How to Perform External DB Look-ups Utilizing J2EE DBPools

1. Overview

This document explains how to access an external database with the purpose of performing look-ups from within an XI mapping, via the SAP J2EE database connection pool feature.

2. Introduction

Within an XI mapping it is a common requirement to be able to perform data lookups on-the-fly against external data sources. In particular, there may be a need to lookup some data which is maintained in an external database. This problem can be resolved in one of two ways:

1) Import the data from the external system, that is, replicate the data in XI and perform a value-based-transformation in the mapping step of XI. This technique is useful when the data does not tend to change much over time.
2) From within the mapping step of XI, execute a call to an external database with the purpose of accessing the desired data. This technique is particularly useful when accessing data that changes regularly and replication is not a viable option.

The focus of this document is on option 2). The “value mappings” offered by XI are not adequate in this case, since the data would have to be manually entered in the Integration Directory.

3. User Exits

Since XI 2.0 does not provide out-of-the-box functionality for performing such look-ups, this functionality has to be implemented manually in the form of an exit in the mapping process.

Both graphical mappings (“Message mapping” objects) and XSLT offer the concept of user exits. Since the XI mapping runtime is Java-based, the user exits must be written in Java.

4. Accessing data from an external relational database.

Even though it is possible to access an external database from within the mapping step of XI utilizing regular JDBC constructs, this approach for performance reasons, generally speaking, is not recommended. The following steps take place when utilizing JDBC:

1. Load the JDBC driver.
2. Define the connection URL.
3. Establish the connection.
4. Create a statement object.
5. Execute a query or update.
6. Process the results.
7. Close the connection.

Keep in mind that each message being sent through XI would represent a mapping instance.

A more efficient solution would be the utilization of J2EE database connection pools. These pools are created in a declarative fashion utilizing the Administrative tools of the J2EE engine. With this solution, the overhead resulting from the creation of a database connection for each mapping instance disappears since these connections are acquired by the J2EE engine and shared by multiple processes during runtime. This solution, as well as the former, utilizes JDBC. What is different is how the JDBC connections are utilized.

5. Java libraries

5.1. JDBC Libraries

Since the DBPool connection utilizes JDBC, it is necessary to acquire the JDBC classes from the database vendor of the external database being accessed. To check an up-to-date list of JDBC drivers please visit: http://servlet.java.sun.com/products/jdbc/drivers.

Other than the JDBC libraries, nothing else is needed except of course the java code that utilizes the DBPools.

5.2. Referencing the Java code used in the lookup from the mapping environment.

The Java code that executes the lookup, could be referenced from within a user-defined function in the graphical mapper or from within XSLT.

The archives associated with the DB look-up code may be:

1) Loaded directly into a specific namespace/software-component in the IR via *the import archive* facilities of the Integration Builder. This technique allows for an easy way to make the code available to those user-defined functions within a namespace in XI. One disadvantage is that the lookup code would have to be loaded every time it needs to be utilized by a different namespace.
2) Declared at the J2EE engine level and then referenced from within the user-defined functions or XSLT program of XI. This technique allows various software components access to the code without having to re–import the code each time.

In the sections below, we will describe step-by-step how to reference the archives by utilizing option 1) above, and will cover at a high level the instructions on how to implement option 2).

6. Prerequisites

For the purpose of this exercise we are assuming the following software to be available and installed where indicated :

Your workstation :

- JDK 1.3.1 is installed on local PC
- JNDI and standard javasoft's JDBC additional libraries are installed in your PC. (place jdbc2_0-stdext.jar and jndi.jar in C:\jdk1.3*\lib\ext)
- Environment variables CLASSPATH and PATH have been adjusted to properly reference necessary libraries.

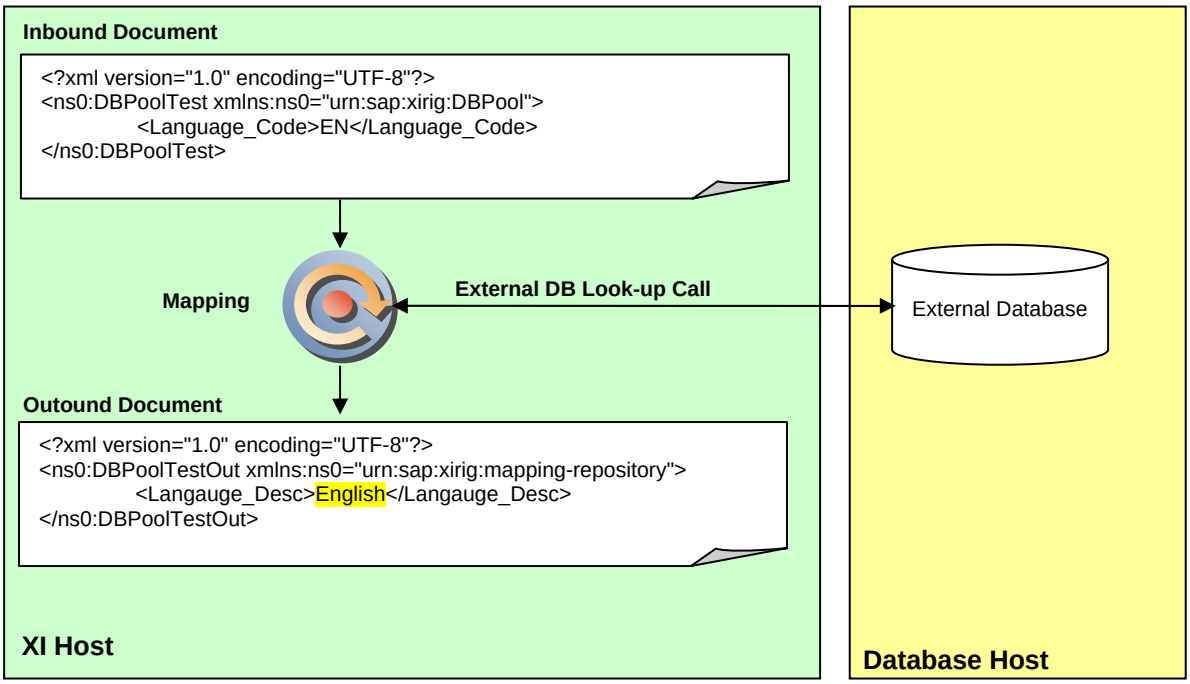
- JDBC classes of the external database should be placed in the following location:. <INSTALLATION>\usr\sap\<INSTANCE>\...\j2ee\cluster\server\services\dbpool\work.

