



Adeptia Developer's Reference Guide

Version 1.0

Release Date Jan 18, 2012

343 West Erie, Suite 440
Chicago, IL 60654, USA
Phone: (312) 229-1727 x111
Fax: (312) 229-1736

Copyright

Copyright © 2000-2007 Adeptia, Inc. All rights reserved.

Trademarks

The Adeptia logo and Manage, Automate and Optimize Business Processes are trademarks of Adeptia, Inc.

Statement of Conditions

Adeptia, Inc. provides this publication "as is" without warranty of any kind, either express or implied. In no event shall Adeptia be liable for any loss of profits, loss of business, loss of use or data, interruption of business, or for indirect, special, punitive, incidental, or consequential damages of any kind.

No part of this work covered by copyright herein may be reproduced in any form or by any means—graphic, electronic or mechanical—including photocopying, recording, taping, or storage in an information retrieval system, without prior written permission of the copyright owner.

This publication is subject to replacement by a later edition. To determine if a later edition exists, contact www.adeptia.com.

Contact Information

In case of any queries, please contact us at:

Contact For	Email ID
Sales	sales@adeptia.com
Support	support@adeptia.com

For more information, you can visit us at www.adeptia.com

Table of Contents

Objective	4
Pre-requisites.....	4
Custom Java Development with Adeptia	5
Adeptia Services That Can Be Extended For Custom Java Development.....	5
Using Adeptia Custom Plugin Service	5
Interacting With the Service Variables and Predefined Methods ..	6
Interacting With Streams.....	6
Interacting With Context Variables.....	9
Data Handling and Transformation.....	11
Calling External Java or Native Programs	12
Logging.....	14
Exception Handling.....	14
Dependency.....	15
Examples	15
Custom API For Custom Java Development	24
Java API for Adeptia Product.....	24
Java APIs For Third Party Libraries	24
JRE	25
HSQLDB.....	25
Castor.....	26
Log4J.....	26
Jasper.....	26
VFS.....	27
Java Mail API	27
Design patterns	29
Objective.....	29
Business Scenarios	29
Developing Complex GUI for Human Workflow	31
Objective.....	31
Requirements.....	31
Approaches	31
Use of custom JSP.....	31
Working with Web Service Publisher	34
Objective.....	34
Tasks	34
Example	34
Appendix A (Input XML schema)	38

Appendix B (Output XML schema)	39
Using Database Event	40
Objective.....	40
Tasks For Using Database Event.....	40
Creating Database Drivers and Database Info.....	40
Creating Database Event.....	42
Using SQL Query	42
Advantages.....	48
Limitations.....	48
Using SQL Trigger	48
Database Name.....	49
Task.....	49
Trigger Definition.....	49
Database Name.....	50
Trigger Definition.....	50
Reference	50
Advantages.....	55
Limitations.....	55
Problem Scenarios where both SQL Query or SQL Trigger serve as solutions.....	55
Appendix 1	55
Using XSL Template for Padding in Mapping.....	56
Objective.....	56
Tasks.....	56

OBJECTIVE

This document describes the scope and the topics covered in Technical Documentation for Developer. It acts as a reference guide for developers, while developing solutions using the Adeptia product. It is complementary to the User Guide.

The current product documentation does not provide sufficient information to develop complex solutions using the product. Thus, there is need to provide:

- Documentation for Custom Java development with Adeptia product
 - [Adeptia services that can be extended for custom Java development](#)
 - [Custom API for Adeptia](#)
 - [Java API for Adeptia Product](#)
 - [Java API for Third Party Libraries](#)
 - Example Code segments
- [Design patterns](#) that cover the readymade solutions to standard business scenarios that a developer encounters while developing complex solutions. It guides the developer as in what to use for what situation and how to use it.

PRE-REQUISITES

Before reading this document, it is assumed that you have conformed to the following pre-requisites:

- Complete knowledge of Java
- Familiarity with the Adeptia product
- Undertaken a 2-day training of the Adeptia product (preferred)
- Read the User Guide

CUSTOM JAVA DEVELOPMENT WITH ADEPTIA

This section describes all the Adeptia features that can be extended using custom Java coding. It provides Java API reference that can be used for coding.

ADEPTIA SERVICES THAT CAN BE EXTENDED FOR CUSTOM JAVA DEVELOPMENT

The Adeptia services that can be enhanced for custom Java coding are outlined below:

- [Custom Plugin](#): You can develop Java code to create and execute customized services in a process flow.
- *Rec2Rec Transformer*: You can develop customize services to process data individually record by record and add it in a process flow.
- *Java expression in decision node*: You can develop Java conditional logic and use it as a Boolean condition in a decision node in a process flow.
- *Custom JSP*: You can develop customized GUI with Java Server Pages (JSP) and incorporate it in the Adeptia GUI to develop its workflow GUI.
- *Java function support in Mapping*: You can develop customized data processing rules in Java and use them in Adeptia's Mapping feature.
- *Inbuilt Code segment and library*: You can develop code segments that can be used for custom Java development. For example, you can develop a code segment for:
 - Get Connection from DBInfo



In this version of document, only Custom Plugin is documented. We will document others later.

USING ADEPTIA CUSTOM PLUGIN SERVICE

A custom plugin service enables you to create customized services apart from the standard Adeptia server services. A developer can write custom java code to process the data as required.

Writing a custom plugin service involves the following steps:

1. [Interacting with the service variables and predefined methods](#)
2. [Interacting with streams](#)
 - [Input Stream](#)
 - [Output Stream](#)
3. [Interacting with Context variables](#)
 - [Getting context variable](#)

- [Setting context variable](#)
- 4. [Data handling and Transformation](#)
- 5. [Calling external native programs](#)
- 6. [Logging](#)
- 7. [Exception handling](#)
- 8. [Dependency](#)
- 9. [Examples](#)

Interacting With the Service Variables and Predefined Methods

A custom plugin service provides an interface to write a java code. The following variables/ methods will be available for the java code. Java code can directly access these variables by name.

Variables

- Context

Context variables are variables whose values can be accessed in a Process Flow context for the purpose of using these values in conditional control flows, passing values as parameters to another activity, or in other Process Flow related functions.

- Inputstream

InputStream variables contain data streams that can be used in a custom plugin service.

- Service

Service refers to current custom plugin service instance.

Methods

- `public void write(byte[] b, String streamName) throws IOException`

This method is used to write data to output streams.

- `public Logger getLogger()`

This method is used to obtain logger instance.



The usage of these methods is covered in the following section.

Interacting With Streams

Like other services, custom plugin service can receive data from other service(s) and generate data to send to other service(s).

While interacting with streams, you need to note the following points:

- Custom plugin service can have 0 or more input streams
- Custom plugin service can have 0 or more output streams
- These streams can be set in the Process Designer applet. To know how to set these streams refer to the section *Using Multiple Stream under Process Designer* chapter of the User Guide.
- For 0 input stream, the attribute “*Consume Stream*” for custom plugin service activity in Process Designer must be set to *false*.
- For 0 output stream, the attribute “*Generate Stream*” for custom plugin service activity in Process Designer must be set to *false*.
- By default, it generates one output stream and consumes no stream.
- Streams are closed when current custom plugin service execution is completed. So you need not close the streams in java code.

Interacting With Input Stream

Input streams are the data streams that are passed from source activity to the current custom plugin service activity. Data from these streams can be parsed, transformed or transmitted to other activities.

Default Stream

Custom plugin service can have one or more input streams. The first stream which is connected to the custom plugin service activity is called default stream. To access the default stream, the variable “**inputStream**” can be used in java code. This variable will be instance of “`java.io.InputStream`”. Input streams other than default stream are accessed by their names.

The following code snippet provides an example of how to access default input stream:

```
int i=0;
while ((i = inputStream.read()) != -1) {
    //data manipulation
}
Default stream can also be accessed using following code
// imports
import com.adeptia.indigo.services.transform.ScriptedService;
// type casting "service" to ScriptedService instance
ScriptedService scriptedService = (ScriptedService) service;
//to get default stream , which is first stream as input
InputStream default = scriptedService.getSourceStream();
int i=0;
while ((i = default.read()) != -1)
{
    //data manipulation
}
```

For more details, refer to the section [Examples](#).

Accessing Any Input Stream

Since the input streams are assigned with a name to custom plugin service, so these can be accessed by name. To check how many streams are assigned to custom plugin service, check "Source Stream" attribute for custom plugin service in PD.

This attribute can take the value such as:

ActivityName1: ActivityName2[stream2]: ActivityName3[mystream]

Its parameters are outlined in the table below.

Parameter	Description
Character ':'	Separates the different input streams
ActivityName1	Name of source activity sending the default stream to custom plugin service
ActivityName2	Name of source activity sending the stream2 stream to custom plugin service
ActivityName3	Name of source activity sending the mystream stream to custom plugin service

The following code snippet provides an example of how to access these streams:

```
// import
import com.adeptia.indigo.services.transform.ScriptedService;
// type casting "service" to ScriptedService instance
ScriptedService scriptedService = (ScriptedService) service;
// getting input streams
// first stream is also default stream
InputStream in1 =
scriptedService.getSourceStream("ActivityName1");
InputStream in2 =
scriptedService.getSourceStream("ActivityName2[stream2]");
InputStream in3 =
scriptedService.getSourceStream("ActivityName3[mystream]");
```

Interacting With Output Stream


Output streams are the data streams, which are passed from current custom plugin service activity to target activities. Data can be written to any stream by following method exposed by custom plugin service:

write(byte[] b, String streamName)

Its parameters are outlined in the table below.

Parameter	Description
'b'	Byte Array

streamName	Name of output stream
------------	-----------------------

	If no stream exists with name given above, then exception will be thrown.
---	---

In PD, check the attribute "*StreamNames*" for custom plugin service activity. This attribute can take the value such as:

Stream1:Stream2:Stream5

Its parameters are outlined in the table below.

Parameter	Description
Character ':'	Separates the different output streams
Stream1	Name of the target stream sending data from custom plugin service to target activity.
Stream2	Name of the target stream sending data from custom plugin service to target activity.
Stream5	Name of the target stream sending data from custom plugin service to target activity.

The following code snippet gives the information to access output stream.

```
// imports
import com.adeptia.indigo.services.transform.ScriptedService;
import java.io.OutputStream;

// type casting "service" to ScriptedService instance
ScriptedService myscriptedService = (ScriptedService) service;
//to write data on stream "default"
myscriptedService.write("data to send".getBytes(),"Stream1");

//alternative way to get outstream is
OutputStream os = service.getOutputStream("Stream1");
```

Interacting With Context Variables

Context is process flow context. Process flow variables can be accessed/created through it.

To get a Context Variable's value from a Process Flow

The following code snippet gives the information to access Process flow variables.

```
//To get value of process flow variable "orderName"
String    orderName = context.get("orderName");

//To get value of filePath attribute of FileSource activity after
//its execution. Here "fileSource1" is of FileSource type
String    activity1_filePath =
context.get("Service.fileSource1.filePath");
```

The activity's attribute can be accessed after its execution by following convention:

Service.<ActivityName>.<AttributeName>

Like **Service.fileSource1.filePath**

Its parameters are outlined in the table below.

