.

  If you link to the `SocketMTSource` first and the `SaveSink` second then if the `SocketMTSource` throws an exception (such as a hospital exception if the event is not a valid event) then the event is not passed to the `SaveSink`, and therefore not logged.

  If you link to the `SaveSink` first and the `SocketMTSource` second, then the event is logged first and then sent to the SocketMTSource for processing. Thus the log file contains all events including those that get hospitalized.

  So, the order of the linkage determines whether you want to log all events sent for processing (including those that get hospitalized), or just those that are actually sent into the Engine.

  Configure the `SaveSink` second so it only logs valid events. This is achieved as follows (new lines shown in **bold** type):

```
BIAEngineAdaptor.SocketMTSource.LinkTo1 = HospitalPipe
BIAEngineAdaptor.HospitalPipe.LinkTo1  = BIAEngineSink
BIAEngineAdaptor.HospitalPipe.LinkTo2  = SaveSink

BIAEngineAdaptor.HospitalSource.LinkTo1 = BIAEngineSink
BIAEngineAdaptor.HospitalSource.LinkTo2 = SaveSink
```

- **Configure the SaveSink**

  You can configure the SaveSink component however you like, but this example configures the sink to write out comma-delimited records. This produces concise output and allows you to import the file into an editor of your choice (maybe MS Excel) for viewing/filtering/sorting/etc.

  To configure the SaveSink as a comma-delimited file writer:

```
BIAEngineAdaptor.SaveSink.ClassName       =
                            org.openadaptor.adaptor.standard.FileSink
BIAEngineAdaptor.SaveSink.DOStringWriter  =
                            org.openadaptor.dostrings.DelimitedStringWriter
BIAEngineAdaptor.SaveSink.CreateOutputFile = true
BIAEngineAdaptor.SaveSink.OutputFileName   = CDF_event_store.txt
```

  Note that by default the output file is created in the *OVBPI-install-dir*\data\conf\bia directory as this is from where the BIAEngineAdaptor runs when started using the OVBPI Administration Console.

  You should really supply a fully qualified file path name for your installation, and place this file where ever is best for you.

Debugging Database Adaptors

You have configured your database adaptor and you start to suspect that events are not coming into the OVBPI Business Impact Engine as you expect. How do you go about debugging this?

Scenario 1: Missing Events

The scenario is as follows:

You have set up a database trigger to monitor the orders table. When new orders come in, your trigger fires and writes each record into a buffer table (BIA_BUF__Orders). Your adaptor is reading this buffer table and sending the events into the OVBPI Business Impact Engine.

You run the OVBPI Business Process Dashboard and start to see the flow instances created, and you watch them as they progress. However, you then start to notice that some flow instances never complete. Or, they complete but with some of the steps missing. That is, the OVBPI Business Impact Engine is not receiving all the events.

Events are going missing.

Possible Cause:

• The buffer table entries are not being identified uniquely

In the example shown in Events From a Database on page 234, when the database trigger writes each record to the buffer table, it assigns each record a unique id (the TheUID attribute).

This is used by the NextPrimaryKeySQL command to grab the set of keys to process in the next poll. More importantly, this unique ID is then used in the CommitSQL command to delete these records once they have been processed. If this unique ID attribute (TheUID) is not actually unique then the CommitSQL command deletes all records that match the ID and this could well be more records than had just been processed.

Always make sure that your CommitSQL command is being applied to the exact same records that were just processed.

Scenario 2: Events Out Of Sequence

The scenario is as follows:

> You have set up a database trigger to monitor the orders table. When new orders come in, your trigger fires and writes each record into a buffer table (`BIA_BUF__Orders`). Your adaptor is reading this buffer table and sending the events into the OVBPI Business Impact Engine.

> You run the OVBPI Business Process Dashboard and start to see the flow instances created, and you watch them as they progress. However, you start to notice that some flow instances go through just fine, but some flow instances are having their nodes processed in weird orders - totally out of sequence.

> Some flow instances are going through OK however they seem to show huge delays. That is, the first few nodes of a flow instance happen, then there is an overly long pause where no activity happens for this flow instance (even though you feel sure that the events have already occurred) and then eventually the rest of the events come through.

> The events are arriving in some strange sort of order.

Possible Causes: (These explanations refer to the example in Events From a Database on page 234.)

- The `PollingSQLSource` is not reading the events in the correct order

  When configuring the `PollingSQLSource` to read from the buffer table, it is important that the records be read in the order they were written. This can be achieved if the trigger has written a timestamp with each record.

  Thus your `SelectSQL` command must include some sort of `order by` specification.

- The trigger needs to write a `GeneratedDate`

  If the `PollingSQLSource` is configured to read records from the buffer table out of sequence, and there are no timestamps on each of these records, then when the events arrive at the OVBPI Business Impact Engine, the Business Impact Engine simply assigns a creation date to each event as it arrives. That is, the data records are written to the orders table (and thus the buffer table) in a certain order, but your adaptor sends them out of sequence and the OVBPI Business Impact Engine therefore records them as happening out of sequence.

So you really should configure your trigger(s) to write each buffer record with a `GeneratedDate` attribute, and then have your adaptor pass this through to the OVBPI Business Impact Engine. This preserves the correct sequencing of events.