  If not done, then in the J2EE Administrator, when adding a driver to the DBPool, the configuration proocess will request and add the jar files to the proper location.

7. Scenario

In this sample scenario we will implement a simple 'language' lookup which allows for the translation of a string representing an abbreviation of a language ('EN') into the fully-spelled meaning of the abbreviation ('English'). A database table contains this translation data and is located in an Oracle 9i external database.

The figure below illustrates the message flow and the lookup process.



In the next sections we will provide step-by-step guides for the following 3 procedures:

1. Call DB Pool lookup method from GUI mapping
2. Call DB Pool lookup method from XSLT.
3. Reference external Java libraries within the same SW component/namespace
4. Reference external Java libraries across all SW components/namespaces

8. Procedures

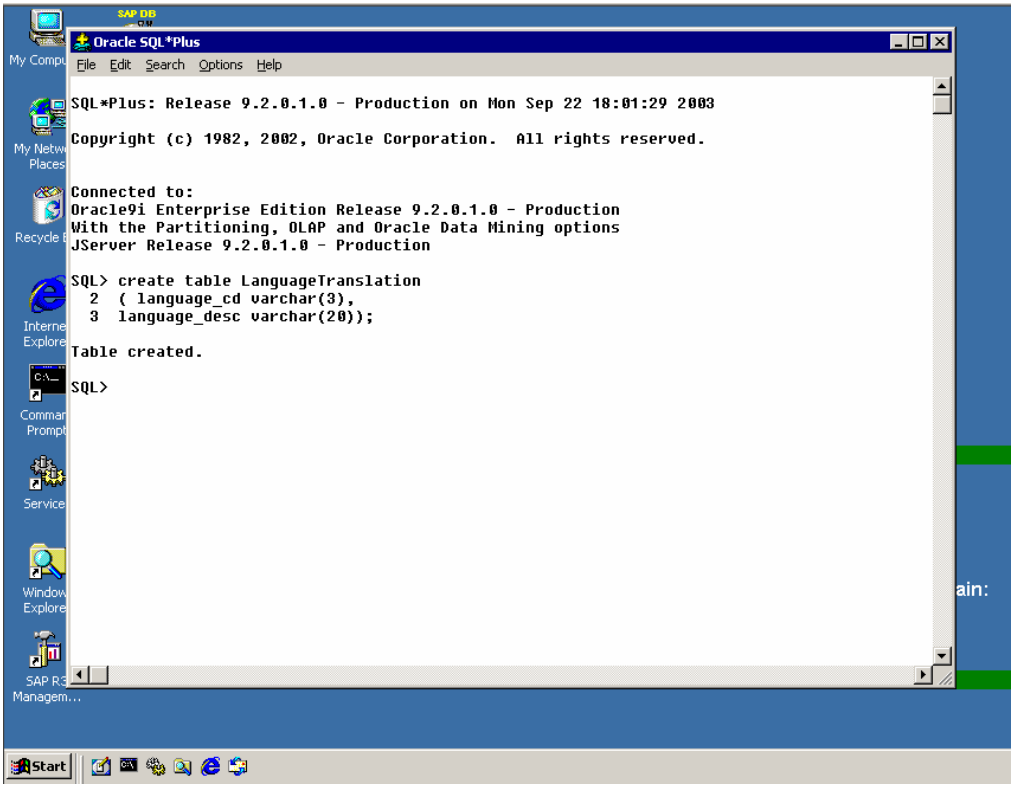
8.1. Call DB Pool lookup method from GUI mapping.

The steps are as follows:

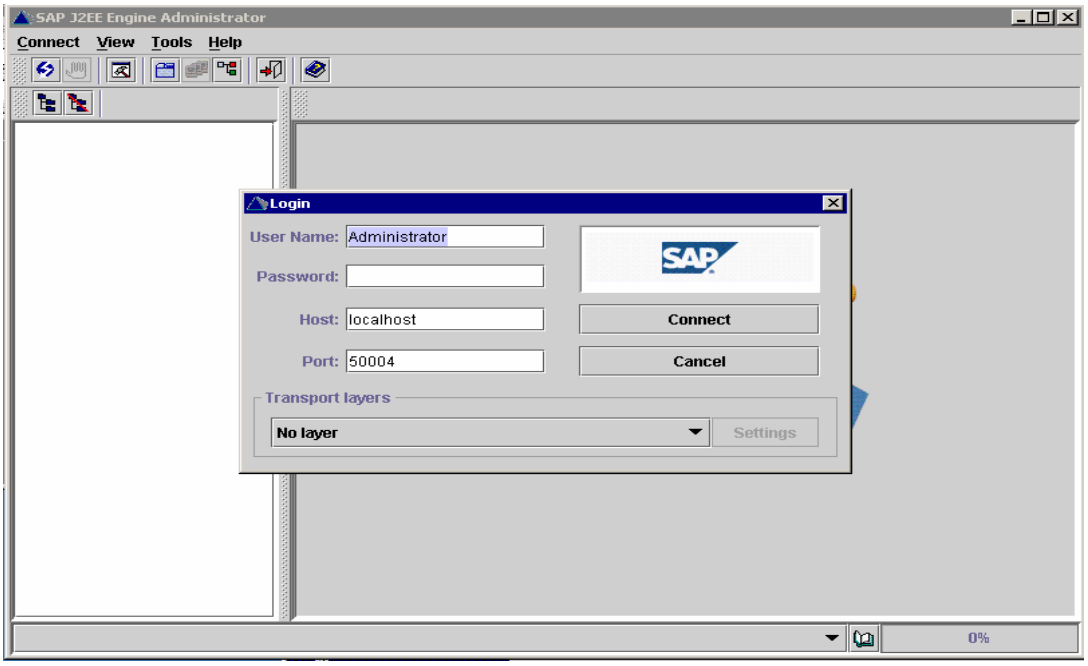
1. Declare database tables to use for this example.
2. Declare the database connection pool utilizing the J2EE Administrator tool.
3. Develop Java code that performs the database lookup leveraging the dbpool connection.
4. Develop a user-defined function in the Integration Repository utilizing the graphical mapping tool.
5. Test mapping.

8.1.1 Declare database tables to use for this example.

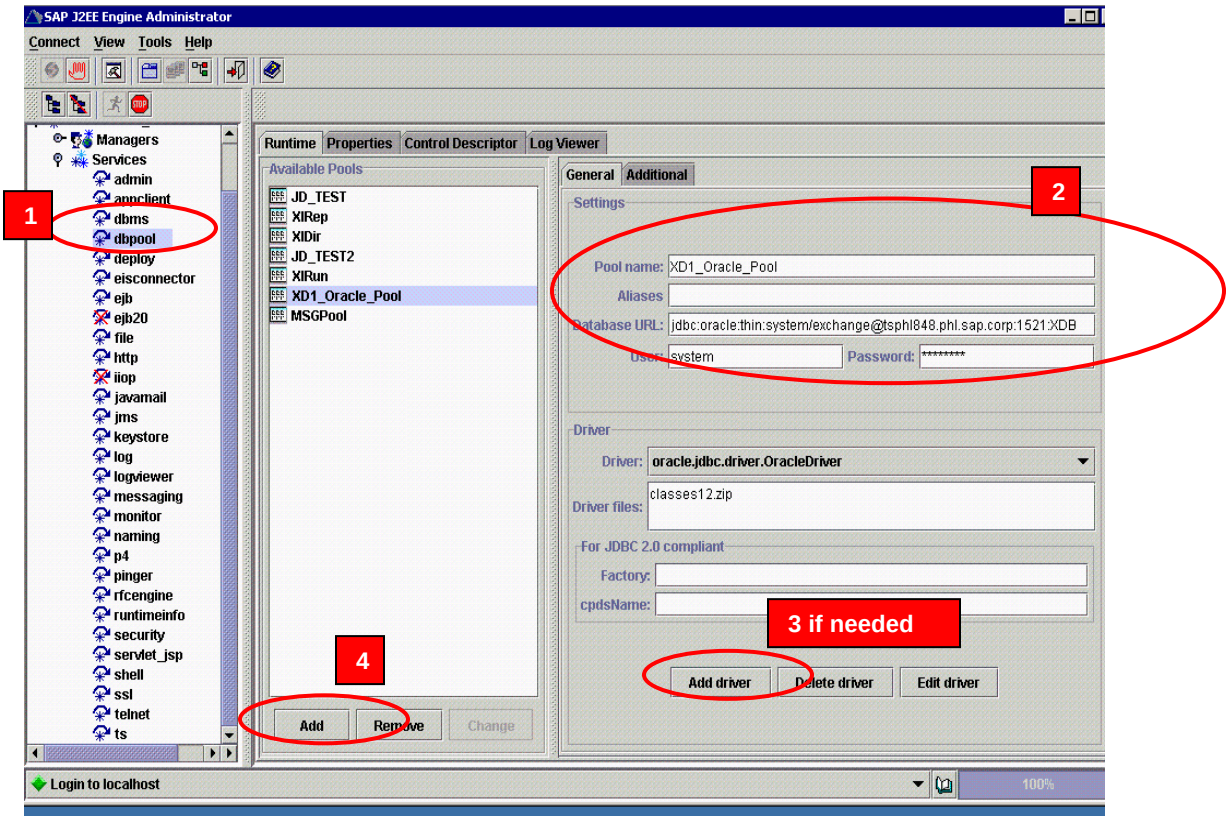
This step involves the creation of a database table or the identification of an existing one. This database can be located in the same application server or an external one. The following figure illustrates the table to be used in this example, in this case the table name is LanguageTranslation and it contains two simple VARCHAR columns called language_cd and language_desc.