Parameter	Description
Service	literal specifying the service
fileSource1	Name of the FileSource type activity.
filePath	Name of the attribute in the filePath

The following code snippet gives the information to get the value of the Process flow variables 'myMap' and 'FirstValue'.

```
//To get value of process flow variables 'myMap' and 'FirstValue'
import java.io.Map;
import java.io.Hashtable;
Map    map = (java.io.Map) context.get("myMap");
Hashtable ht= (Hashtable)context.get("FirstValue");
```

To set a context variable's value in process flow

The following code snippet gives the information to set the context variable's value in the process flow.

```
//To set value in process flow variable "orderName"
context.put("orderName", "purchaseOrder");

//To set value of filePath attribute of FileSource activity before its
execution. Here "fileSource1" is of FileSource type
import java.io.Map;
Map map= (Map)context.get("fileSource1.params");
```

```
map.put("filePath", "c:\\mypath\\myfile.txt");
context.put("fileSource1.params", map);
```

The activity's attribute can be set before its execution by following convention:

```
Map map= (Map)context.get("<ActivityName>.params");
map.put("<AttributeName>", "c:\\mypath\\myfile.txt");
context.put("<ActivityName>.params", map);
```

Its parameters are outlined in the table below.

Parameter	Description
ActivityName	Name of the activity like fileSource1
AttributeName	Name of the attribute like filePath

The following code snippet gives the information to set the value in the process flow variable.

```
//To set value in process flow variable
import java.io.Map;
import java.io.HashMap;
Map mymap = new HashMap();
mymap.put("customerName", "John");
mymap.put("customerId", "34344");
context.put("customerMap", mymap);
```



The created process flow variables may not be available in PD GUI. But these can be accessed through Put-Context-Variable.

Data Handling and Transformation

Custom plugin service can get two types of input streams:

XML/Plain Streams

An XML stream can be generated by schema type activity and can be passed to current custom plugin service. It can also be generated by file source activity if the file source activity is pointing to an XML file. Data into XML stream are passed into XML format.

For example:

FileSource → *TextSchema* → *ScriptedService* → *Target*

The Text Schema activity has the attribute *TransformerType*, which has value Stream2XML.

Plain stream can be generated by Source Type activity (except XML Source) and can be passed to current custom plugin service. No transformation is done in Plain stream.

For example:

FileSource → *ScriptedService* → *Target*

These streams are accessible by accessing input streams mentioned in section 2.

Record Streams

A record stream is a stream where records are written one by one. This stream can be accessed the same way as mentioned in section 2 but with minor changes.

```
import com.adeptia.indigo.io.RecordStreamFactory;
import com.adeptia.indigo.io.RecordInputStream;
import com.adeptia.indigo.io.Record;

// getting record stream by using RecordStreamFactory's static
method
//public static RecordInputStream createInputStream(InputStream
inputStream, Schema schema, String format)

RecordInputStream ris=
RecordStreamFactory.createInputStream(inputStream, null,
RecordStreamFactory.NATIVE_FORMAT);

Record rec= null;
while( (rec=ris. ReadRecord())!=null)
{

//code.....
}
```

For example

Record stream can be generated by TextSchema.

FileSource → *TextSchema* → *ScriptedService* → *Target*

Here TextSchema activity has following attributes values

```
Transformer Type -- SchemaStream2IntermediateSchema
Format - Native
```

Calling External Java or Native Programs

Calling Java programs

Java programs can be called directly from custom plugin service provided the class path has been set in Adeptia server environment. To know how to set the path in Adeptia Server environment refer to the section [Dependency](#). Custom plugin service also needs to add 'import' for the classes to be used in the custom plugin service.

Creating Java Packages

Steps to create Java packages

1. Create java packages to create classes in particular class hierarchy. For example, folders can be created in hierarchy *samples/javaprograms*.

2. Create a java program.

```
package samples.javaprogram;
public class sampleJavaProgram
{
    public String hello(String name)
    {
        return "Hello "+name+"!";
    }
}
```

3. Create *samples.jar* of this class and put in folder *ServerKernel/Samples*.
4. Set jar path in *launcher.properties*. Add line *samples/samples.jar*.
5. Restart Adeptia Kernel server.

Importing Java Packages into Custom plugin service

The sample code used to import Java Packages is outlined below

```
import samples.javaprogram.sampleJavaProgram;

String username = (String) context.get("username");
try {
    sampleJavaProgram cjp= new sampleJavaProgram();
    String greeting = cjp.hello (username);
    System.out.println("Result ---> " +username);
    context.put("result", username);
} catch (Exception e) {
    // customize exception message
    throw new Exception(e);
}
```

Calling Native programs

Native programs can be called by Custom plugin service using the following command:

```
Runtime.getRuntime().exec("executable command")
```

For more details, read java docs for *java.lang.Runtime* class.

For example,

```
Process Child = Runtime.getRuntime().exec("ADD.exe 44 56");
DataInputStream dis =new DataInputStream(child.getInputStream)
String sum =dis.readLine();
```

Here, executable file name is ADD.exe and arguments are 44 and 56.

Logging

Log instance is available with custom plugin service instance. This can be accessed by the following command:

```
import com.adeptia.indigo.logging.Logger;

//getting logger for the current custom plugin service
Logger log= service.getLogger();

//following type of message can be sent

//debug message
log.debug(msg);
```

The debug message will not appear in log messages if log level is set to *INFO* or *ERROR*

```
//info message
log.info(msg);
```

Info message will not appear in log messages if log level is set to *ERROR*

```
//error message
log.error(msg);
```

Exception Handling

Custom plugin service can be written to have complex logic. There are two types of errors associated with java code:

Compile Time Errors

Compile time errors are those errors which are related to syntax of the code as well as handling of exception. If there is any Compile time error in the java code, Custom plugin service is aborted on the first compilation. If there are checked exceptions (like *FileNotFoundException*), add them to handle in the code.

Runtime Errors

Runtime errors can be caught by java try catch block. But it is recommended to re-throw the error after adding customized message.

```
try{

}catch(Exception e){
    throw new Exception(message, e);
}
```

Dependency

If customized code has dependency upon external classes or jar files, then path of those classes or jar files needs to be set in the Adeptia Server environment.

Steps to set the path in the Adeptia Server environment

1. Open launcher.properties file from `../../AdeptiaServer4.6/serverkernel/etc` folder. Here first two dots represent drive letter and second two dots represent the folder where Adeptia Server is installed.
2. Add the absolute path or relative path (with respect to `serverkernel` folder) of the external class or the jar files.
3. Save the file and close it.
4. Restart the Adeptia Server kernel.

Examples

Example 1: Sending Input Stream Data to two Output Streams

Description

Stream coming from File Source (zip file) and extracting two files and sending it two output streams.

Setup

1. Custom plugin service should have one incoming input stream and two output streams.
2. Check PD for current custom plugin service activity for attribute `Source Stream` (for input) and `Stream Names` (for output)

Code

```
import java.io.BufferedInputStream;
import java.util.zip.ZipInputStream;
import java.util.zip.ZipEntry;

ZipEntry ze = null;

        //      Reading the input data
        //      inputStream - represents the input data Stream,
which is
        //      implicitly available to Script
        //      Any data processing logic

        ZipInputStream zin = new ZipInputStream(new
BufferedInputStream(
            inputStream));
```



```
while ((ze = zin.getNextEntry()) != null) {
    if (ze.getName().equals("FirstFile.txt")) {
        //getting size
        int filesize = (int)ze.getSize();
        byte[] bytes = new byte[ n];
        zin.read(bytes,0,n);
        //    Writing output data to output stream read
by another
        // activity.
        //    service - this is the "Custom plugin
service" Service object,
        //    which is also available implicitly to the
script
        service.write(bytes, "Stream1");
        //    Note- Stream1 should be output stream,
this can be
        //    checked in PD for current custom plugin
service activity
    }

    if (ze.getName().equals("SecondFile.txt")) {
        //    getting size
        int filesize = (int)ze.getSize();
        byte[] bytes = new byte[ n];
        zin.read(bytes,0,n);
        //    Writing output data to output stream read
by another
        // activity.
        //    service - this is the "Custom plugin
service" Service object,
        //    which is also available implicitly to the
script
        service.write(bytes, "Stream2");
        //    Note- Stream2 should be output stream,
this can be
        //    checked in PD for current custom plugin
service activity
    }
}

zin.close();
```

Example 2: Getting XML as input stream and sending the data to one output stream

Description

Stream coming from Schema type activity or File Source (XML File) activity and parsing and creating DOM parser object to do specific task and then sending data to output stream.

Setup

1. Custom plugin service should have one incoming input stream and one output stream.

2. Check PD for current custom plugin service activity for attribute Source Stream (for input) and Stream Names (for output)

Code

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import org.w3c.dom.Document;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

//getting factory instance
DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();

    //getting parser
    DocumentBuilder parser = factory.newDocumentBuilder();

    //getting document
    Document doc = parser.parse(inputStream);

    //getting all employee nodes
    NodeList nlist=doc.getElementsByTagName("Employee");

    int length=nlist.getLength();
    //getting employee name
    for(int i=0;i<length;i++){
        Node node= nlist.item(i);
        NamedNodeMap nodeMap= node.getAttributes();
        Node nameNode= nodeMap.getNamedItem("name");
        String name = nameNode.getNodeValue();
        //writing employee names
        service.write(("Employee "+i+": "+
name+"\n").getBytes(),"stream1");
    }
}
```

Input XML Data

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Root xmlns:java="http://xml.apache.org/xslt/java">
<Record>
  <Employee name="John" age="22" salary="6000" />
</Record>
<Record>
  <Employee name="Peter" age="35" salary="12000" />
</Record>
<Record>
  <Employee name="James" age="45" salary="8000" />
</Record>
<Record>
  <Employee name="Ricky" age="21" salary="10000" />
</Record>
```

```
</Record>
<Record>
  <Employee name="Robert" age="25" salary="5000" />
</Record>
<Record>
  <Employee name="Nick" age="35" salary="6000" />
</Record>
<Record>
  <Employee name="Mike" age="42" salary="9000" />
</Record>
</Root>
```

Example 3: Getting Records from input record stream and sending the data to output stream

Description

A Schema type activity can send record stream to custom plugin service. Attributes of Schema activity (check PD for schema activity) namely “Transformer Type” and “Format” can be set to “SchemaStream2IntermediateTransformer” and “NATIVE” respectively.

Setup

1. Custom plugin service should have one incoming input stream and one output stream.
2. Check PD for current custom plugin service activity for attribute
3. Source Stream (for input) and Stream Names (for output)
4. Record stream is coming from Text Schema having three fields *name*, *age*, and *salary* having format string, number, and number respectively.

Record Format

A record can have multiple fields and each field has a value. This value can be used to create

java.lang.String, *java.util.Date* or *java.lang.Double* objects

Generic method to get object

```
Object obj= record.getObject("field_name");
```

Code

```
import com.adeptia.indigo.io.RecordStreamFactory;
import com.adeptia.indigo.io.RecordInputStream;
import com.adeptia.indigo.io.Record;
import java.io.EOFException;

/*
 * getting record stream by using RecordStreamFactory's following
static
 * method
 *
 * public static RecordInputStream createInputStream(InputStream
```

```
* inputStream, Schema schema, String format)
*/

RecordInputStream ris = RecordStreamFactory.createInputStream(
    inputStream, null,
    RecordStreamFactory.NATIVE_FORMAT);

    Record record = null;
    try {
        //header present
        boolean headerRecordCheck=false;

        //writing to output stream
        service.write("Writing record \n".getBytes(),
"Stream1");

        //reading record
        while ((record = ris.readRecord()) != null) {
            //check if header record is present
            if(headerRecordCheck){
                headerRecordCheck=false;
                continue;
            }

            // getting field values
            String name = (String)record.getObject("name");
            String age = (String)record.getObject("age");
            String salary =(String)
record.getObject("salary");

            service.write((" Name:--" + name ).getBytes(),
"Stream1");
            service.write((" Age:--" + age).getBytes(),
"Stream1");
            service.write((" Salary:--" +
salary).getBytes(), "Stream1");
            // new line
            service.write("\n".getBytes(), "Stream1");

        }
    } catch (EOFException e) {
        //end of record stream reached
        service.write("End of records \n".getBytes(),
"Stream1");

    } catch (Exception e) {

        //error in stream handling
        throw e;

    }
}
```

Example 4: Getting Records from input record stream and sending the data to output stream as XML output

Description

A Schema type activity can send record stream to custom plugin service. Attributes of Schema activity (check PD for schema activity) namely "Transformer Type" and "Format" can be set to "SchemaStream2IntermediateTransformer" and "NATIVE" respectively. Custom plugin service will read the record stream and it will convert it to XML stream.