Debugging A Database Source

The `PollingSQLSource` examples you have seen so far have tended to show the following kind of configuration segment:

```
B.DB.NextPrimaryKeySQL = select TheUID from BIA_BUF__orders
                                        where EventStatus = 'NEW'
B.DB.SelectSQL         = select * from BIA_BUF__orders
                                        where TheUID IN ('PK')
                                            order by GeneratedDate
B.DB.CommitSQL         = delete BIA_BUF__orders where TheUID IN ('PK')
```

If you then run your adaptor and get problems with events not seeming to arrive in the correct order, or events going missing, or whatever, how can you try to guess what has happened?

Obviously if you configure the BIAEngineAdaptor to log all events (as described in Debugging Your Adaptor Network on page 386) then you can see what order the events arrived at the OVBPI Business Impact Engine.

Or you might like to focus on the individual database adaptor itself. Here are some suggestions:

- When configuring an adaptor for the first time you should probably configure the sink to be the `FileSink`

  This way you can visually see the records on the screen and manually verify that they are coming out in the correct order. Once you are happy, you could then reconfigure your adaptor to send events to the OVBPI Business Impact Engine.

  The advantage of this test mechanism is that it can help you check that your SQL select command is selecting the data in the correct order, and that all the attributes are present in the resultant event.

  You could extend this idea and configure the `FileSink` to write comma-delimited output to a file. You could then import this file into something like MS Excel, as this would allow you to apply searches, sorts and filtering to the output.

Here is the necessary configuration for the FileSink to write out comma separated output:

```
A.Sink.ClassName        = org.openadaptor.adaptor.standard.FileSink
A.Sink.DOStringWriter   = org.openadaptor.dostrings.DelimitedStringWriter
A.Sink.CreateOutputFile = true
A.Sink.OutputFileName   = filename.txt
```

- Try altering the `CommitSQL` command to be an `update` rather then a `delete`

  With the commit action being a delete, this means that once you have run the adaptor all your buffer records have gone.

  It might be helpful to configure the adaptor to leave the records in the buffer table, but with the `EventStatus` column set to `DONE` (or whatever). For example:

```
B.DB.CommitSQL          = update BIA_BUF__orders
                             set EventStatus='DONE'
                             where TheUID IN ('PK')
```

  This would then allow you to perform select statements afterwards to test what has happened.

  You could also export the database entries from the buffer table into something like MS Excel, as this would allow you to apply searches, sorts and filtering to the data records to help you see what should have happened.

- You could always configure two sinks

  You could configure the adaptor to output comma-delimited output to a file and to also send the events to the OVBPI Business Impact Engine. This way you could debug what was actually sent to the Business Impact Engine...in parallel.

Auto-ReStarting Adaptors

When OVBPI is installed, the main components, such as the OVBPI Business Impact Engine, Business Event Handler, Servlet Engine, etc., are configured as Windows services.

You can see the OVBPI services by viewing the installed services on your PC. These services are set up with names beginning OVBPI.

Figure 71 The OVBPI Installed Services
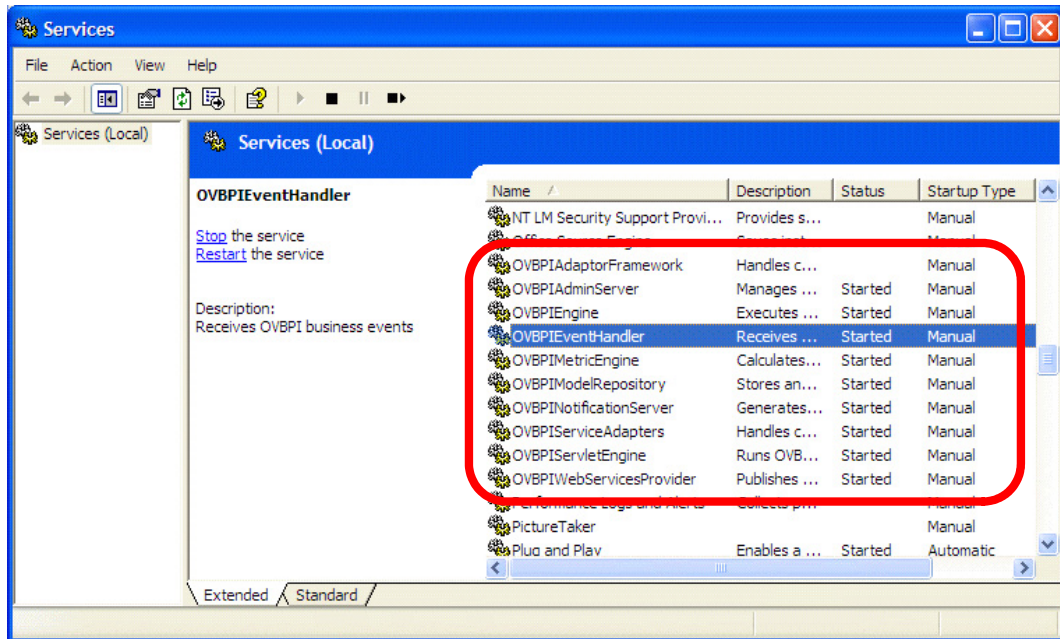
The service called `OVBPIEventHandler` is the service that runs the Business Event Handler. If this service should stop for any reason then, assuming you were using the socket-based architecture, all your connected client adaptors would lose the socket connection and close down.

You may want to configure this `OVBPIEventHandler` service to automatically restart itself if anything causes it to fail. To do this, simply right-click on the `OVBPIEventHandler` service, select `Properties`, and then the `Recovery` tab.