8.1.2 Declare the database pool utilizing the J2EE Administrator tool.



Logon on to the J2EE Administration Tool



When you enter the first character in the field “Pool Name”, the Add button will appear. You may need to register the JDBC classes if the ones needed for your backend are not there.

8.1.3 Develop Java code that performs the database lookup leveraging the dbpool connection.

8.1.3 1 Code needed to access the DBPool declaration and which later will be imported into the repository.

The following java code allows for the utilization of a connection pool. This code may need to be enhanced to meet your functional needs as well as robustness level desired in areas such as error handling. This is a working template.

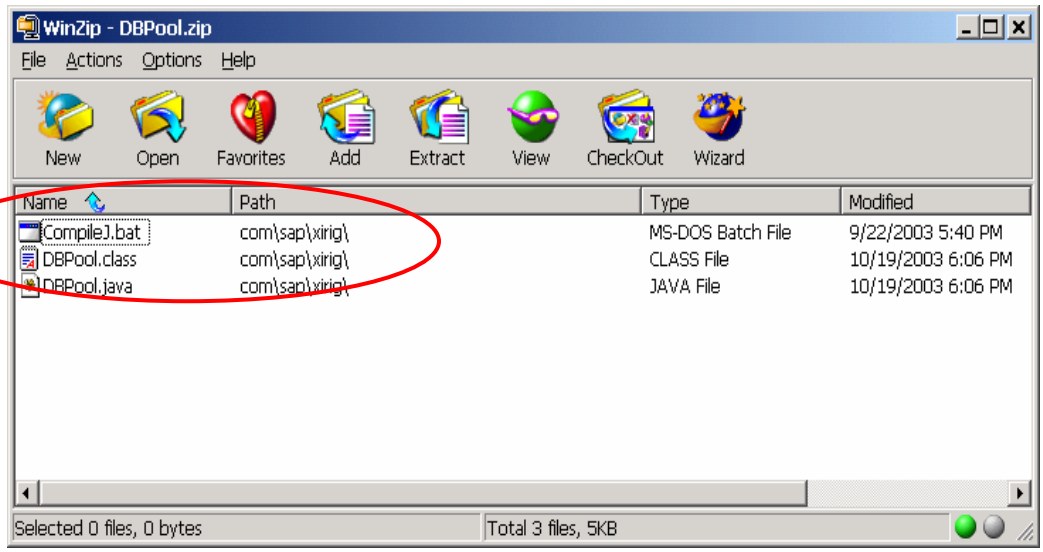
```
package com.sap.xirig;

import javax.naming.*;
import javax.sql.*;
import java.sql.*;

public class DBPool {

	String Conn_Status = "Not Connected";
	String Language_Desc = "Empty";
	String Language_Cd = "Empty";
	Connection conn;
	Context ctx;
	DataSource ds;


	/**
	 *  Constructor for the DBPool object
	 */
	public DBPool() {
		try {

			ctx = new InitialContext();
			if (ctx == null) {
				throw new Exception("Boom - No Context");
			}

			ds = (DataSource) ctx.lookup("jdbc/XD1_Oracle_Pool");
			if (ds == null) {
				throw new Exception("Boom - No dataSource");
			}
```

```
		}
		catch (Exception e) {
			e.printStackTrace();
		}
	}


	/**
	 *  Gets the LanguageDesc attribute of the DBPool object
	 *
	 *@param  v_str  Description of Parameter
	 *@return        The LanguageDesc value
	 */
	public String getLanguageDesc(String v_str) {
		Statement stmt = null;
		ResultSet rst = null;

		try {
			if (ds != null) {
				conn = ds.getConnection();
				Conn_Status = "Could not get connection to
datasource";
				if (conn != null) {
					Conn_Status = "Got Connection " +
conn.toString();
					stmt = conn.createStatement();
					rst = stmt.executeQuery("SELECT LANGUAGE_DESC
FROM LANGUAGETRANSLATION WHERE LANGUAGE_CD='" + v_str + "'");
					if (rst.next()) {
						Language_Desc = rst.getString(1);
					}
					conn.close();
				}

			}
		}
		catch (Exception e) {
			e.printStackTrace();
		}

		finally {
```