Setup

1. Custom plugin service should have one incoming input stream and one output stream.
2. Check PD for current custom plugin service activity for attribute
3. Source Stream (for input) and Stream Names (for output)
4. Record stream is coming from Text Schema having three fields *name*, *age* and *salary* having format string, number, and number respectively.

Record Format

A record can have multiple fields and each field has a value. This value can be used to create

java.lang.String, java.util.Date or java.lang.Double objects

Generic method to get object

```
Object obj= record.getObject("field_name");  
Object obj= record.getObject("field_name");
```

Code

```
import com.adeptia.indigo.io.Record;  
import com.adeptia.indigo.io.RecordInputStream;  
import com.adeptia.indigo.io.RecordStreamFactory;  
import com.adeptia.indigo.io.XmlEncoder;  
import com.adeptia.indigo.services.transform.ScriptedService;  
  
//getting the current custom plugin service instance  
ScriptedService scriptedService = (ScriptedService) service;  
  
/*  
 * getting record stream by using RecordStreamFactory?s following  
 static  
 * method  
 * public static RecordInputStream createInputStream(InputStream  
 * inputStream, Schema schema, String format)  
 */  
    RecordInputStream ris =  
RecordStreamFactory.createInputStream(  
    inputStream, null,  
RecordStreamFactory.NATIVE_FORMAT);  
  
    Record record = null;
```

```
        XmlEncoder xme = null;
        try {
            /*
             * creating instance of XmlEncoder to send XML data use
            constructor
             * public XmlEncoder(OutputStream ostream, String
             * characterSetEncoding)
             */

            xme = new XmlEncoder(scriptedService.getOutputStream("Stream1"),
            "ISO-8859-1");

            //      writing xml header
            xme.writeProlog();

            //      reading record
            while ((record = ris.readRecord()) != null) {
                //      writing record
                xme.writeRecord(record);
            }
        } catch (EOFException e) {
            //end of record stream reached
            xme.close();

        } catch (Exception e) {

            //error in stream handling
            throw e;

        }
    }
```

Example 5: Calling a native program and passing input parameters and getting result back

Description

A native program can be called from java program and result can be used back in program.

Setup

1. A native program .exe/.bat should accessible.
2. Custom plugin service generates an output stream "Stream1" and consume no stream.

Code

```
import java.io.BufferedReader;
import java.io.InputStreamReader;

// store the result in string buffer
final StringBuffer result = new StringBuffer();
final StringBuffer errorResult = new StringBuffer();
```

```
String osName = System.getProperty("os.name");
System.out.println("OS name " + osName);

String commandStr = null;
/* for windows 2000 use "cmd" and for windows 95 use
"command" for
    command shell
*/
if (osName.equalsIgnoreCase("WINDOWS 95")) {
    commandStr = "command";
} else if (osName.equalsIgnoreCase("WINDOWS 2000")
    || osName.equalsIgnoreCase("WINDOWS NT")
    || osName.equalsIgnoreCase("WINDOWS XP")) {
    commandStr = "cmd";
}

/*
 * To get window directory command output
 * creating command string
 */
String[] cmd = new String[4];
cmd[0] = commandStr ;
cmd[1] = "/c"; //Carries out the command specified by
string and then terminates
cmd[2] = "dir"; //Path to exe/bat file
cmd[3] = "c:\\"; //argument can be passed from context
context.get(arg1);

// getting runtime object, for more detail see java docs
for java.lang.Runtime
Runtime rt = Runtime.getRuntime();
//     executing command
Process p = rt.exec(cmd);

// starting error and outputstream streams in separate
threads.
final BufferedReader outputStream = new BufferedReader(
    new InputStreamReader(p.getInputStream()));
final BufferedReader errorStream = new BufferedReader(
    new InputStreamReader(p.getErrorStream()));

Thread errorThread = new Thread(new Runnable() {
    void run() {
        try {
            String err_str = "";
            while ((err_str = errorStream.readLine()) !=
null) {
                errorResult.append(err_str+"\n");
            }
        } catch (IOException e) {
            System.err.println(e);
        }
    }
})
```

```

    });
    // starting error thread
    errorThread.start();

    Thread outputThread = new Thread(new Runnable() {
        void run() {
            try {
                String out_str = "";
                //capturing process output
                while ((out_str = outputStream.readLine()) !=
null) {
                    result.append(out_str+"\n");
                }
            } catch (IOException e) {
                System.err.println(e);
            }
        }
    });

    // starting output thread
    outputThread.start();
    // waiting for process to end
    int k = p.waitFor();

    System.out.println("Process status: " + k);
    System.out.println("Process output:\n" + result);
    System.out.println("Error Stream:\n" + errorResult);
    System.out.println("End.....");

    if (k > 0) {
        /*
        * if process status is greater than zero , then
this could be an
        * error depending upon the called application
        */

        //throw exception if necessary

        //throw new Exception(errorResult.toString());
    }
    //setting result in context
    context.put("result", result);
    //writing to output stream
    service.write(result.toString().getBytes(), "Stream1");

```



Some applications do not set process status to greater than zero in case of error occurs.

Some applications do not send error to the error data streams but they send them to the output stream.

CUSTOM API FOR CUSTOM JAVA DEVELOPMENT

Adeptia uses custom APIs for custom Java development. They can be categorized as:

- [Java API for Adeptia product](#)
- [Java API for Third Party Libraries](#)

A few of these APIs are covered in this document. More APIs will be added in future versions.

JAVA API FOR ADEPTIA PRODUCT

Java APIs for the Adeptia product are available in a separate document (**adeptiaAPI.zip**) in a java doc format. Following APIs are included in this version:

- com.adeptia.indigo.utils.MBeanUtils
- com.adeptia.indigo.system.IndigoConfig
- com.adeptia.indigo.system.Context
- com.adeptia.indigo.storage.EntityManager
- com.adeptia.indigo.storage.EntityManagerFactory
- com.adeptia.indigo.storage.TypedEntityId
- com.adeptia.indigo.logging.Logger
- com.adeptia.indigo.utils.TransactionInformation

JAVA APIs FOR THIRD PARTY LIBRARIES

In addition to the inbuilt Java APIs, Adeptia supports certain external APIs for custom coding. These are outlined in the table below.

Third Party API	Version	Source	Description
JRE	1.5	Sun	It is the Java runtime environment.
HSQLDB	1.8	SourceForge	It is used as an embedded database. It is a relational database management system written in Java.
Castor	0.9.6	Codehaus	It is used for Java-to-SQL persistence with JDO approach. It is an Open Source data binding framework for Java[tm]. It's the shortest path between Java objects, XML documents and relational tables. Castor provides Java-to-XML binding, Java-to-SQL persistence, and more.
Log4J	1.2.8	Apache	It is a Java-based logging utility and is used for logging.
Jasper	4.0.4	SourceForge	Jasper Reports is a powerful open source

			Java reporting tool that has the ability to deliver rich content onto the screen, to the printer or into PDF, HTML, XLS, CSV and XML files.
VFS	1	Apache	The Commons VFS provides a single API for accessing various different file systems.
Java Mail API	NA	Sun	This is the Java API for mail.

JRE

Jar Files Required

No jar files are required for this API

License Type

<http://www.java.com/en/download/license.jsp>

Website

<http://www.java.com/en/download/manual.jsp>

Download Link

<http://www.java.com/en/download/manual.jsp>

HSQldb

Jar Files Required

- hsqldb-1.7.1.jar
- jtds-1.2.jar

License Type

Based on BSD License

<http://hsqldb.org/web/hsqLicense.html>

Website

<http://hsqldb.org/>

Download Link

http://sourceforge.net/project/showfiles.php?group_id=23316&release_id=339171

Castor

Jar Files Required

- castor-0.9.6.jar

License Type

Apache 2.0 License.

<http://www.apache.org/licenses/LICENSE-2.0>

Website

<http://www.castor.org/index.html>

Download Link

<http://www.castor.org/download.html>

Log4J

Jar Files Required

- log4j-1.2.8.jar
- log4j-jdbcplus.jar

License Type

Apache 2.0 License.

<http://www.apache.org/licenses/LICENSE-2.0>

Website

<http://logging.apache.org/log4j/docs/index.html>

Download Link

<http://logging.apache.org/log4j/docs/download.html>

Jasper

Jar Files Required

- jasper-compiler-4.0.4.jar
- jasper-runtime-4.0.4.jar

Website

<http://jasperforge.org/>

Download Link

http://www.jasperforge.org/index.php?option=com_content&task=section&id=16&Itemid=277

VFS

Jar Files Required

- commons-vfs-1.0-dev.jar
- commons-httpclient-2.0.2.jar
- commons-net-1.4.1.jar
- jsch-0.1.5.jar
- jakarta-slide-webdavlib-2.1.jar
- jdom-1.0.jar
- jdom-1.0b8.jar
- jcifs-1.2.9.jar

License Type

Apache 2.0 License.

<http://www.apache.org/licenses/LICENSE-2.0>

Website

<http://jakarta.apache.org/commons/vfs/>

Download Link

<http://jakarta.apache.org/commons/vfs/download.html>

Java Mail API

Jar Files Required

- mail.jar

License Type

<http://www.java.com/en/download/license.jsp>

Website

<http://java.sun.com/products/javamail/>

Download Link

<http://java.sun.com/products/javamail/downloads/index.html>

DESIGN PATTERNS

A design pattern offers readymade solutions to standard business scenarios, encountered while developing complex solutions.

OBJECTIVE

- Provide readymade solutions to standard problems
- Identify which service to use in a business scenario
- Provide guidelines on how to use a service in a business scenario
- Provide comprehensive information associated with using a service

BUSINESS SCENARIOS

Business scenarios refer to the standard problems that are commonly encountered by a developer, while developing complex solutions. These can be divided into the different groups. A few scenarios are described in this version. More scenarios will be covered in future versions.

The scenarios are divided into the following groups:

- [Developing complex GUI for Human workflow](#)
- [Publishing a process flow as Web Service](#)
- Event Triggered Process Flows

This section covers common problems that occur when triggering a process flow on an event. One such problem and its solution is described in this version. Others will be covered in future version.

- [Triggering a process flow on database trigger \(Database Event\)](#)
- Triggering a process flow on File Modified event on FTP (FTP Event)
- Triggering a process flow on File Modified event on local system (File Event)
- Triggering a process flow on email
- Triggering a process flow on HTTP event

- Mapping Complex scenarios

Common problems that occur when mapping fields are outlined as:

- [Using XSL template for padding fields](#)
- JTA Rollback

Common problems that occur when using JTA Rollback are outlined as:

- Using same database related tables under Transaction

- Using different database tables under Transaction

- Stored Procedure

Common problems that occur when using stored procedures are outlined as:

- Using Stored procedure in Database source for SQL Server

- Jasper report design and generation

- Resubmitting erroneous records after correction

- Usage of PutContextVar

Common problems that occur when using PutContextVar are outlined as:

- Dynamically overriding activity parameters
- Dynamically overriding activity

- Dynamically overriding activities in a process flow

Common problems that occur when dynamically overriding activities in a process flow are outlined as:

- Overriding schema to handle various data format
- Overriding Source to decide data source at run time
- Overriding target to decide data target at run time for dynamic content routing
- Overriding Mapping to apply transformation as per incoming data format

- Dynamically assigning a human workflow task to a user

DEVELOPING COMPLEX GUI FOR HUMAN WORKFLOW

Objective

This document describes methods to create complex forms in Human Workflow.