Configuring the `OVBPIEventHandler` service to auto-restart is one thing, but what about your client adaptors. They also need to be configured to auto-restart.

If you have a client adaptor running on an OpenView Managed Node then you have the software and ability to set up auto-restart. However, what about the adaptors running on non-OVO Managed Nodes?

Java Service Wrapper

There is some free software available called the Java Service Wrapper, available from Tanuki Software. This software provides the framework for easily configuring Java applications to run as Windows services. For non-Windows systems (such as Unix systems), the framework makes it easy to install you Java applications as daemon processes.

The standard OVBPI services are all configured using this Tanuki Java wrapper software.

The Tanuki Software Web site is found at:

`http://wrapper.tanukisoftware.org`

From there you can read the documentation and download versions appropriate for your operating system environments.

Using the Java Wrapper software, you are able to configure your client adaptors to run as Windows services (or daemons) and to configure them to auto-restart should they stop because (for example) the connection to the `OVBPIEventHandler` goes down.

Windows Example

For each adaptor you wish to run as a service, you need to do the following:

- Configure a Java wrapper configuration file

  The Java wrapper software provides a template file for this.

- Install the service

  The Java wrapper software provides a command for this.

- From the Windows services Control Panel:

  — Configure auto-start options

  — Configure recovery options

  — Start the service

Configure a Java Wrapper Configuration File

To configure a service you need to create a Java wrapper configuration file. In this configuration file you specify things such as:

- How to start your adaptor

- How to stop your adaptor

- The classpath

- Where to write all logging output

- The name of the Windows service

Let's look at an example Java wrapper configuration file, and then go through it step by step:

```
# Java Application
# ----------------
wrapper.java.command=%JAVA_HOME%/bin/java.exe

# The main class of the wrapper
# -----------------------------
wrapper.java.mainclass=org.tanukisoftware.wrapper.WrapperStartStopApp

# Application parameters
# ----------------------
wrapper.app.parameter.1=org.openadaptor.adaptor.RunAdaptor
wrapper.app.parameter.2=5
```

```
wrapper.app.parameter.3=c:/events-tg/sols/DB_Adaptor_mssql.props
wrapper.app.parameter.4=B
wrapper.app.parameter.5=-property
wrapper.app.parameter.6=AdaptorHistory.Directory
wrapper.app.parameter.7=%OVBPI_ROOT%/data/conf/bia
wrapper.app.parameter.8=org.openadaptor.adaptor.util.TerminateAdaptor
wrapper.app.parameter.9=true
wrapper.app.parameter.10=2
wrapper.app.parameter.11=c:/events-tg/sols/DB_Adaptor_mssql.props
wrapper.app.parameter.12=B

# Java Classpath
# --------------
wrapper.java.classpath.1=c:/wrapper_win32_3.1.2/lib/wrapper.jar

###OVBPI OA Enhancements
wrapper.java.classpath.2=%OVBPI_ROOT%/java/bia-event.jar

###openadaptor standard classpath
wrapper.java.classpath.3=%OA_ROOT%/classes/openadaptor.jar
wrapper.java.classpath.4=%OA_ROOT%/classes/jdk_1_4/regexp.jar
...etc...
wrapper.java.classpath.23=%OA_ROOT%/classes/xml4j.jar

###Database classes (if any)
wrapper.java.classpath.24=%OVBPI_ROOT%/nonOV/inet/Opta2000.jar

# Java Library Path (location of Wrapper.DLL or libwrapper.so)
# -----------------
wrapper.java.library.path.1=c:/wrapper_win32_3.1.2/lib

# Java Additional Parameters (if any)
# -------------------------
##wrapper.java.additional.1=-DOV_INSTALL_DIR="C:/bin/OVBPI"

# Initial Java Heap Size (in MB)
##wrapper.java.initmemory=3

# Maximum Java Heap Size (in MB)
##wrapper.java.maxmemory=64

# Log file to use for wrapper output logging
# ------------------------------------------
wrapper.logfile=c:/wrapper_win32_3.1.2/conf/Mywrapper.log

# Format of output for the log file.
wrapper.logfile.format=M

# Log Level for log file output.
wrapper.logfile.loglevel=INFO
```

```
# Maximum size that the log file is allowed to grow to before
#  the log is rolled. Size is specified in bytes.  The default value
#  of 0, disables log rolling.  May abbreviate with the 'k' (kb) or
#  'm' (mb) suffix.  For example: 10m = 10 megabytes.
wrapper.logfile.maxsize=0

# Maximum number of rolled log files which are allowed before old
#  files are deleted.  The default value of 0 implies no limit.
wrapper.logfile.maxfiles=0

# Log Level for sys/event log output.
wrapper.syslog.loglevel=NONE

# Format of output for the console.
wrapper.console.format=M

# Log Level for console output.
wrapper.console.loglevel=NONE

# Wrapper Windows Properties
# --------------------------
wrapper.console.title=OVBPI Example Adaptor

# Wrapper Windows NT/2000/XP Service Properties
# ---------------------------------------------
wrapper.ntservice.name=OVBPI-301-Adaptor
wrapper.ntservice.displayname=OVBPI-301-AdaptorDisplayName
wrapper.ntservice.description=Polls the DB and send OVBPI business events

# Service dependencies.  Add dependencies as needed starting from 1
##wrapper.ntservice.dependency.1=<other service name>

# Mode in which the service is installed.  AUTO_START or DEMAND_START
wrapper.ntservice.starttype=AUTO_START

# Allow the service to interact with the desktop.
wrapper.ntservice.interactive=false

# Disable the Wrapper auto-restart option
# ---------------------------------------
wrapper.max_failed_invocations=1
wrapper.successful_invocation_time=2147483647
```