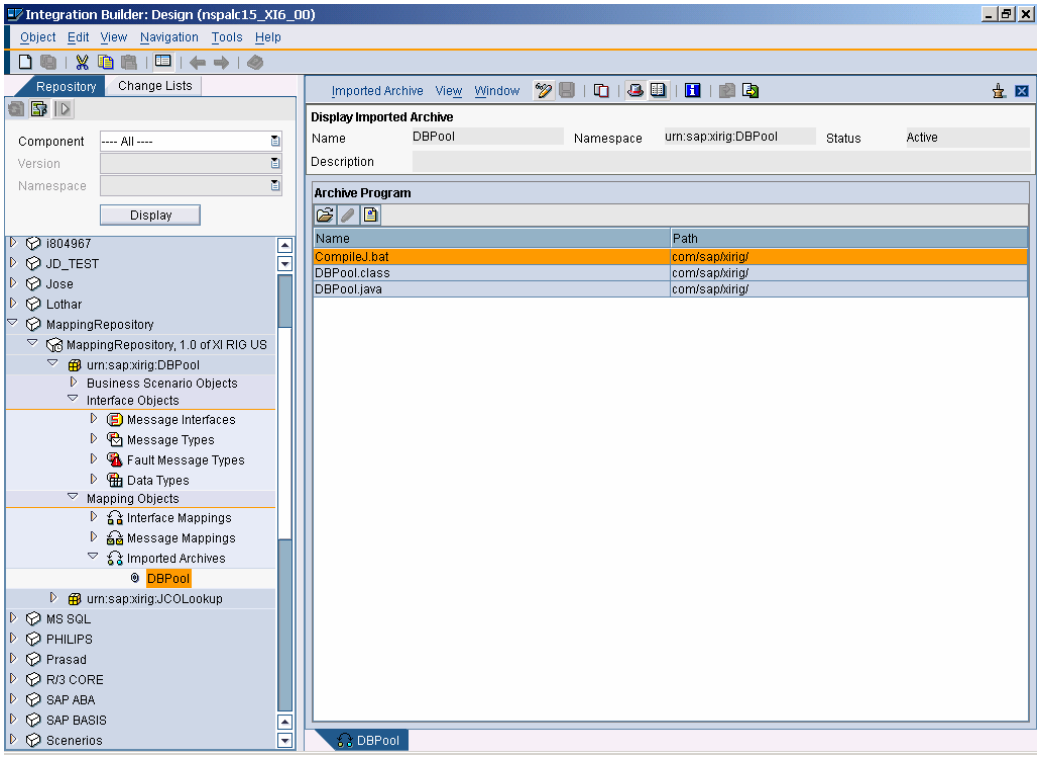
```
                        if (rst != null) {
                                try {
                                        rst.close();
                                }
                                catch (Exception e) {
                                        e.printStackTrace();
                                }
                        }

                        if (stmt != null) {
                                try {
                                        stmt.close();
                                }
                                catch (Exception e) {
                                        e.printStackTrace();
                                }
                        }
                        if (conn != null) {
                                try {
                                        conn.close();
                                }
                                catch (Exception e) {
                                        e.printStackTrace();
                                }
                        }

                }

                return Language_Desc;
        }

}
```

8.1.3.2 Compile the code with the following command, replacing C:\jdk1.3.1_09 with your own setting. In our case we named the above program **DBPool.java**:

```
javac -classpath C:\jdk1.3.1_09\lib\ext\jndi.jar;C:\jdk1.3.1_09\lib\ext\jdbc2_0-stdext.jar;.
DBPool.iava
```

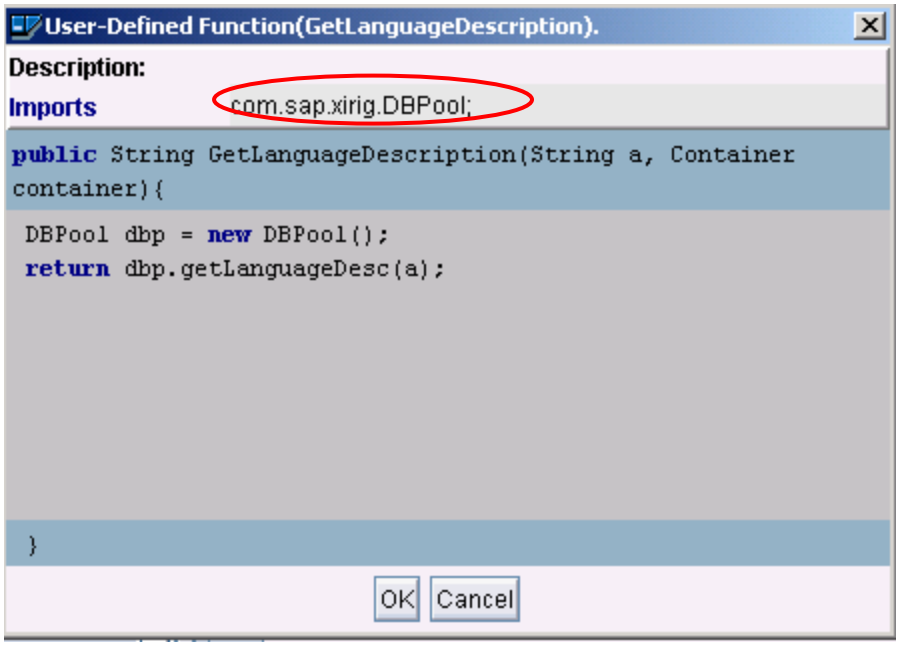


8.1.3.3 After the code is compiled, zip it up along with any other component you want to import into the Integration Repository.