Requirements

- To allow a user to review the data in a simple layout in the browser
- To allow a user to update the data in Browse after review and send it back to the process
- To allow a user to add new data fields in Browser, insert value in data fields and send it back to the process
- To organize the data in two layouts:
 - Grid Layout
 - It supports Flat data
 - It supports Hierarchical data
 - Form Layout
 - Form layout supports navigation from one page to another
 - Form can have pre-populated values that comes from database based on parameters passed from process flow

Approaches

Most of the practical scenarios can not be addressed by dynamic HTML only. You require additional dynamic behavior of the web pages and interaction with server end to get the required information. Adeptia BPM platform supports this and allows a user to integrate the web application that user has created using JSP or AJAX using GWT outside the product in standard editor.

Use of custom JSP

This approach is described with the help of an example.

Example scenario

Case

You have a business process that gets data in excel format. Data is validated and verified and the correct data is loaded into the database. The data records that are erroneous are to be reviewed and updated online and integrated as Human Workflow activity in the business process.

Assumptions

- A dummy table "ERROR_DATA" is created to capture the error records
- All the error record is populated in table.
- The table has additional column to store process flow domain ID (PID)

- The PID is set in process flow context using Custom Plugin (scripted service).

Example code

```
import com.adeptia.indigo.system.*;
ObjectAddress obj=(ObjectAddress)context.get("TransactionAddress");
String pid=obj.getTransactionPid();
context.put("PID",pid);
```

Creating JSP

1. Get the process flow Domain ID. This will be passed from process flow
String PID= request.get("PID");
2. Query the database to get all the error records for this PFID
String dbInfoID= "12334455566677778"
3. This is an ID of the "DatabaseInfo" activity that has been defined in the Adeptia Sever. It captures all the parameters to connect to database.
4. Use Adeptia API to read all database connectivity parameter and write JDBC code to query database.
*Select * from ERROR_DATA where PFDID= PFID*
5. Render the result set in required layout for example grid. Keep all the values that can be updated as editable.
6. Give user a submit button that he can use after review and update.
7. The update data is pushed back to database using "Insert query"
For example, you have created "Review.jsp".

Integrating with Adeptia

1. Deploy the JSP by copying the jsp to:
"ServerKernel/web/" repository
2. Pass the information from process flow context to custom JSP (for example Review.jsp)
 - Define all the context variables that need to be passed to custom JSP in Human Workflow activity form.
 - Call the script function in HTML of Human Workflow activity. For example, in the given case you need to pass PID to custom JSP. You have already set PID value in context.
3. Use the following HTML code
 - The Review.jsp is invoked through review link in java script function
 - Write a Java script function is mandatory because it will allow passing dynamic parameters to custom JSP through URL rewriting
 - PID is passed to custom JSP. You can pass any number of variables

```
<link rel="stylesheet" href="css/ui.css" type="text/css" />
<table width="100%" border="0" background="images/top-streach.gif"
cellspacing="0" cellpadding="0">
<tr>
```

```

        <td><img src=images/adeptia-logo.gif width="354"
            height="37"></td>
    </tr></table><br>
    <table width="100%" border="0" cellspacing="0"
        cellpadding="0">
        <tr>
            <td height="18" class="headings">Template
                Review</td></tr></table><br><br>
        <a href='javascript:review();'>
            <font color=blue>Review</font></a>
        <table width='100%' border=0><tr height='10'><td></tr><tr
            ><td><form name=HTMLForm><TABLE><TR><td>Processflow
                DomainId</td><td><input type=text name=PID
                readOnly></TD></TR><TR><TD colspan=2><input type=button
                name='partialSubmit' value='Save Task'><input type=button
                name='fullSubmit' value='Complete
                Task'></TD></TR></TABLE></form></td></tr></table>
    </Form>
    <script>
    function review()
    {
        var pid=document.HTMLForm.PID.value;
        var vWinTrans=window.open("Review.jsp?PID="+pid
        ,"_blank","toolbar=yes,location=no,directories=no,status=no,menubar=
        yes,scrollbars=yes,resizable=yes,copyhistory=no");
    }
    </script>

```

Execution of Human Workflow

While executing the Human Workflow activity, a task will be added to user task. To execute this task:

1. Login into Adeptia.
2. Go to the Task Manager and execute the task. A basic page / Framework page is displayed (This page can be design quite well with HTML expertise).
3. Invoke the custom jsp as a popup in a new window by clicking the URL on the page.
4. Once you have navigated through required custom pages, you can close them.
5. To complete the Human Workflow activity, you need to take the control back to Framework page and click the "Complete Task" button.



In complex Human Workflow activities where multiple forms are used, the Framework page can be treated as the Adeptia Home page (It is just one design best practice).

Enhancements in Adeptia

- Allow session to pass the process flow domain ID to any custom JSP invoked from Human Workflow Html form.
- Allow Human Workflow Html form to keep value for hidden variables so that it can be passed to the custom JSP invoked from Human Workflow Html form.

WORKING WITH WEB SERVICE PUBLISHER

A business process can be treated as service and can be made available in Enterprise. The most interoperable way is to make it available as web service. Adeptia provides “WsProvider” to publish any business process as web service.

Objective

This document describes the method of publishing a business process as a web service.

Tasks

1. Creating a process flow
2. Creating a security policy (optional)
3. Creating a ‘WsProvider’

Example

Case

You have a business process that takes the Auto Insurance Policy application and returns back the Policy Quote. You want to publish this process as a web service.

Solution

To publish the process as a web service, you need to design a process from perspective that it is similar to a function that takes input, processes it and generates output. While designing, you need to remember:

- Web service always takes input as XML data.
- Web service always generate output as XML data

You need to define both XML data formats, the one that the web service will take as input and the format that the web service will generate as output. You can do this using the Adeptia XML schema activity. You need to define input and output XML schemas.

Creating process flow

Based on function analogy a process flow will have four parts:

1. Defining input and output XML schemas
2. Getting input data from client request
3. Data processing logic
4. Sending output back as response to client


Defining Input and Output XML schema

You need to use Adeptia XML schema activity to define XML schemas for input and output. To do that you need to define:

- The XSD for the input XML. For example, in given scenario you have an Auto Insurance “Policy application” as an input whose XML schema is defined in [Appendix A](#).
- The XSD for the output XML. For example, in given scenario the output generated is the “Policy Quote” .The output XML schema is defined in [Appendix B](#).

Getting Input Data from Client Request

In the process flow you need to use “ContextSource” as activity. You need to define a “parameterName” in the “ContextSource” property. Web service client input data will be assigned to this parameter. The context source will read it from this “parameterName” and pass the data to any further activity for data processing. This is displayed in the Figure below.



The “parameterName” user here will be used while defining the WsProvider activity as “input variable name”

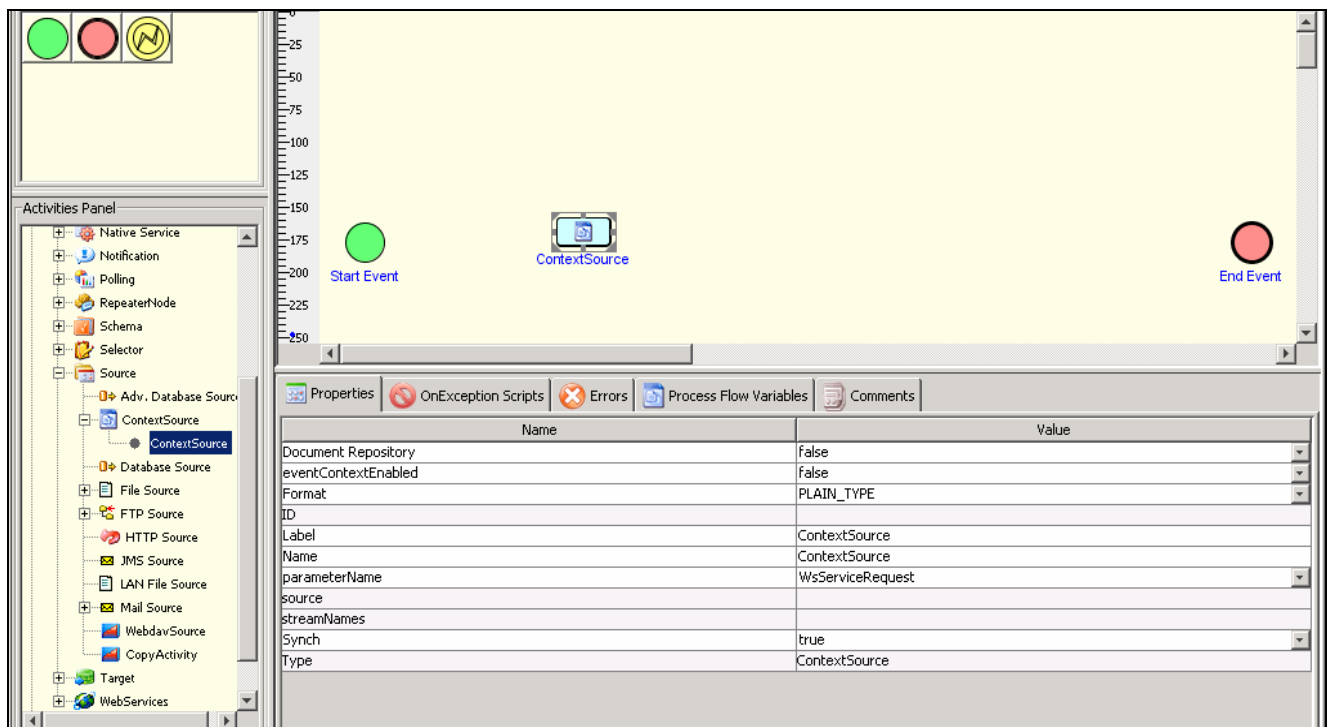


Figure: Adding context source activity and defining it in process flow


Data Processing logic

The input data is then processed to generate a required output. The data processing logic can include data transformation activity and other Adeptia activity or group of activities. To process the input data, it will use input XML schema activity. It will use output XML schema activity to generate output data in XML format. The output generated after processing the input data is set in the Process flow context through “contextTarget”.

Sending Output back as response to client

In the process flow, you need to use “ContextTarget” as activity. You need to define a “parameterName”. Web service client output data will be assigned to this “parameterName”.

The “ContextTarget” will take data from the data processing activity and assign to the “parameterName” defined above. This is displayed in the Figure below.