Java Wrapper Configuration File - Explained

Let's go through this configuration file step by step:

```
# The main class of the wrapper
# ----------------------------
wrapper.java.mainclass=org.tanukisoftware.wrapper.WrapperStartStopApp
```

Tanuki Software offer a number of "main" classes that you could choose to run. The `WrapperStartStopApp` is a good one to run as it allows you to specify the Java application to start your adaptor and the Java application to stop your adaptor.

These star and stop classes are specified by the `wrapper.app.parameter.<n>` section:

```
# Application parameters
# ---------------------
wrapper.app.parameter.1=org.openadaptor.adaptor.RunAdaptor
wrapper.app.parameter.2=5
wrapper.app.parameter.3=c:/events-tg/sols/DB_Adaptor_mssql.props
wrapper.app.parameter.4=B
wrapper.app.parameter.5=-property
wrapper.app.parameter.6=AdaptorHistory.Directory
wrapper.app.parameter.7=%OVBPI_ROOT%/data/conf/bia
wrapper.app.parameter.8=org.openadaptor.adaptor.util.TerminateAdaptor
wrapper.app.parameter.9=true
wrapper.app.parameter.10=2
wrapper.app.parameter.11=c:/events-tg/sols/DB_Adaptor_mssql.props
wrapper.app.parameter.12=B
```

Because you have specified `WrapperStartStopApp`, you need to specify both the start and stop command lines.

You specify the following parameters:

.1   = The java class to start your adaptor (`RunAdaptor`).

.2   = The number of parameters this starting class needs (`5`).

.3   = The adaptor property file.

.4   = The name of the adaptor within this property file.

.5   = Passing in an additional property.

.6   = The adaptor history property.

.7   = The directory to contain any adaptor history.

.8 = The java class to stop the adaptor (`TerminateAdaptor`).

.9 = Whether to wait for all threads to complete (`true`).

.10 = The number of parameters to this stopping java class (`2`).

.11 = The adaptor property file. This provides the remote control port.

.12 = The name of the adaptor within this property file.

The `org.openadaptor.adaptor.util.TerminateAdaptor` class is supplied in the `bia-event.jar` library. It just needs to be passed the adaptor property file and the name of the adaptor to be stopped. `TerminateAdaptor` uses the adaptor property file to determine the adaptor remote control port. Your adaptor must be configured to offer remote control for this mechanism to work.

Notice that the Java wrapper configuration can use environment variables. For example:

```
wrapper.app.parameter.7=%OVBPI_ROOT%/data/conf/bia
```

uses the OVBPI_ROOT environment variable.

```
# Java Classpath
# -------------
wrapper.java.classpath.1=c:/wrapper_win32_3.1.2/lib/wrapper.jar

###OVBPI OA Enhancements
wrapper.java.classpath.2=%OVBPI_ROOT%/java/bia-event.jar

###openadaptor standard classpath
wrapper.java.classpath.3=%OA_ROOT%/classes/openadaptor.jar
wrapper.java.classpath.4=%OA_ROOT%/classes/jdk_1_4/regexp.jar
...etc...
wrapper.java.classpath.23=%OA_ROOT%/classes/xml4j.jar

###Database classes (if any)
wrapper.java.classpath.24=%OVBPI_ROOT%/nonOV/inet/Opta2000.jar
```

You specify the classpath for your adaptor. Notice that the first class on the classpath is `wrapper.jar`. This comes with the Java wrapper when you download it. This can be installed anywhere on your system.

The classpath then just needs to include `bia-event.jar`, the standard openadaptor classpath (refer to your *OA-install-dir*\examples directory for examples of this classpath) and any additional database libraries that you might require. You need to set up whatever classpath your adaptor requires.

```
# Java Library Path (location of Wrapper.DLL or libwrapper.so)
# -----------------
wrapper.java.library.path.1=c:/wrapper_win32_3.1.2/lib
```

You need to specify the location of the Java wrapper library. This comes with the Tanuki Software when you install it. You just need to specify the directory that contains the `wrapper.dll` file.

```
# Log file to use for wrapper output logging
# ------------------------------------------
wrapper.logfile=c:/wrapper_win32_3.1.2/conf/Mywrapper.log
```

Specify a file to hold the standard output of your Windows service. This contains all adaptor standard output as well.

```
# Wrapper Windows NT/2000/XP Service Properties
# ---------------------------------------------
wrapper.ntservice.name=OVBPI-301-Adaptor
wrapper.ntservice.displayname=OVBPI-301-Adaptor
wrapper.ntservice.description=Polls the DB and send OVBPI business events

# Service dependencies.  Add dependencies as needed starting from 1
##wrapper.ntservice.dependency.1=<other service name>

# Mode in which the service is installed.  AUTO_START or DEMAND_START
wrapper.ntservice.starttype=AUTO_START
```

This is where you specify the service definition. You can give your service an internal name and display name. The display name is the name that appears when you list the service in the Windows control panel.