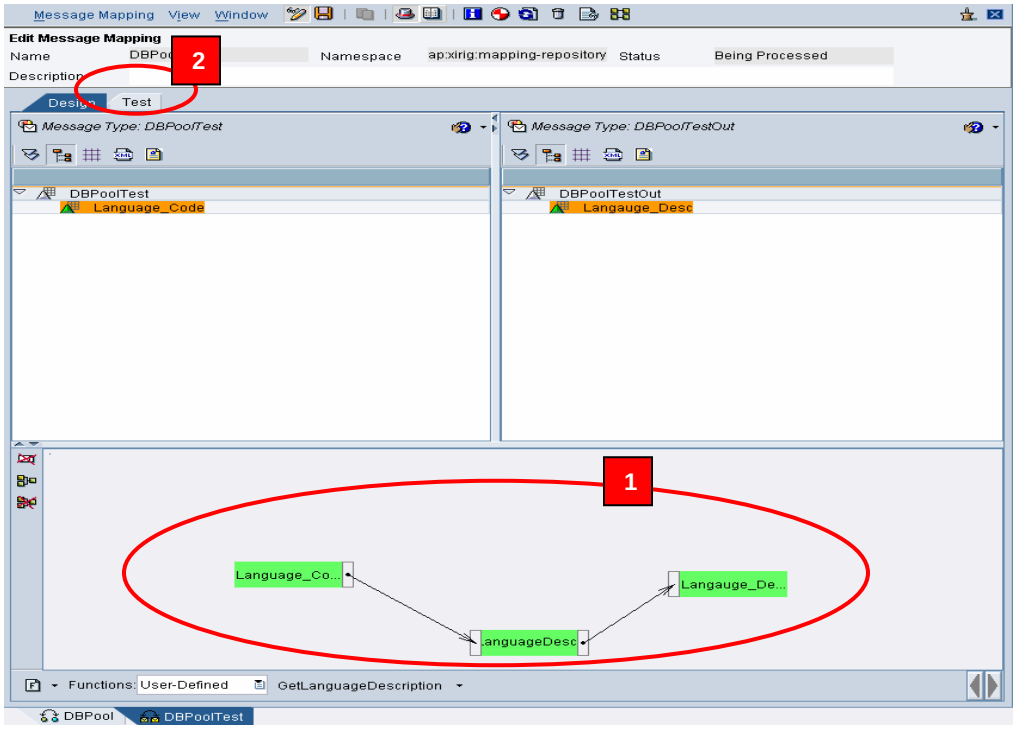


Import the archive into the repository utilizing the “Import Archive” functionality. For documentation purposes the source code and compile statement were also imported in our zip file example.

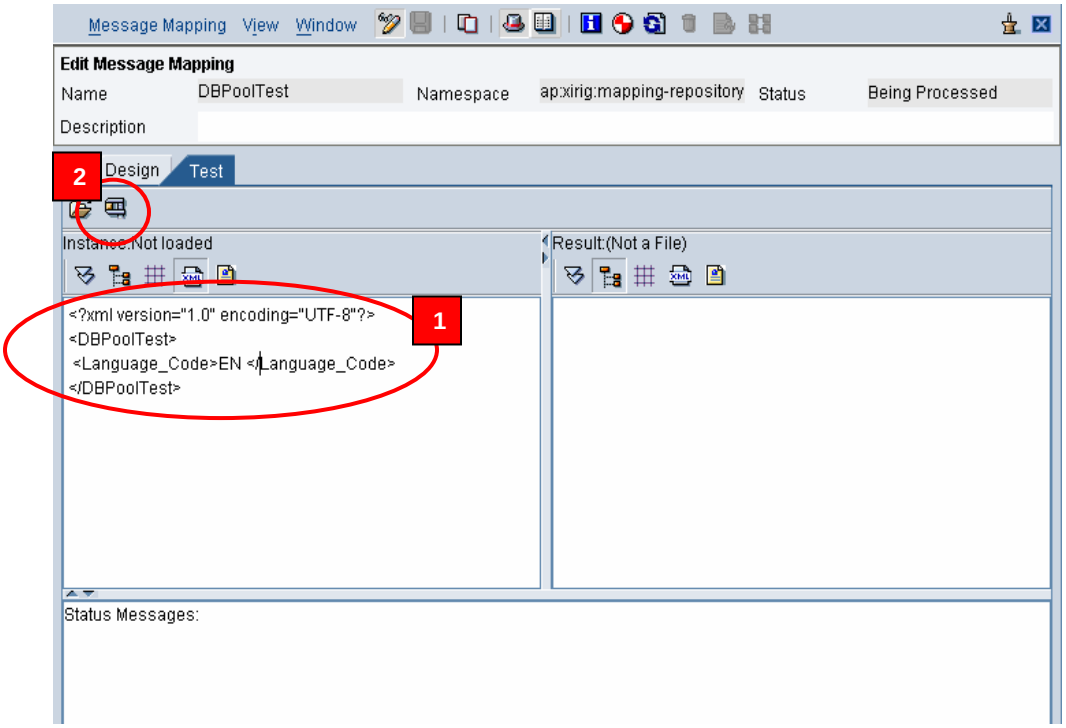
8.1.4 Develop a user-defined function in the Integration Repository utilizing the graphical mapping tool.



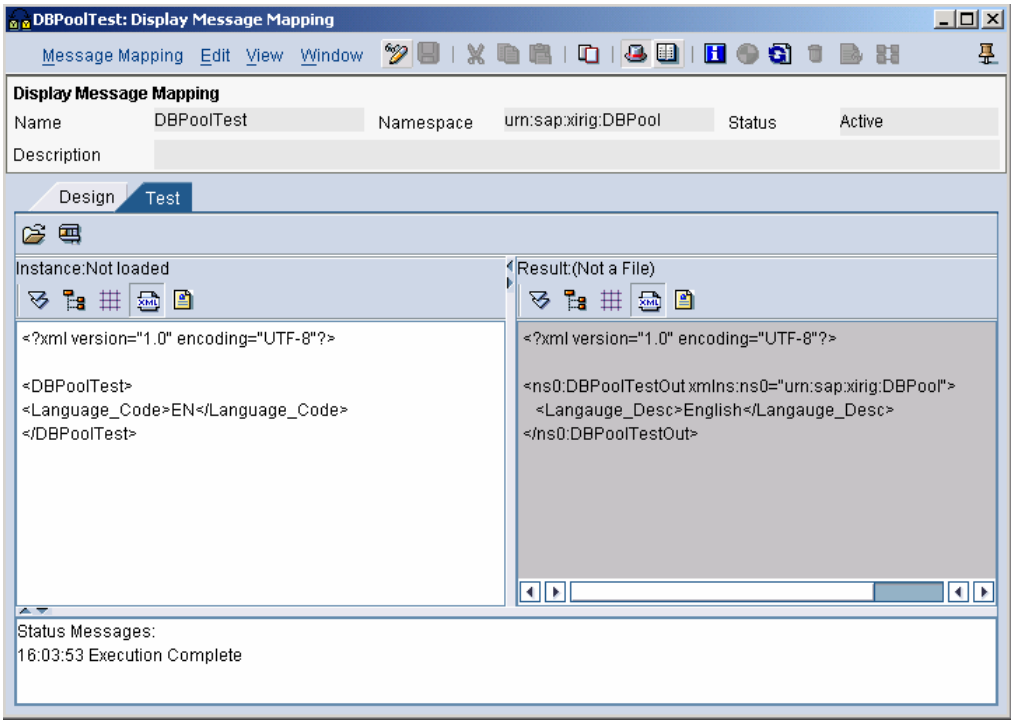
In our example, we call this user-defined module *GetLanguageDescription*. This code first instantiates the class and then calls the method to do the translation.