 The “parameterName” user here will be used while defining the WsProvider activity as “output variable name”

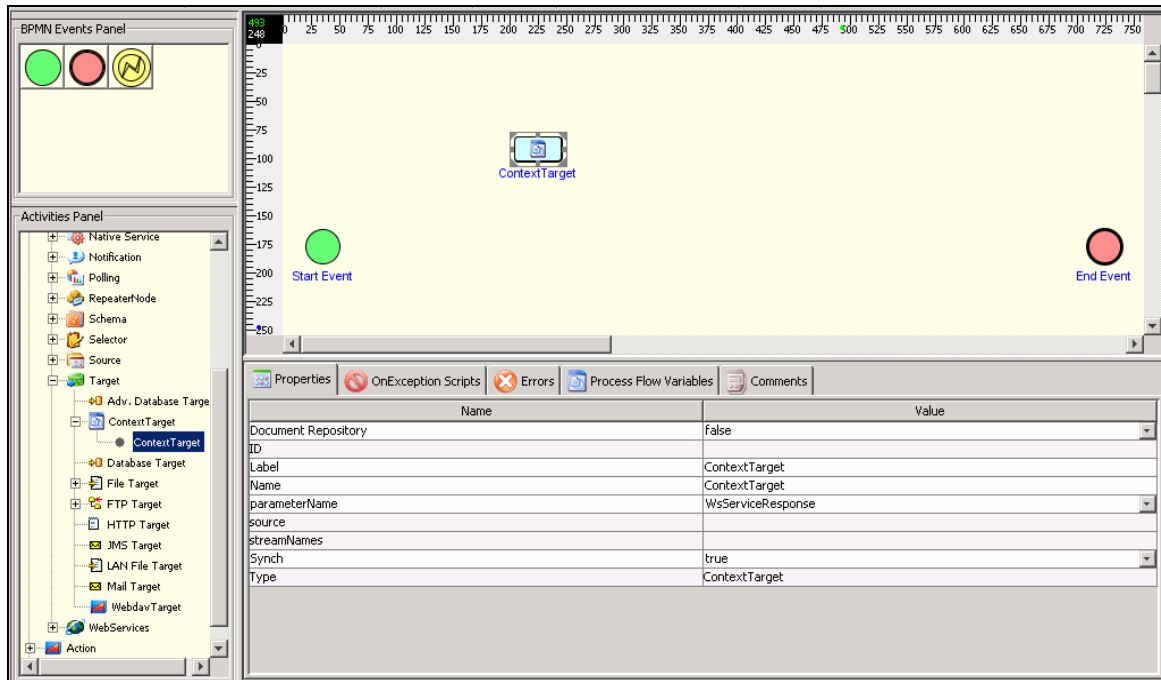



Figure: Adding “contextTarget” activity and defining it in process flow

Once you connect all these activities in the process flow, you can publish the process flow.

Creating Security Policy (Optional)

Since Web Services expose crucial business information, Web services security is critically important. A Web service can be secured using Security Policy activity. If you want to secure the web service then you need to create an appropriate security policy before publishing the Web Service using the Web service provider.

 For details on creating Security Policy, refer to the *Creating Security Policy Activity for Web Service* section in the User Manual.

Creating WsProvider

Web Service Provider is used to publish a process flow that Web Service Client can access. Once a Web Service is published, it creates a WSDL and makes it available to the Adeptia Server users. You can use this WSDL to invoke the Web Service. The Web Service can be published in two modes:


- *Synchronous*: In case the Web Service is published in synchronous mode the consumer waits for the completion of the process flow and hence for the output of the process flow.
- *Asynchronous*: In case the Web Service is published in asynchronous mode the consumer does not wait for the process flow to be completed. Thus consumer gets

only a Co-relation ID not the output of the process flow. Later on, using this Co-relation ID, consumer can get the output.

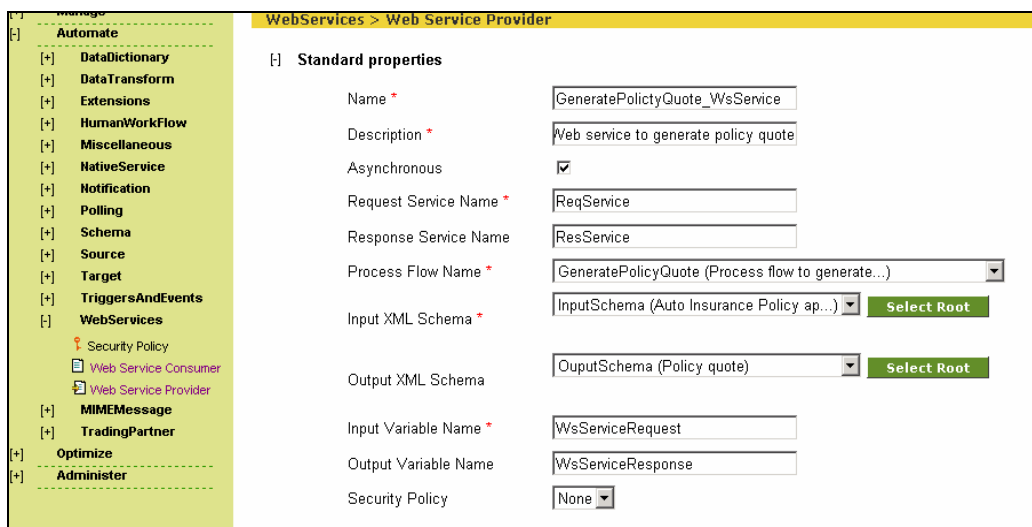
To create Web Service Provider:

1. Enter the name and description of the new Web Service Provider activity in the *Name* and *Description* fields respectively.
2. Uncheck the *Asynchronous* checkbox if you want to publish Web Service in Synchronous mode or else the Web Service will be published in asynchronous mode.
3. Enter the request service name and response service name in the *Request Service Name* and *Response Service Name* respectively. The Web Service will be published with the respective service name given in *Request/Response Service Name* field.
4. Select the process flow, which you want to publish as Web Service from the *Process Flow Name* drop-down list. (In this scenario, the process flow will be the above created Process flow).
5. Select Input XML Schema from the *Input XML Schema* drop-down list. This XML Schema corresponds to the XML Input provided by Web Service consumer activity (In this scenario it will be the schema defined for Auto Insurance "Policy Application").
6. Select the Output XML Schema from the *Output XML Schema* drop-down list (In this scenario it will be the schema defined for "Policy Quote").
7. Enter the Input and Output Variables in the *Input Variable* and *Output variable* fields respectively (The variable name should be same as that of the "parameterName" value of context source and context target respectively).
8. Select the Security Policy activity from the *Security Policy* drop-down list.

The data entry is displayed in the Figure below.



If any security policy is not selected, then the web service is published in anonymous mode, i.e., the web service published is not secure and any web service client can invoke the published Web service without any Adeptia authentication.



The screenshot shows the configuration window for a Web Service Provider. The left sidebar contains a tree view with categories like Automate, DataDictionary, Extensions, HumanWorkflow, Miscellaneous, NativeService, Notification, Polling, Schema, Source, Target, TriggersAndEvents, WebServices, Security Policy, Web Service Consumer, Web Service Provider, MIMEMessage, TradingPartner, Optimize, and Administrator. The main area is titled 'WebServices > Web Service Provider' and shows 'Standard properties' with the following fields:

- Name: GeneratePolicyQuote_WsService
- Description: Web service to generate policy quote
- Asynchronous:
- Request Service Name: ReqService
- Response Service Name: ResService
- Process Flow Name: GeneratePolicyQuote (Process flow to generate...)
- Input XML Schema: InputSchema (Auto Insurance Policy ap...) [Select Root]
- Output XML Schema: OuputSchema (Policy quote) [Select Root]
- Input Variable Name: WsServiceRequest
- Output Variable Name: WsServiceResponse
- Security Policy: None

Figure: Creating WsProvider

Execution of WsProvider


Any Web service client can be used to invoke the Adeptia Process flow published as a Web service. To invoke the published web service the client requires the path where the WSDL is stored. The user can get the http path of the WSDL through the manage page of Web service Provider activity (see Figure below).



#	Activity Name	Description	Action	WSDL	User/Group	Created	Modified
1	GeneratePolicyQuote_Ws Service	Web service to generate policy quote	View Revision Dependent History Activities Edit Delete	ViewDownload Service to post data: ViewDownload Service to receive data: ViewDownload	admin [administrators]	03/07/2007 12:51:55	03/07/2007 12:51:55

Figure: Manage Web Service Provider

The ViewDownload link can be used to know the WSDL path. User can save the WSDL in his local system and access the web service through that location.



Adeptia server can also be used to invoke the published web service through the Web service Consumer activity. For details on creating Web Service Consumer activity, refer to the *Creating Web Service Consumer Activity* section in the User Manual.

Appendix A (Input XML schema)

Definition of input XML Schema is displayed in the Figure below.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="Auto Insurance Policy">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="AgentName"/>
        <xs:element name="AgentEmail"/>
        <xs:element name="CustomerName"/>
        <xs:element name="CustomerState"/>
        <xs:element name="DriverRecord"/>
        <xs:element name="VehicleType"/>
        <xs:element name="VehicleModel"/>
        <xs:element name="VINNumber"/>
        <xs:element name="VehicleValue"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

```
</xs:complexType>
</xs:element>
</xs:schema>
```

Appendix B (Output XML schema)

Definition of output XML Schema is displayed in the Figure below.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="Policy Quote">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="AgentName"/>
        <xs:element name="AgentEmail"/>
        <xs:element name="CustomerName"/>
        <xs:element name="CustomerState"/>
        <xs:element name="DriverRecord"/>
        <xs:element name="VehicleType"/>
        <xs:element name="VehicleModel"/>
        <xs:element name="VINNumber"/>
        <xs:element name="VehicleValue"/>
        <xs:element name="Premium"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```


USING DATABASE EVENT

A Database event enables you to schedule a process flow to be triggered when a record is inserted, updated or deleted in a database table.

Objective

- To analyze the various methods of providing solutions when using Database event in business scenarios, on different database servers
- To configure settings for different servers and their drivers to be used when using Database event
- To identify the advantages and limitations of using a particular method to create Database event

Tasks For Using Database Event

The tasks involved in using a Database event are outlined as:

- Creating Database Drivers and Database Info activities
- Creating Database Event
- Creating Process Flow
- Registering Database Event
- Activating Database Event

Creating Database Drivers and Database Info

The Database Driver and Database Info activities need to be created before creating the Database event.

Database drivers that are used for a database event can be classified as:


- Standard Type 4 Drivers
- JDBC-ODBC Drivers

Standard Type 4 Drivers

"Type 4" drivers are pure Java drivers. They are database-specific, which implies that each database type has a separate dedicated "Type 4" driver. Some of these drivers are available for free on the web, whereas some need to be purchased under a commercial license. The free "Type 4" drivers that can be downloaded from the web are listed in the table below.

A list of the standard Type 4 Drivers with their class name and jar files, and the Server URL required for Database Info, supported by a Database Event are outlined in the table below.

Database Name	Driver Main Class Name	Driver Jar Files	Server URL
MSSQL	com.microsoft.jdbc.sqlserver.SQLServerDriver	msbase.jar, mssqlserver.jar, msutil.jar (Bundled with Adeptia)	jdbc:microsoft:sqlserver://<databaseServerName>:<portNumber>;DatabaseName=< DatabaseName >
SQL(using JTDS)	net.sourceforge.jtds.jdbc.Driver	jtds-1.2.jar	jdbc:jtds:sqlserver://<databaseServerName>:<portNumber>/DatabaseName=< DatabaseName >
Oracle	oracle.jdbc.driver.OracleDriver	classes12.jar (Bundled with Adeptia)	jdbc:oracle:thin:@<databaseServerName>:<portNumber>:<databaseName>
DB2	com.ibm.db2.jcc.DB2Driver	db2jcc.jar	jdbc:db2://<databaseServerName>:<portNumber>/<databaseName>
HSQldb	Org.hsqldb.jdbcDriver	hsqldb-1.8.0.1.jar (Bundled with Adeptia)	jdbc:hsqldb:hsqldb://<databaseServerName>:<portNumber>
Sybase	com.sybase.jdbc2.jdbc.SybDriver	jconn2.jar	jdbc:sybase:Tds:<serverName>:<portNumber>/?jconnect_version=5

 In addition to the listed "Type 4" drivers, you can also use other "Type 4" drivers that are available for these databases.