You can specify any service dependencies. If you do specify some dependencies then this service is stopped whenever any of the dependent services are stopped.

You can also configure whether the service is to be configured to start when the machine is booted or not.

```
# Disable the Wrapper auto-restart option
# ------------------------------------
wrapper.max_failed_invocations=1
wrapper.successful_invocation_time=2147483647
```

When you configure an adaptor within the Java wrapper software, you have the ability to configure auto-restart within the wrapper itself. However, as this adaptor is being installed as a Windows service, you can control the auto-restart options from within the Windows control panel. So there is no need for the Java wrapper's built-in auto-restart option.

Install the Service

Once you have created a Java wrapper configuration file, you need to install this as a service. Tanuki Software provide an executable for installing and de-installing services. The command lines are as follows:

Install a Service:

```
$ Wrapper-Install-dir\bin\wrapper.exe -i wrapper-config-file
```

De-Install a Service:

```
$ Wrapper-Install-dir\bin\wrapper.exe -r wrapper-config-file
```
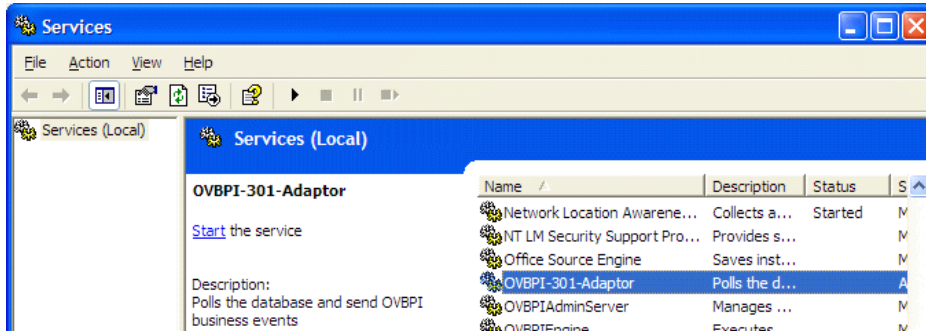
For example:

```
$ cd Wrapper-install-dir\bin
$ wrapper.exe -i ..\conf\MyWrapper.cfg
```

Configure the Windows Service Behavior

Once you have installed your service (`wrapper.exe -i`) you can see it in the Windows services control panel:

Figure 72 Windows Service Control Panel

From here you can start the service, which starts up your adaptor. You can monitor the log file (that you specified in the wrapper configuration file) to see the adaptor starting up.

If there are any issues with the configuration of your adaptor, the log file shows you this. You can then de-install the service, fix the problem and re-install the service.

You can also set up recovery options for your service. Right-click on the service, select `Properties`, and then click on to the `Recovery` tab:

Figure 73 Recovery Options
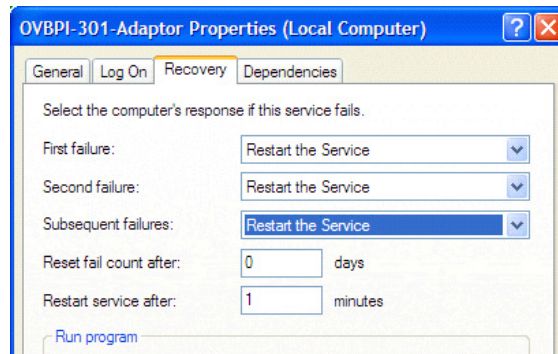
Here you can set the failure actions to be (for example) `Restart the Service`.

Unix Example

To configure a restartable service on a Unix machine requires virtually the same steps as when configuring on a Windows machine. The main difference being that, because you do not have any Windows Control Panel, you need to configure the adaptor to restart itself if it fails unexpectedly.

For each adaptor you wish to run as a service, you need to:

- Configure a Java wrapper configuration file

  Refer to Configure a Java Wrapper Configuration File on page 396. You make the obvious substitutions for things such as variable referencing (`$` rather than `%`) and directory names (no `c:` required :-)

  However, you also need to configure the adaptor to be restartable. Refer to Configuring Auto-Restart Within the Adaptor on page 404 for details of how to do this.

- Install the service

  Refer to Install the Service on page 402.

Configuring Auto-Restart Within the Adaptor

You need to configure each adaptor to be able to auto-restart itself if it encounters an error and subsequently closes down. To do this, you need to edit the wrapper configuration file and set the following configuration options:

```
wrapper.max_failed_invocations
wrapper.successful_invocation_time
wrapper.on_exit.default
wrapper.on_exit.0
wrapper.restart.delay
```

For example:

```
wrapper.max_failed_invocations=5
wrapper.successful_invocation_time=300
wrapper.on_exit.default=RESTART
wrapper.on_exit.0=SHUTDOWN
wrapper.restart.delay=60
```

where:

- `wrapper.restart.delay=60`

  Tells the wrapper to always wait `60` seconds before trying to restart the adaptor.

- `wrapper.on_exit.default=RESTART`
  `wrapper.on_exit.0=SHUTDOWN`

  These settings tell the adaptor to auto-restart in the event of a failure.

  If the adaptor goes down with an error code other than zero, it tries to restart the adaptor.

- `wrapper.max_failed_invocations=5`

  This setting tells the wrapper that it should try to restart the adaptor up to a maximum of five times. If it cannot successfully restart the adaptor after five attempts, then the wrapper fails and logs a failure message.