Create a mapping where the user-defined function is utilized and then proceed to test.



Test the mapping.



Check your results.

8.2 Call DB Pool lookup method from XSLT.

The steps are as follows:

1. Declare database tables to use for this example. (as seen in step 8.1.1)
2. Declare the database connection pool utilizing the J2EE Administrator tool. (as seen in step 8.1.2)
3. Develop Java code that performs the database lookup leveraging the DBPool connection. (as seen in step 8.1.3)
4. Develop an XSL stylesheet.

```
<?xml version="1.0" ?>
- <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:DBPool="com.sap.xirig.DBPool">
    <xsl:script implements-prefix="DBPool" language="java" src="java:com.sap.xirig.DBPool" />
    <xsl:variable name="v_Instance" select="DBPool:new()" />
    <xsl:variable name="v_value" select="//Language_Code" />
  - <xsl:template match="*">
    - <DBPoolTestOut>
      - <Language_Desc>
          <xsl:value-of select="DBPool:getLanguageDesc($v_Instance, string
            ($v_value))" />
        </Language_Desc>
      </DBPoolTestOut>
    </xsl:template>
  </xsl:stylesheet>
```

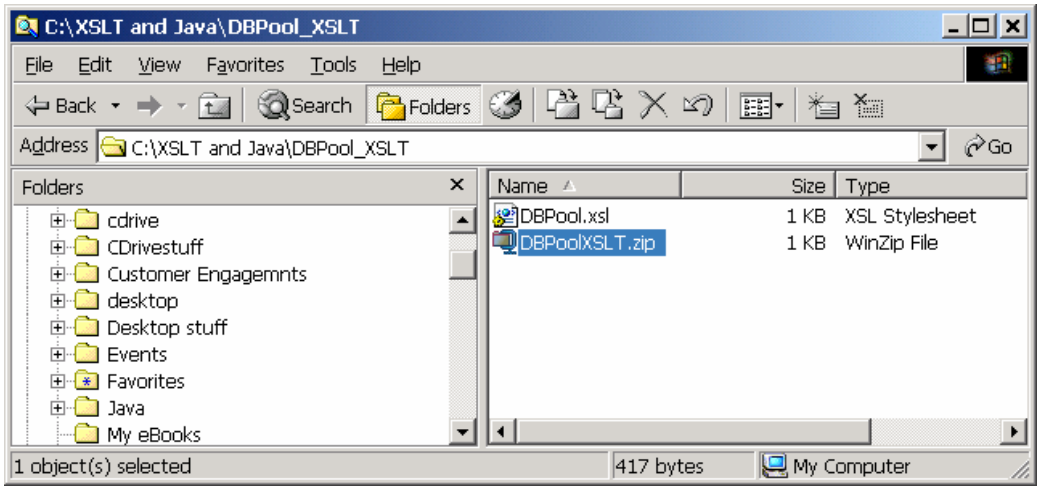
The above XSLT example illustrates how to call the DB pool code from within the stylesheet.

The input XML, before the XSLT mapping:

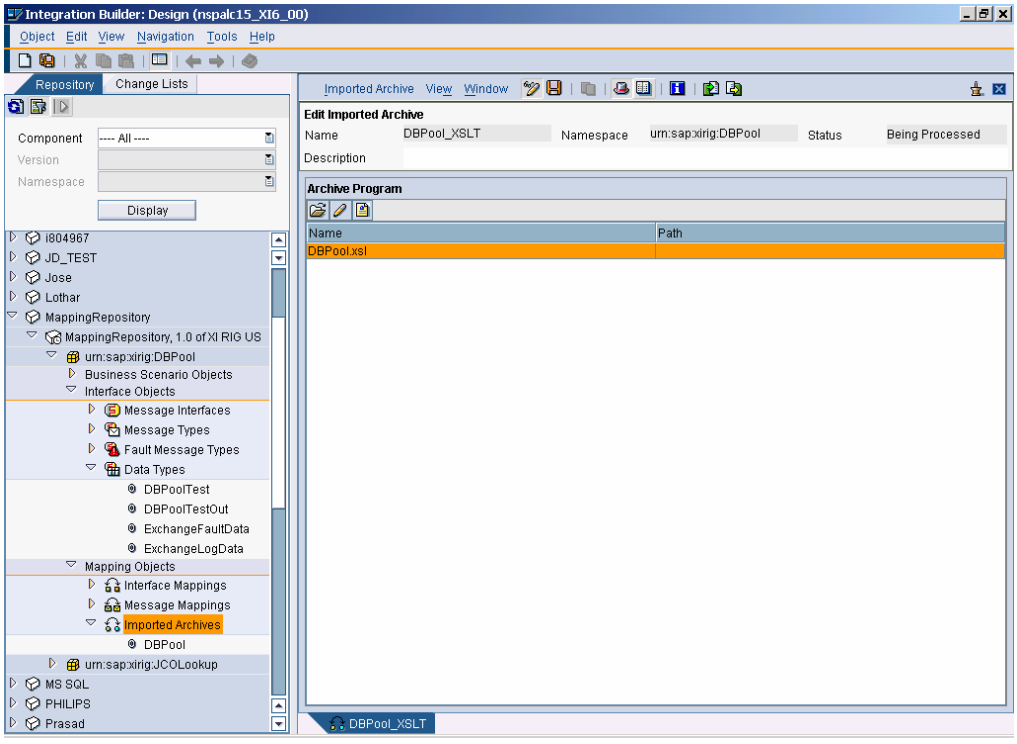
```
<?xml version="1.0" encoding="UTF-8" ?>
- <DBPoolTest>
    <Language_Code>EN</Language_Code>
  </DBPoolTest>
```

The output XML, after mapping:

```
<?xml version="1.0" encoding="utf-8" ?>
- <DBPoolTestOut>
    <Language_Desc>English</Language_Desc>
  </DBPoolTestOut>
```



Create an XSL stylesheet based on the code seen above. In our example we named it DBPool.xsl. Proceed to create a zip file from it.



Import the archive containing the XSL Stylesheet.

8.3. Reference external Java libraries within the same SW component/namespace

In most cases it makes sense to make some mapping logic available through one or more separate Java classes. The idea is to be able to reuse those classes from any mapping object (graphical mapping or XSLT) within the **same** SW component and namespace.

The procedure to follow is simple: Each JAR/ZIP file should be imported into the XI Integration Repository, as an “Imported Archive” object. Then it should be accessible from any mapping object within the same SW Component and namespace, being this a graphical map or XSLT.

For example, for the XSLT mapping described in section 8.2, we separated the Java classes from the XSLT mapping itself. Each of them was included in a separate archive.

8.4. Reference external Java libraries across all SW components/namespaces

This is a more complex procedure which involves system configuration of the SAP J2EE engine.

This should only be performed by advanced users, or with the assistance of a J2EE administrator.

If one or more classes are to be reused across different SW components and namespaces, it makes sense to configure the SAP J2EE Engine to expose them as shared libraries.

These shared libraries can be accessible from one or several J2EE applications. It is important to understand that XI consists of several J2EE applications.

In particular the J2EE applications which are relevant for this exercise are the following:
“IntegrationServices”: mapping runtime
“ExchangeRepository”: integration repository

Our goal in this exercise is to take the Java CLASS defined in 8.1 and make it visible across the whole XI design-time and runtime environments.

The steps to take are as follows:

1. Upload the libraries onto the file system where the SAP J2EE Engine resides
2. Adjust library.txt and reference.txt to expose the shared libraries
3. Restart J2EE Engine.

8.4.1 Upload libraries onto the J2EE engine file system

Every JAR file must be copied into the directory \usr\sap\...\j2ee\cluster\server\additional-lib

8.4.2 Adjust library.txt and reference.txt

These files are located in the directory \usr\sap\...\j2ee\cluster\server\managers
First an entry needs to be added in the file library.txt. The purpose is to define an alias for each library which needs to be shared. The entry should be as follows:

```
library DBPool DBPool.jar
```

Now entries need to be added to the tile reference.txt. The purpose is to make each library visible to the appropriate J2EE applications, as well as to define interdependencies between libraries. In our case the entries are as follows:

```
reference library:DBPool library:jdbc
```

(The `DBPool` class needs access to JDBC, which has already been defined for jdbc20.jar)

```
reference IntegrationServices library:DBPool
```

(make DBPool visible to mapping runtime)

```
reference ExchangeRepository library:DBPool
```

(make DBPool visible to design-time)

in XI 3.0 the restriction of using imported archives only within the same namespace in a software component version does not exist anymore.

In XI 3.0 it will be possible to refer all imported archives in a software component (across namespaces) and even from underlying components (e.g. CRM->ABA).

Deploying mapping code as explained above on the J2EE is a temporary workaround for XI 2.0.

8.4.3 Restart J2EE engine

Follow standard procedure from transaction SMICM.
The classes in DBPool.jar should now be accessible from all mapping objects.

Please note, this is applicable to SAP Web AS 6.20 only. In release 6.30, the procedure will be enhanced.

9. Conclusion

The principles of performing the JDBC lookup itself are straightforward. Any programmer with basic JDBC knowledge should be able to implement this functionality.

The integration of external Java libraries into XI can be done at different levels, depending on the degree of reusability that needs to be achieved:

Integration of single method as User-defined function in graphical mapping	User-defined function is only visible within the same mapping object
Java classes as imported archives within XI Integration Repository	Classes are visible from any mapping (GUI or XSLT) within the same SW

	Component/Namespace
Java classes as shared libraries at the J2EE engine level	Classes are visible throughout the XI design-time and runtime