JDBC-ODBC Drivers

A JDBC-ODBC bridge driver enables a Java application to communicate with any ODBC complaint database. ODBC represents open database connectivity. Currently, mostly all databases are ODBC complaint.

Pre-requisites

Before creating a JDBC-ODBC Driver, you need to ensure that the following pre-requisites are conformed to:

- The database and Adeptia product must be installed on the same machine.
- The DSN must be created on the client machine.

 For details on creating a DSN, refer to [Appendix 1](#).

A list of JDBC-ODBC Drivers with their class name and jar files, and the Server URL required for Database Info, supported by a Database event are outlined in the table below.


Database Driver	Driver Main Class Name	Driver Jar Files	Server URL
MS Access	sun.jdbc.odbc.JdbcOdbcDrivers	NA	jdbc:odbc: <DSN name > jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};DBQ=<Microsoft Access DB filename (.mdb) with full path>
MS Excel	sun.jdbc.odbc.JdbcOdbcDrivers	NA	jdbc:odbc:Driver={Microsoft Access Driver (*.xls)};DBQ=<Microsoft Excel filename (.xls) with full path>


An example Server URL for each database is outlined in the table below.

Database	Server URL
MS Access Database	jdbc:odbc:Driver={MicroSoft Access Driver (*.mdb)};DBQ=\\dbserver\project1\project1.mdb
MS Excel Database	jdbc:odbc:Driver={MicroSoft Excel Driver (*.xls)};DBQ=\\dbserver\project1\project1.xls

There are a few limitations in accessing Excel data with JDBC-ODBC bridge drivers. These are outlined as:

- You cannot update the data into target excel sheet.


	This driver is not recommended for production scenarios.
---	--

	If you are using paid drivers, you have to manually update the <i>Driver Class Name</i> and <i>Server URL</i> fields.
---	---

Creating Database Event

Database event can be created in two ways: -

- Using SQL Query
- Using SQL Trigger

	When creating a Database event, you need to ensure that the User ID and password must be defined with appropriate permissions. They should be assigned <i>Read Only</i> permission when using <i>SQL Query</i> and <i>Read</i> and <i>Create</i> permissions for SQL Trigger.
---	---

Using SQL Query


A SQL query is a command that is defined using a valid SQL statement. It is used to define trigger criteria in a process flow.

When to use SQL Query

There are certain scenarios in which you can use only the SQL Query method to define the Database event. These are outlined as:

- Trigger Creation is not supported by the Database or by Adeptia

At times, the database being used does not support trigger creation, or the database supports trigger creation but Adeptia may not have added support for it. In such cases, you need to create the Database event using the SQL Query option.

	For details on databases supported by Adeptia, refer to the SQL Trigger section.
---	--

- Process record that matches the SQL 'Where' condition (Condition on column value)

If you want to process an individual record when a certain column value is matching some criterion, then using SQL query is the only option. You cannot use SQL Trigger in this case, as it just notes that a record has been added, updated or deleted, but does not check its value.

For example:

- When a new order is added into the **Order** table, the business process shall process that order, generate invoice and send it. The SQL Query for adding the order is defined as:

```
Select * from Order where order_status="new"
```

- When a new order is added in the Order table, you want to process this order only if its value is higher than 500. You will define the SQL Query for this as:

```
Select * from Order where order_status="new" and order_amount > 500
```

- When you want to synchronize the processing of a record, such as inserting a new record into the table and processing them based on some business logic that will update the status of the record in the table, you define the SQL Query as:

```
Select * from synchronize where status= "ReadyToProcess"
```

- Trigger process if the value from Aggregate () function in SQL Query satisfy the condition in event definition.

Examples of such a scenario are outlined below:

- You want to trigger a process flow whenever the count of people who have not paid their premium goes beyond 50. For this, you define the SQL Query as:

If (Select Count (customer) where premium_status= "notPaid") > 50

- You want to trigger a process flow if the average Inventory level goes below the reorder level of 100. For this, you define the SQL Query as:

If (Select Avg (Quantity) from Inventory) < 100

How to use SQL Query

Creating a SQL Query is a four step process:

1. Defining the SQL Query
2. Defining the Polling Frequency
3. Triggering of Event
4. Passing the information to the Process Flow

Using a SQL Query varies based on the task to be performed. These tasks are outlined as:

- Process Record that matches the SQL 'Where' condition (Condition on column value)
- Trigger process if the value from Aggregate () function in SQL query satisfies the condition in event definition.

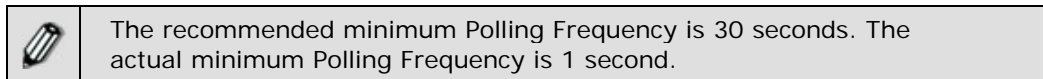
Process Record that matches the SQL where condition (Condition on column value)

Defining the SQL Query

1. You need to select the *SQL Query* radio button. This activates the *SQL Query* field.
2. Define a valid SQL Query that has 'Where' clause on one or more columns in the table.

Defining the Poling Frequency

You need to define a polling frequency. This is the frequency, on the basis of which the data in the table gets updated. If you think data is getting updated (on an average) every minute, then you can define the polling frequency as 1 minute. This implies that the select query will get executed every minute.



Triggering of Event

1. On activating the Database Event, the event will execute the select query after each polling time interval.

2. If that query returns one or more records, the process flow bound to the Database Event will get triggered one or more times respectively, one time for each row, returned by the query. In case no row is returned, then the system will log the information, but will not trigger a process flow.


Passing Information to the Process Flow

Pre-requisite

- The process flow registered to the Database Event must contain Adeptia Database Schema and Database Source activities.
- The *EventContextEnabled* property of the Adeptia Database Source must be set to *True*.

The passing of information to the Process Flow involves the following steps:

1. When the Database event is triggered, the record for which the process flow is fired is set in the process flow context with the context variable ROW. The complete record is stored in the form of name and its value (Name (Value)), and it is set in this variable ROW.
2. The values set in the context variable ROW are passed to the Database Source of the process flow.
3. The Database Source does not execute the SQL Query defined in it. The record set in the context variable ROW gets assigned to the output result set of the SQL Query specified in the Database Source.
4. This result set is converted into an XML record. This is generated as output by the Database Source activity and is sent to the process flow for further processing.

	<p>You can also access the ROW context variable from the context variable option instead of the Database source in the process flow.</p> <p>For details on other values passed by the Database event, refer to Appendix 1.</p>
---	--

Example

Case

You have a table named **Order** in the database. When an order is placed, a new record gets added in this table. When a new record is inserted in the table, the *order_status* field is populated with the value of "new".

Your company has a business policy to process those orders which have the order amount higher than 10000. You want to automate the processing of the orders. Additionally, you want that whenever a new order is placed, its processing should start within one minute.

The database, in which the **Order** table exists, does not support SQL trigger.

Solution

In such a case, you need to create a Database event using a SQL query that will trigger a process flow, each time an order higher than an amount of 10000 is placed. The event will then pass this order to the process flow for processing.

Defining Database Event

For this, you define the SQL query by performing the following steps:

1. Enter a select query (*Select * from Order where order_status='new' and order_amount>10000*) in the *SQL Query* field.
2. Mark the *Check Condition* checkbox as unchecked.
3. Select the polling frequency as 1 minute, from the *Polling Frequency* drop-down list.

Defining Process flow

Define a process flow for processing the record. Please ensure that the Database Source activity in the process flow should be created with the *EventContextEnabled* property selected as *True*.

Registering Process Flow with Database Event

Register the process flow with its corresponding Database Event using the Adeptia Event Registry.

Activating the Database Event

On activating the Database event, the select query is executed and the value of the first object obtained from the result is compared with the conditional value as per the relational operator selected. Hence the process flow gets triggered only once. If no row is returned, then the system logs the information but does not trigger a process flow.

Trigger the process if the value from Aggregate () function in SQL Query satisfies the condition in the Event definition.

Defining the SQL Query

1. You need to select the *SQL Query* radio button. This activates the *SQL Query* field.
2. Define a valid SQL Query that uses the **Aggregate ()** function which always returns one value.
3. Mark the *Check Condition* checkbox as checked.
4. Define the conditional value.

Defining the Poling Frequency

You need to define a polling frequency. This is the frequency, on the basis of which the data in the table gets updated. If you think data is getting updated (on an average) every minute, then you can define the polling frequency as 1 minute. This implies that the select query will get executed every minute.

Triggering of Event

1. On activating the Database Event, the event will execute the select query after each polling time interval.
2. If that query returns a value that matches the conditional value specified in the Database Event activity, then the process flow gets triggered. Else, an error is logged and the process flow is not triggered.

Passing the Information to the Process Flow

In this case, no data is passed from the event to the process flow. It works like a plain trigger.

Example

Case

You have a table named **CustomerPolicy** in the database. This table maintains all policy information and also records information such as the status whether the customer has paid premium amount or not.

Your company monitors the number of customers who have not paid their premium policy, at regular intervals. Each time this number exceeds 100, you need to notify the senior management and send all details of the customers. You need to do this, within one hour of this number reaching 100.

The **CustomerPolicy** table exists in MSAccess database.

Solution

In such a case, you need to create a Database event using a SQL query that will trigger a process flow; each time the number of customers who have not paid their premium amount exceeds 100. The process flow will send an email to the senior management with the customer details.

Defining Database Event

For this, you define the SQL query by performing the following steps:

1. Enter a select query (*Select Count (CustomerPolicy) where Premium_Status="notPaid"*) in the *SQL Query* field. This query returns the total number of customers who have not paid their premium amount.
2. Mark the *Check Condition* checkbox as checked.
3. Select the Relational operator as Greater Than from the *Operator* drop-down list.
4. Enter the conditional value as 100 in the *Value* field.
5. Select the polling frequency as 60 minutes, from the *Polling Frequency* drop-down list.

Defining Process flow

Define a process flow for processing the record. Please ensure that the Database Source activity in the process flow should be created with the *EventContextEnabled* property selected as *True*.

Registering Process Flow with Database Event

Register the process flow with its corresponding Database Event using the Adeptia Event Registry.

Activating the Database Event

On activating the Database event, the select query is executed and the value of the first object obtained from the result is compared with the conditional value as per the relational operator selected. Hence, the process flow gets triggered only once. If no row is returned, then the system logs the information but does not trigger a process flow.

Advantages

- It is easy to use as it is simple to write and implement.
- It is supported by all databases.
- It can be used to trigger the same process flow multiple times, depending upon the number of rows returned by the query.

Limitations

- When you use the SQL Query to match a conditional value, then it checks only the first value returned by the query. For example, in the SQL query where:

Select EmpID from EMP ; which returns more than one row
Check Condition checkbox is marked as *checked*
Relational Operator is selected as *Equal To*
Conditional value is entered as *10*

When the event is activated, then only the first value returned by the query is matched with the conditional value. If it matches, then the process flow gets triggered otherwise not.

Using SQL Trigger

A SQL Trigger is a set of SQL statements that get executed when data in a table is changed due to any Insert/Update/Delete operation.

If the database supports SQL Trigger creation, then it is recommended to use the SQL Trigger method to trigger a process flow whenever any DML (Insert/Update/Delete) operation is performed on a database table.