- `wrapper.successful_invocation_time=300`

  This setting works in conjunction with the `max_failed_invocations` setting.

  Assuming the wrapper is able to restart the adaptor, if the adaptor then stays running for at least 300 seconds, the wrapper resets its retry counter back to zero. That is, if the adaptor fails sometime later (after 300 seconds) the wrapper again retries up to five times.

For example, if you wanted to set an adaptor to keep restarting, every five minutes, for up to 12 hours, you would set the options as follows:

```
wrapper.max_failed_invocations=144
wrapper.successful_invocation_time=3600
wrapper.on_exit.default=RESTART
wrapper.on_exit.0=SHUTDOWN
wrapper.restart.delay=300
```

where:

- This retries up to 144 times, each at five minutes (300 seconds) apart, and 144 * 5 minutes = 12 hours.

- Successful invocation time is set to 3600 seconds (one hour), which means that no matter how many retries it took to restart the adaptor last time, once the adaptor has been up and running for over one hour it resets the retry count back to zero. Thus if the adaptor then failed again, the wrapper would retry up to another 144 times.

Operational/Service Status Events

Right from the start of this guide the distinction was made between Business Data Events and Operation Events, and it was stated that the Event Handler handles Business Data Events.

Well...it does! But the Event Layer can also handle Operation Events.

The built-in integrations with products such as OVO, OVIS, and SOA Manager, all use dedicated mechanisms for sending Operational Events into the OVBPI Business Impact Engine. But to allow the OVBPI Business Impact Engine to receive Operation Events from third party service monitoring systems, the Event Handler is able to accept Operation Events directly.

The SERVICE_STATUS_CHANGE Event

The event that sets the status of a service within OVBPI is the `SERVICE_STATUS_CHANGE` event. This is not an event that you define within the Modeler, it is an event **pre-defined** within OVBPI.

The `SERVICE_STATUS_CHANGE` event contains the following attributes:

- `EventName`

  Contains the globally unique ID (the service ID) assigned to the service definition within the OVBPI Business Impact Engine database.

- `EventGroup`

  Contains the string value: `SERVICE_STATUS_CHANGE`.

- `EventSeverity`

  Set to one of the string values: `Critical`, `Major`, `Minor`, `Warning`, `Normal`.

- `OldStatus`

  Set to the previous state for this service.

- `RootCause` (optional)

  A string to describe the root cause for the service changing state.

- `GeneratedDate` (optional)

  Set to the date/time this event was created.

To generate your own `SERVICE_STATUS_CHANGE` event you simply need to configure an adaptor to send this event through the Event Handler.

The trick is knowing how to set the correct value for the `EventName` attribute.

EventName/Service ID

The `EventName` attribute specifies the actual service for which you are generating a status change. However, the value within this `EventName` attribute must be the globally unique ID assigned to this service within the OVBPI Business Impact Engine database. The problem for most people is that they only know the external "display" name of the service.

For a standalone service, this unique service ID is actually just the name of the service (as defined within the OVBPI Modeler) prefixed with the special string tag: `SALS:`

For example:

> If your services are defined as follows within the Modeler:

```
Standalone Services
          ServiceA
          MyOtherService
```

> Then the full service IDs are:

```
SALS:ServiceA
SALS:MyOtherService
```

So, the unique service ID for a standalone service is very easy to work out.

If you ever want to check this service ID for yourself then you can, by looking directly at the OVBPI Business Impact Engine database. The data table that holds all the OVBPI service definitions is called: `Resources`. The column that holds the unique service "ID" is called: `Resource_ID`.

For any service, you can determine the service ID by looking at the `Resource_ID` column of this table. This table holds only those services that are used within deployed flow.

Generating a SERVICE_STATUS_CHANGE Event

Suppose you want to generate a SERVICE_STATUS_CHANGE event to set the standalone service MyService to be in a Critical state:

Your input event could be:

```
EventName:      SALS:MyService
EventGroup:     SERVICE_STATUS_CHANGE
EventSeverity:  Critical
OldStatus:      Normal
RootCause:      It all went wrong!
```

Suppose this is written to a file called service_events.txt in the form:

```
SALS:MyService|SERVICE_STATUS_CHANGE|Critical|Normal|It all went wrong!
```

Your adaptor configuration could be as follows:

```
# List and name all components for this adaptor
# --------------------------------------------
SS.Controller.Name       = Controller
SS.Logging.Name          = Logging
SS.Component1.Name        = Source
SS.Component2.Name        = Sink

# Show how the components link together
# ------------------------------------
SS.Source.LinkTo1         = Sink

# Source is the FileSource
# --------------===========
SS.Source.ClassName       = org.openadaptor.adaptor.standard.FileSource

SS.Source.InputFileName   = service_events.txt

# Using a | separator (Decimal 124 = "|")

SS.Source.DOStringReader  = org.openadaptor.dostrings.DelimitedStringReader

SS.Source.Type            = ServiceStatusChange
SS.Source.NumAttributes   = 5
SS.Source.FieldDelimiter  = 124
SS.Source.AttName1        = EventName
SS.Source.AttName2        = EventGroup
SS.Source.AttName3        = EventSeverity
SS.Source.AttName4        = OldStatus
SS.Source.AttName5        = RootCause
```

```
# Sink is a SocketSink
# ---------------------
SS.Sink.ClassName           = org.openadaptor.adaptor.standard.SocketSink
SS.Sink.Port                = 44005
SS.Sink.HostName            = localhost
```

When sending SERVICE_STATUS_CHANGE events it seems that you do not have to get the OldStatus value correct. That is, the OldStatus attribute is required within the event, however it seems to be OK if you always set it to Normal. Obviously you can set it to the previous state value if you happen to know it.

The Command Line

If you wish to generate SERVICE_STATUS_CHANGE events directly from the command line then there is a script file that comes with the Generic Event Injector contributed utility that you might like to use.

To install the Generic Event Injector contributed utility please refer to the *OVBPI Integration Training Guide - Modeling Flows*.

With the Generic Event Injector installed, you find the script under the directory:

    webapps\event-injector\WEB-INF\classes

The script is called:

    statuschangeadaptor.bat

You may need to edit this script file and set the following things:

- BIA_ROOT

    The script sets the BIA_ROOT environment variable to the default install location. If you have installed OVBPI to a non-default location then you need to edit the script and set BIA_ROOT accordingly.

- JAVA_HOME

    If the JAVA_HOME environment is not set within your command window environment, then you need to set it within the script.

To run the script:

- Open a command window

- cd to the `webapps\event-injector\WEB-INF\classes` directory

- Run the script file, specifying the full service ID followed by the severity to which you want it set. The default severity is `Critical`

  For example:

  `$ statuschangeadaptor "SALS:My Service"`

  would set the service to a state of `Critical`.

  `$ statuschangeadaptor "SALS:My Service" Normal`

  would set the service to a state of `Normal`.

If you get any unexpected errors then first check the `statuschangeadaptor.bat` file for any directory names/paths that might be incorrect for your installation.

Intervention Client Scripts

The OVBPI Intervention Client allows you to intervene on your run time flows and data. It is a utility that needs to be used with extreme care, but can be very helpful during both development and production.

The OVBPI Intervention Client comes installed under:

OVBPI-install-dir\nonOV\jakarta-tomcat-5.0.19\webapps\ovbpiintclient

To run the OVBPI Intervention Client, refer to the *OVBPI Integration Training Guide - Modeling Flows* in the chapter on Advanced Flow Definition.

The OVBPI Intervention Client provides a separate JSP for each function that it provides. All of the actual work is performed by a central bean called `com.hp.ov.bia.views.util.InterventionBean`. This bean can be called directly from your own scripts to automate any intervention.

Again, you must be extremely careful with any such scripts as the potential to delete/update the wrong thing is very great. **So be careful!**

Resetting The Flow Totals/Averages

One function of the OVBPI Intervention Client is where you can reset the totals and averages maintained for a given flow definition. This includes the following set of statistics:

- The total number of flow instances for a flow

  This is the total number of flow instances that have been created since the flow was deployed, or since you last reset it.

- The average time for a flow to complete

  This is the average of all the completed flow instances since the flow was deployed, or since the last reset.

- The totals and averages maintained for each node of this flow

   These include (for each node):

   — Total number of node instances

   — Total cumulative node instance time

   — Total cumulative node weight value

   — Average node completion time

By using the OVBPI Intervention Client you can reset all these values back to zero for a specified flow.

This can be very useful if you are using these fields to monitor how many flow instances you are handling each day (or week, or whatever), as you can reset them at the beginning of each measurement period. It is just like a "trip meter" on your car, allowing to measure the distance for a particular trip you travel.

This "resetting the counters" might be something that you want to automate and have them reset at (for example) midnight every saturday night.

A Reset Application Example

In the `sols\intervention` directory, there is an example Java application that resets the totals and averages for a given flow name.

The file is called: `FlowTotalsReset.java`

This example application works as follows:

- Reads the flow name from the command line

- Invokes the intervention bean to reset the totals/averages

- Resets the totals and averages for the given flow and any superseded versions of the given flow

If you only want to reset the totals/averages for the active flow name and not any superseded definition, then refer to the example file:

   `FlowTotalsResetSpecific.java`

Let's have a look at the basic structure of the `FlowTotalsReset.java` application (you can refer to the actual file in the `sols\intervention` directory if you want to see all the code):

```
import com.hp.ov.bia.views.util.InterventionBean;
import com.hp.ov.bia.views.util.Constants;

public class FlowTotalsReset
{

 public static void main(String[] args)
 {

    ...first read the flow name from the command line

    /*
     * Now use the Interactive Bean to reset the flow totals
     */

    try
    {
      InterventionBean iBean = new InterventionBean();

      iBean.loadEngineConfiguration();
      iBean.connectToRmiService(Constants.MODEL_MANAGER_SERVICE);
      iBean.connectToRmiService(Constants.EVENT_MANAGER_SERVICE);

      iBean.resetTotals(flowName);
    }
    catch (Throwable e)
    {
      e.printStackTrace();
    }
 }

}
```

where:

- **Import the classes**

  ```
  import com.hp.ov.bia.views.util.InterventionBean;
  import com.hp.ov.bia.views.util.Constants;
  ```

  These import the two classes that you need.

- **The code reads the flow name from the command line (code not shown in the above example)**

It's important to realize that the flow name **is** case sensitive. If you try to specify the flow name in the wrong case, the application sends in the reset event and nothing is returned to say that it failed...but it has failed.