Many databases support SQL Trigger creation, but Adeptia supports SQL Trigger creation for the listed databases:

- Oracle
- SQL Server
- Sybase



In the current release, Adeptia supports these databases. In future releases, it will support all databases that support SQL Trigger creation.

When to use SQL Trigger

There are certain scenarios in which you can use only the SQL Trigger method to define the database event. These are outlined as:

- DML Operation in a Database: You need to trigger a process flow whenever a record is inserted, updated or deleted in a table and process that record in the process flow. For example, you want to process an order whenever a new record is added into the *Order* table. But the database does not have any column like status or timestamp that enables you to identify if the record is new.
- Process a record that matches the SQL 'Where' condition (condition on column value)
- Trigger a process if the value from Aggregate () function in the SQL query satisfies the condition in event definition

How to use SQL Trigger

Using the SQL Trigger includes the following steps:

1. Defining the SQL Trigger
 1. Triggering the Database Event
 2. Passing the information to the Process Flow

Defining the SQL Trigger

This step is performed by the developer.

To define the SQL Trigger:

1. Select the **SQL Trigger** option.

A sample Trigger code is seen in the *SQL Trigger* field. The words present in angle bracket (i.e., <WHERE CLAUSE>) of the Insert statement can only be replaced by a correct WHERE clause and rest of the Insert Statement should be as it is. In this scenario, we consider an EMP table, and we assume that a NAME column in it is unique or primary Key.

This *Name* field is taken to identify the row for which the trigger is fired. This information is passed to process flow to make sure that process flow process the same record.

A sample trigger for above-mentioned scenario is displayed below.

Database Name	Task	Trigger Definition
Oracle	Add a new row in the trigger database table when a record is inserted, updated or deleted in the emp table. When a new row is added into the dbeventtrigger table, a process flow is triggered.	CREATE OR REPLACE TRIGGER trig After insert or update or delete on emp for each row begin INSERT INTO dbeventtriggertable VALUES ('Query = WHERE rowid =''' :new.rowid ''''); END trig ;

Sql Server		<pre> create trigger trigtest on emp for insert,update,delete as declare @empname varchar begin set @empname=(select empname from inserted); INSERT INTO dbeventtriggertable VALUES ('Query =WHERE empname=" +@empname+"); END ; </pre>
Sybase		<pre> CREATE TRIGGER mytrig on EMP for insert, update as declare @name varchar begin set @name = (select NAME from inserted) INSERT INTO dbeventtriggertable VALUES ('Query = WHERE NAME="" @name "") END </pre>

When the trigger is fired the **Where** clause will be updated to **Where NAME='Smith'** where Smith is the inserted value in the NAME field of the EMP table.

The syntax of the trigger varies for each database. It is outlined in the table below.

Database Name	Trigger Definition	Reference
Oracle	<pre> CREATE [OR REPLACE] TRIGGER <trigger_name> { BEFORE AFTER} { INSERT DELETE UPDATE} ON <table_name> [REFERENCING [NEW AS <new_row_name>] [OLD AS <old_row_name>]] [FOR EACH ROW [WHEN (<trigger_condition>)] <trigger_body> End <<trigger_name>; </pre>	http://infolab.stanford.edu/~ullman/fcdb/oracle/or-triggers.html#basic%20trigger%20syntax

SQL Server	<pre>CREATE TRIGGER trigger_name ON {table view} [WITH ENCRYPTION] { { { FOR AFTER INSTEAD OF } { [INSERT] [,] [UPDATE] [,] [DELETE] } [WITH APPEND] [NOT FOR REPLICATION] AS [{ IF UPDATE (column) [{ AND OR } UPDATE (column)] [...n] [IF (COLUMNS_UPDATED () { bitwise_operator } updated_bitmask) { comparison_operator } column_bitmask [...n] }] sql_statement [...n] } } END</pre>	http://www.devbuilder.org/asp/dev_article.asp?aspid=16
Sybase	<pre>CREATE TRIGGER [owner.] trigger_name ON [owner.]table_name { FOR {INSERT, UPDATE, DELETE} AS SQL_statements FOR {INSERT, UPDATE} AS IF UPDATE (column_name) [{AND OR} UPDATE (column_name)]... SQL_statements }</pre>	http://manuals.sybase.com/onlinebooks/group-asp/asg1250e/sqlug/@Generic__BookTextView/48018;hf=0

Triggering of Event

Once the developer defines the SQL trigger, he executes it to trigger the Database event. A Process flow can be bound to the Database event using the Event Registry activity. The above Database event is triggered for each row inserted or updated in the EMP table and this executes the associated Process Flow.


Passing the Information to the Process Flow

Once the Database event is fired, the required information is passed to the process flow. This is done in the backend. The <where clause > which keeps the track of the NAME field (of EMP table) of the row inserted or updated is passed to the Process Flow context. For example in this case **Where NAME='Smith'** is passed to the Process flow context.

Pre-requisites

- The process flow registered to the Database event should contain an Adeptia Database Source activity.
- The *EventContextEnabled* property of the Adeptia Database Source should be set to true. If the **EventContextEnabled** property is true then **where clause** of the Database Source gets overridden by the where clause of the Database Event set in the process flow context.

In the current scenario the Where clause of the Database source gets overridden by **Where NAME='Smith'** and this makes sure that the database source picks up the same record for which the event was triggered.

	If the database event is deactivated, then the system deletes the polling frequency value and the complete trigger.
---	---

Examples

Case 1

You have a table named **EMP** in the Oracle database in your company. When a new employee joins your company, a new record gets added in this table. When a new record is inserted in the table, you have to initiate a few transactions.

Solution


In such a case, you need to create a Database event using a SQL Trigger that will trigger a process flow, each time a new employee record is inserted in the **EMP** table. The event will then pass this Employee record to the process flow for processing.

Defining Database event

For this, you define the SQL Trigger by performing the following steps:

1. Enter the SQL Trigger as:

```
CREATE OR REPLACE TRIGGER trig After insert or update on emp
for each row
begin
  INSERT INTO dbeventtriggertable VALUES ( 'Query = WHERE rowid
= ''||:new.rowid ||'' ');
END trig ;
```

	<p>The <i>rowid</i> field is used to identify the row, for which the trigger is fired. The information (<i>Where rowid=<rowid of the inserted record></i>) is passed to the process flow to make sure that process flow processes the same record. This information overrides the Where clause of the Adeptia Database Source activity, if its <i>EventContextenabled</i> property is selected as True.</p> <p>If the database does not support <i>rowid</i>, then you can define any other column as the primary key.</p>
---	---

3. Select the polling frequency as 1minute, from the *Polling Frequency* drop-down list.

Defining Process flow

Define a process flow for processing the record. Please ensure that the Database Source activity in the process flow should be created with the *EventContextEnabled* property selected as *True*.

Registering Process flow with Database Event

Register the process flow with its corresponding Database Event using the Adeptia Event Registry.

Activating the Database Event

1. If the trigger syntax is valid then the trigger is created in the backend database, else it throws an exception and the Database event is not activated. Once the Database event is activated the trigger is created in the database.
2. If a new record is inserted in the table, the rowid of the inserted record is added in Adeptia's temporary table called *dbeventtriggertable*.
3. The Database event is triggered as per the polling frequency specified in the Database event activity. It triggers the bound process flow, if an entry corresponding to the Database event is found in the *dbeventtriggertable*.
4. Different process flows are triggered for each entry in the *dbeventtriggertable* and the Where clause (*Where rowid=<rowid of the inserted record>*) of the INSERT statement of the trigger body is set in the context.
5. This information set in the context overrides the Where clause of the Database Source activity, if its *EventContextEnabled* property is selected as True.

Case 2

You have a table named **Order** in the Oracle database in your company. When an order is placed, a new record gets added in this table. When a new record is inserted in the table, the *order_status* field is populated with the value of "new".

Your company has a business policy to process those orders which have the order amount higher than 10000. You want to automate the processing of the orders. Additionally, you want that whenever a new order is placed, its processing should start within one minute.

Solution

In such as case, you need to create a Database event using a SQL Trigger that will trigger a process flow, each time an order higher than an amount of 10000 is placed. The event will then pass this order to the process flow for processing.

Defining Database event

For this, you define the SQL Trigger by performing the following steps:

1. Enter the SQL Trigger as:

```
CREATE OR REPLACE TRIGGER MYTRIG AFTER INSERT
ON ORDERTABLE FOR EACH ROW
BEGIN
```

```


IF (:NEW. ORDER_STATUS='NEW' AND :NEW.AMOUNT>10000) THEN

    INSERT INTO DBEVENTTRIGGERTABLE VALUES ('QUERY = WHERE
ROWID='||:NEW.ROWID);

    END IF;

END MYTRIG;

```

	<p>The <i>rowid</i> field is used to identify the row, for which the trigger is fired. The information (<i>Where rowid=<rowid of the inserted record></i>) is passed to the process flow to make sure that process flow processes the same record. This information overrides the Where clause of the Adeptia Database Source activity, if its <i>EventContextEnabled</i> property is selected as True.</p> <p>If the database does not support <i>rowid</i>, then you can define any other column as the primary key.</p>
---	---

4. Select the polling frequency as 1minute, from the *Polling Frequency* drop-down list.

Defining Process flow

Define a process flow for processing the record. Please ensure that the Database Source activity in the process flow should be created with the *EventContextEnabled* property selected as *True*.

Registering Process flow with Database Event

Register the process flow with its corresponding Database Event using the Adeptia Event Registry.

Activating the Database Event

1. If the trigger syntax is valid then the trigger is created in the backend database, else it throws an exception and the Database event is not activated. Once the Database event is activated the trigger is created in the database.
2. If a new record is inserted in the table with *order_status="new"* and amount >10000, the *rowid* of the inserted record is added in Adeptia's temporary table called *dbeventtriggertable*.
3. The Database event is triggered as per the polling frequency specified in the Database Event activity. It triggers the bound process flow, if an entry corresponding to the Database Event is found in the *dbeventtriggertable*.
4. Different process flows are triggered for each entry in the *dbeventtriggertable* and the Where Clause (*Where rowid=<rowid of the inserted record>*) of the INSERT statement of the trigger body is set in the context.
5. This information set in the context overrides the Where clause of the Database source activity if its *EventContextEnabled* property is selected as True.

Advantages

- SQL Trigger enables you to keep a track of the data inserted, update or deleted from a table.

Limitations

- SQL Trigger creation is supported by limited databases only. These databases are outlined as:
 - Oracle
 - SQL Server
 - Sybase

Problem Scenarios where both SQL Query or SQL Trigger serve as solutions

In some problem scenarios, both the SQL Query or SQL Trigger methods can be used as solutions. A few such cases are described below.

- Triggering the process per record bases and process that record in process flow

In such a case, if the database and Adeptia support SQL Trigger creation, then it is recommended to use the SQL Trigger option.

Appendix 1

The Variables passed by the Event to the process flow context are listed below.

Context Variable Name	Context Value
EventId	Event Activity Id
EventName	Event Activity Name

USING XSL TEMPLATE FOR PADDING IN MAPPING

Objective

- To map fields from source schema to target schema where the length of the field of the target schema is fixed. If the field length of the source schema is lesser, then you need to pad the field using a padding character. You can define an XSL template for the padding.

Tasks

1. Creating XSL Template
2. Using XSL Template

Creating XSL Template

To create an XSL template for padding a field:

1. Enter the name of the template in the *Name* field.
2. Define four parameters for the padding in the XSL template. These are outlined in the table below.