• Send in the actual reset event

```
InterventionBean iBean = new InterventionBean();

iBean.loadEngineConfiguration();
iBean.connectToRmiService(Constants.MODEL_MANAGER_SERVICE);
iBean.connectToRmiService(Constants.EVENT_MANAGER_SERVICE);

iBean.resetTotals(flowName);
```

You get an `InterventionBean` instance. To initialize this bean you need to tell it to load its configuration and to connect to the necessary RMI services. This just means that the intervention bean gets connected to the OVBPI Business Impact Engine in readiness to send the event.

You then send in the reset totals event, specifying the flow name (case sensitive).

There is no feedback to know whether the event actually does anything once it gets into the OVBPI Business Impact Engine :-)

Running The Reset Example

The scripts to compile and run this flow reset application are located in the `sols\intervention` directory:

• You need to copy all of the file in the `sols\intervention` directory to the `WEB-INF\classes` directory of your OVBPI Intervention Client

So you would copy them to the directory:

OVBPI-install-dir\nonOV\jakarta-tomcat-5.0.19
                    \webapps\ovbpiintclient\WEB-INF\classes

• Now work on the files in this `WEB-INF\classes` directory

• Edit the script: `runReset.bat`

Set the `BIA_ROOT` variable correctly for your install.

- You can now run the reset application as follows:

  ```
  $ runReset filename
  ```

  For example:

  ```
  $ runReset "Order Flow"
  ```

  resets the totals for the `Order Flow` flow

The flow name is case sensitive, and there is unfortunately no feedback from the OVBPI Business Impact Engine to say whether your reset event actually did anything. But you can obviously check that it worked by looking at the OVBPI Business Impact database directly or by running the OVBPI Business Process Dashboard.

Installing openadaptor Standalone

Let's look at how you can configure a standalone openadaptor installation to make use of the OVBPI extensions and send events to your OVBPI server.

To install openadaptor on your server you do the following:

- Install the openadaptor product
- Set the openadaptor classpath
- Install the OVBPI Enhancements

Install the openadaptor Product

The required version of openadaptor is `1.6.5`.

If you are installing openadaptor on another Windows PC then you simply take a copy of the *OVBPI-install-dir*`\nonOV\openadaptor` directory on your OVBPI server, and place that on your other Windows PC. This gives you a full installation of openadaptor.

If you are installing openadaptor on another platform then you need to visit the openadaptor Web site, **www.openadaptor.org**, download version `1.6.5` for your operating environment, and install it.

Set the openadaptor Classpath

There are script files supplied with the openadaptor installation that may or may not be auto-configured at installation time. It is always best to check these and make sure that the classpath has been set correctly:

- change directory to the *OA-install-dir*`\examples` directory.
- Edit the appropriate `set_classpath_jdk_*` file for your installation.
- Make sure that the `CLASSES_DIR=` variable is set to the fully qualified path of the `classes` directory for your openadaptor installation.

Install The OVBPI Enhancements

The OVBPI enhancements to openadaptor are located in a single Jar file. This Jar file is on your OVBPI server in the file:
`OVBPI-install-dir`\java\bia-event.jar.

- Copy this `bia-event.jar` file over to your non-OVBPI server

    You can place this anywhere you like...just so long as you remember where you put it.

- Create yourself a new script file to set the correct classpath

    Create a script file that first runs the standard openadaptor `set_classpath_jdk_*` file and then prepends the `bia-event.jar` file to the classpath.

    If your adaptor is going to access databases then you also need to add the necessary database Jar/Zip files to your classpath. This is obviously installation, and database version, specific.

    Here is an example script (for a UX install):

```
# Invoke the standard OA environment
# --------------------------------
OA_install=/var/opt/openadaptor/1_6_5
. ${OA_install}/examples/set_classpath_jdk_1_4.profile

# Add in the OVBPI-extensions
# --------------------------
OVBPI_CLASSES=/var/ovbpi/classes
CLASSPATH=${OVBPI_CLASSES}/bia-event.jar:${CLASSPATH}

# Add any database class(es)
# -------------------------
SQL_CLASSES=/var/ovbpi/db

# Example Oracle classes
CLASSPATH=${SQL_CLASSES}/oracle/ojdbc14.jar:${CLASSPATH}
CLASSPATH=${SQL_CLASSES}/oracle/orai18n.jar:${CLASSPATH}
CLASSPATH=${SQL_CLASSES}/oracle/classes12.jar:${CLASSPATH}

# Example MSSQL classes
CLASSPATH=${SQL_CLASSES}/mssql/Opta2000.jar:${CLASSPATH}
```

    You might save this file as: `standaloneOA_set_ovbpi.sh`

Running Your Adaptor

You run adaptors by first running your environment set up script and then using the standard openadaptor `org.openadaptor.adaptor.RunAdaptor` class.

For example:

```
$ ./oa_set_ovbpi.sh
```

This sets the environment to include the OVBPI extensions

```
$ java org.openadaptor.adaptor.RunAdaptor file.props adaptor
```

Runs the adaptor.

Adaptor History

If you are wanting to use adaptor components that save adaptor history then you might want to set the additional property:

```
-property AdaptorHistory.Directory directory_name
```

when you run the adaptor. This would set the directory in which the component looks for, and saves, any adaptor history information.

For example:

```
$ java org.openadaptor.adaptor.RunAdaptor file.props adaptor
            -property AdaptorHistory.Directory /ovbpi/history
```

would save any adaptor history into the directory `/ovbpi/history`.