Parameter	Description
Field Name to be padded	Field name of source schema that is to be padded
Padding Character	Character used to pad the field. This can be a < 'space character'>, '\$', '?', '&' or any other character.
Target length of the field	The target length that is to be achieved by the padding. This is the fixed length of the field of the target schema.
Padding Side	Direction in which the field is to be padded. It can be padded either in the left or right direction. However, you cannot pad a field in both directions at one time.

3. Assign values to these parameters and define the XSL code for the padding. Enter the XSL code shown below in the *XSL Template* field.

```
<xsl:template name="paddingTemplate">
  <xsl:param name="sourceParam" />
  <xsl:param name="padChar" />
  <xsl:param name="length" />
  <xsl:param name="paddingSide" />
```

```

<xsl:choose>
  <xsl:when test="$paddingSide = 'left'">
    <xsl:choose>
      <xsl:when test="string-length($sourceParam) < $length">
        <xsl:call-template name="paddingTemplate">
          <xsl:with-param name="sourceParam"
            select="concat($padChar,$sourceParam)"/>
          <xsl:with-param name="padChar" select="$padChar"/>
          <xsl:with-param name="length" select="$length"/>
          <xsl:with-param name="paddingSide"
            select="$paddingSide"/>
        </xsl:call-template>
      </xsl:when>
      <xsl:otherwise>
        <xsl:value-of select="substring($sourceParam,string-
          length($sourceParam) -$length + 1)"/>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:when>
  <xsl:otherwise>
    <xsl:if test="$paddingSide = 'right'">
      <xsl:choose>
        <xsl:when test="string-length($sourceParam) < $length">
          <xsl:call-template name="paddingTemplate">
            <xsl:with-param name="sourceParam"
              select="concat($sourceParam,$padChar)"/>
            <xsl:with-param name="padChar" select="$padChar"/>
            <xsl:with-param name="length"
              select="$length"/>
            <xsl:with-param name="paddingSide"
              select="$paddingSide"/>
          </xsl:call-template>
        </xsl:when>
        <xsl:otherwise>
          <xsl:value-of select="substring($sourceParam,string-
            length($sourceParam) -$length + 1)"/>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:if>
  </xsl:otherwise>
</xsl:choose>
</xsl:template>

```

Figure: XSL Code

4. Save the template.



For details on creating an XSL Template, refer to **Using XSL Template** section in the User Guide.

Using XSL Template

Once you have created the XSL template, you need to use it to map the fields from the source schema to the target schema. The newly created template appears in the Parameters Panel.

To use the XSL template to map the fields:

1. Click the field that is to be mapped in the target schema. It appears in the Mapping Graph Area.
2. Double-click the field to be mapped in the source schema. It appears in the Mapping Graph Area.
3. Define three constants for creating the padding character, target length and padding side parameters.
4. Click the newly created XSL template for the padding. It appears in the Mapping Graph Area.
5. Map the source field and the constants to the XSL template and thereon to the target field, by dragging the fields and clicking the **Apply Mapping** button.

Example

Case

You need to map the *Name* and *Salary* fields from Text schema (source) to Positional schema (target). The length of the *Name* field in the Text schema is 4 characters, whereas it is 25 characters in the Positional schema. Similarly, the field length of *Salary* is 6 characters in the Text schema, whereas it is 10 characters in the Positional schema. Since the target schema is a Positional schema, the field length is fixed, and for mapping it requires the field in the source schema to be of the same length.

Solution

You need to pad the *Name* and *Salary* fields in the Text schema to increase their length to the desired length in the Positional schema. You can pad it with the space character. You can pad the *Name* field in the left direction and the *Salary* field in the right direction.

Creating XSL Template

1. Enter the details in the Manage XSL Template screen as shown in the figure below.

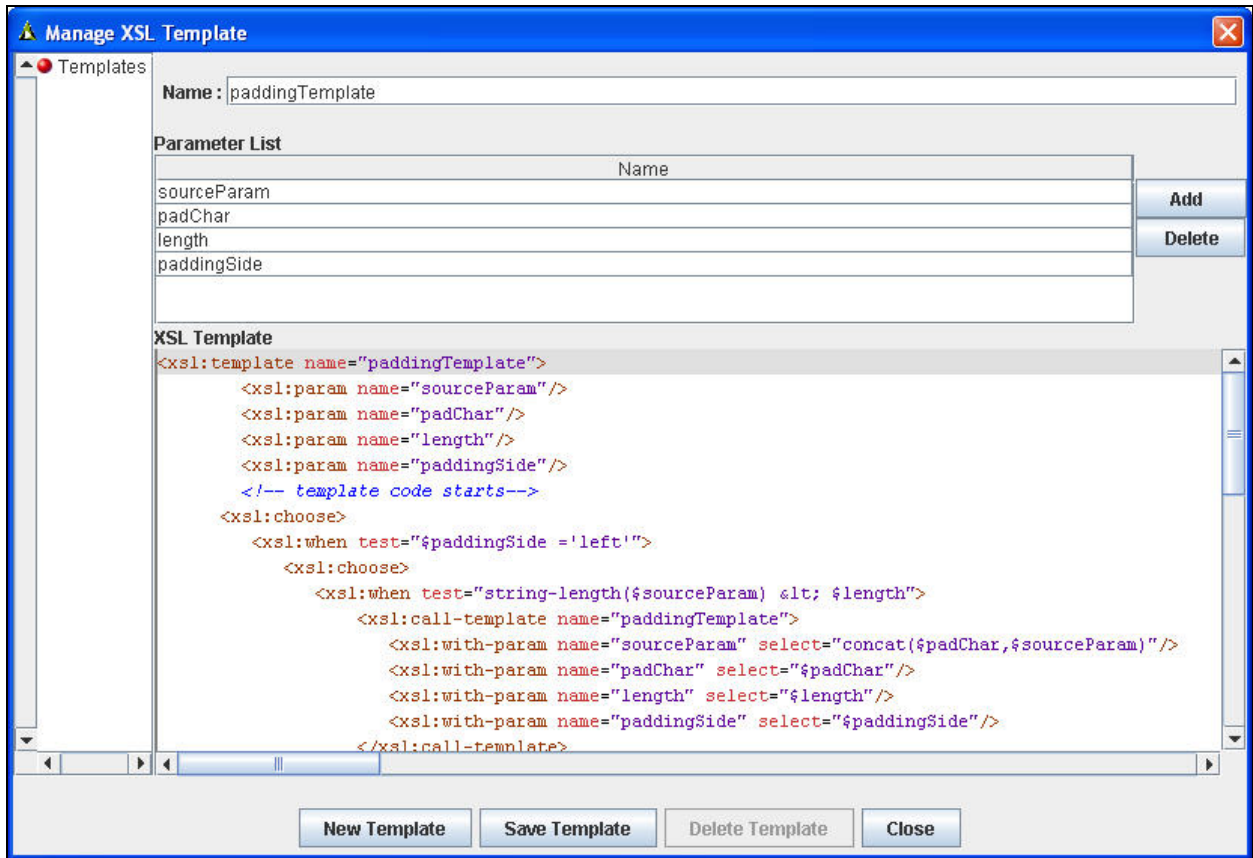


Figure: Creating XSL Template

2. Click **Save Template** and then **Close** to save the template and return to the Data Mapper screen.

Using XSL Template

1. Click the *Name* field in the Target panel. It appears in the Mapping Graph Area.
2. Double-click the *Name* field in the Source panel. It appears in the Mapping Graph Area.
3. Define three constants for creating the parameters as:

Constant	Data Type
'< >'	String
25	Number
Left	String

4. Click *paddingTemplate* from the Parameters Panel. It appears in the Mapping Graph Area.
5. Map the *Name* field (source) and the constants to the XSL template and thereon to the *Name* field (target) by dragging the fields and clicking the **Apply Mapping** button (see Figure below).

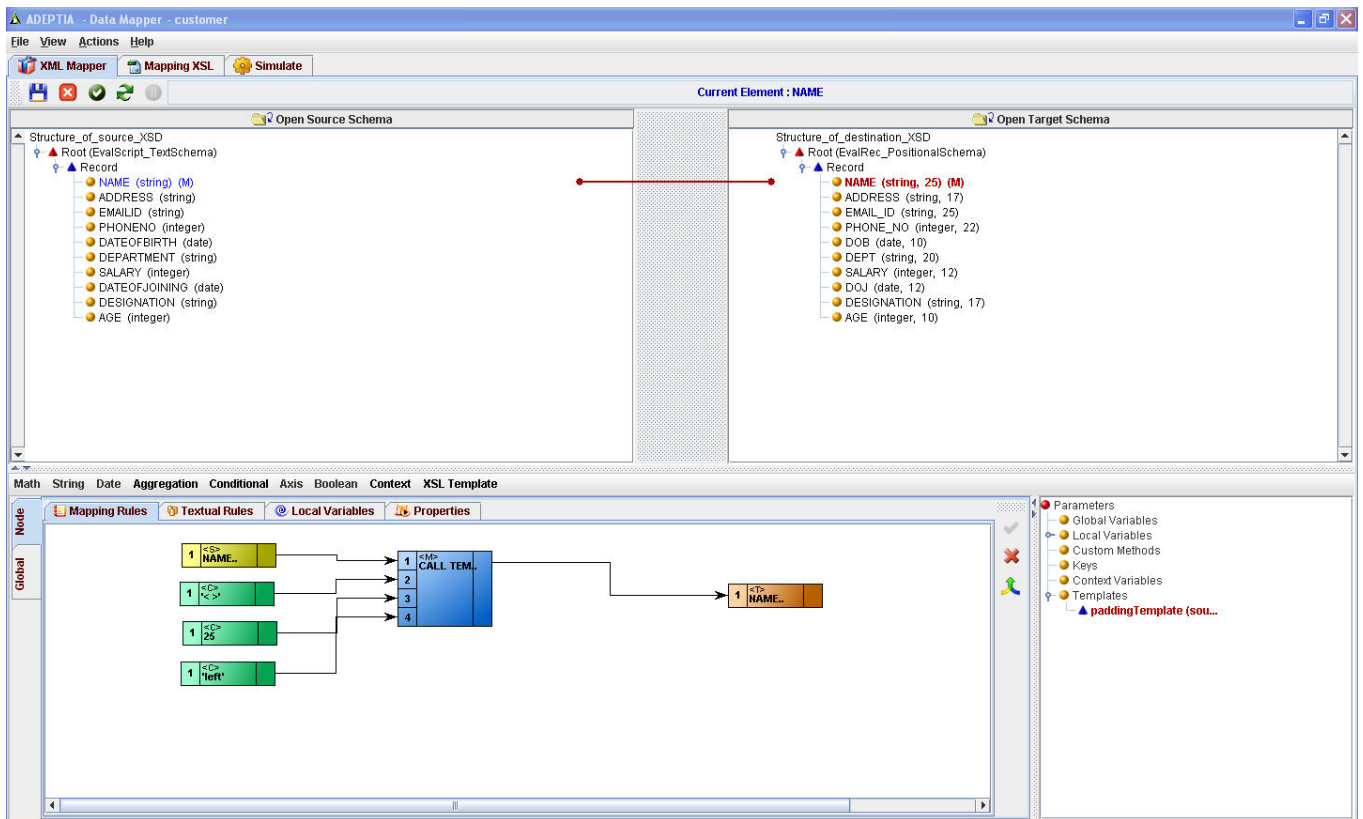



Figure: Using XSL Template

- Similarly, pad the *Salary* field. The constants will be defined as outlined in the table below.

Constant	Data Type
'< >'	String
10	Number
right	String

 You can test the mapping using **Simulate** tab.