



Solutions to Complex Data Integration



Transformation Manager Long-Lived Deployment

2008



Contents

INTRODUCTION	3
LONG-LIVED DEPLOYMENT EXAMPLE	3
FILE SETUP.....	3
<i>Step 1 – File Creation</i>	3
<i>Step 2 - Main</i>	3
INITIALISING THE TRANSFORMATION SYSTEM	4
<i>Step 3 – Initialise</i>	4
<i>Step 4 – New Transformer</i>	4
<i>Step 5 – Source Adapter</i>	5
<i>Step 6 – Target Adapter</i>	6
<i>Step 7 – Initialise TMTransformExecutor</i>	7
RUNNING THE TRANSFORMATIONS.....	9
<i>Step 8 – Accessing the Source XML files</i>	9
<i>Step 9 – Setting the Source File</i>	9
<i>Step 10 – Transforming the Data</i>	10
EXECUTION	11
<i>Step 11 – Running the Program</i>	11
DIFFERENCES FOR WRITING TO MULTIPLE XML TARGETS.....	12
<i>Step 6 (Replacement) – XML Target Adapter</i>	12
<i>Step 7 (Replacement) – Initialise XML TMTransformExecutor</i>	13
<i>Step 9b (Additional) – Setting the XML Target File</i>	14
APPENDIX A – ENTIRE FILE XML TO DATABASE	16
APPENDIX B – ENTIRE FILE XML TO MULTIPLE XML FILES	19
FURTHER INFORMATION	21



Introduction

Transformation Manager transforms can be deployed in one of two ways. The simplest approach is to deploy a transform that executes in a single pass reading data from the source, transforming the data and writing to the target. This is referred to as a short-lived transform as the transform is terminated on completion of the transform. An alternative is to deploy a transform that waits for source data to be available before transforming the data and writing to the target. This form of transform is executed many times and is referred to as a long-lived transform.

This document first describes a long-lived deployment example which reads XML files from a specific directory at 10 second intervals, transforms the data and writes to a database. It then describes the changes required to create a deployment example which reads XML files from a specific directory and writes to multiple target XML files.

Code fragments are detailed at each step.

Code snippets displayed in **bold** indicate new lines that have been added to the example.

The general principles outlined in the examples apply equally to long-lived transforms reading and writing from different sources and targets.

The Appendix contains complete example code for the two examples.

Long-lived Deployment Example

File Setup

The example code is built up in a single Java file.

Step 1 – File Creation

We start with the outline of a java file containing the main building blocks of any application. Begin by defining the package and class for your deployed code.

```
package longlived;

public class LongLived
{
    public LongLived()
    {
    }
}
```

Step 2 - Main

As the entire example will be self-contained in a single file we also add a main method in order to run the application. The main method below contains a single line to create an instance of the LongLived object.

```
package longlived;

public class LongLived
{
    static public void main(String[] args)
    {
        LongLived longLived = new LongLived();
    }
}
```

```
}  
  
public LongLived()  
{  
}  
}
```

Now these steps are complete, the actual methods to execute the transforms can be considered.

Initialising the Transformation System

The first method to be created is the initialisation method. This sets up the connections to the source and target data sources. For the purpose of this example the source will be taken from an XML file, with the data being written to a target Database.

Step 3 – Initialise

Add the outline of the method “initSystem”.

```
package longlived;  
  
public class LongLived  
{  
    static public void main(String[] args)  
    {  
        LongLived longLived = new LongLived();  
        longLived.initSystem();  
    }  
  
    public LongLived()  
    {  
    }  
  
    public void initSystem()  
    {  
        System.out.println("Long-lived XML to Database Deployment Example");  
    }  
}
```

Step 4 – New Transformer

The first step of the initialisation is to create a new “Long-lived” TMTransformExecutor object. This is the main object used to carry out the transformations which have been created using the TM Design Tool.

Within this example the TMTransformExecutor object will be used in both the initialisation method (to configure its source and targets) and secondly in a method to run the transformations, and therefore be created as a class member.

```
package longlived;  
  
import net.etl.*;  
  
public class LongLived  
{  
    //Class Members  
    TMTransformExecutor myTMTransformer;  
  
    static public void main(String[] args)  
    {  
        LongLived longLived = new LongLived();  
        longLived.initSystem();  
    }  
}
```

```
}

public LongLived()
{
}

public void initSystem()
{
    System.out.println("Long-lived XML to Database Deployment Example");

    try
    {
        //Long-lived
        myTMTransformer =
            TMTransformerFactory.getNewTMTransformExecutor(this,"longLivedProj",1);
    }
    catch (TMException atMException)
    {
        atMException.printStackTrace();
        return;
    }
}
}
```

1. First add the import line at the top of the file. Declare myTMTransformer as a global variable of type TMTransformExecutor.
2. A “Try-Catch” block is required around the creation of the TMTransformExecutor.
3. The static factory method used to create the TMTransformExecutor is described in detail within the API documentation found within the installation directory \TransMan\docs\api-docs\index.html.

The 3 parameters passed in are:

1. userRefObject – This can be any java object. It may be retrieved later in any of the java functions defined in the SML project.
2. Project Name – The name of the project for the transformations.
3. The Version – The major version of the project for the transformations.

Step 5 – Source Adapter

The next step is to set up the source adapter. The source adapter is used to access the source instance data for transformation. As the source will be required in later methods also for this example, it is set as a global variable.

```
package longlived;

import net.etlrm.*;

public class LongLived
{
    //Global Variables
    TMTransformExecutor myTMTransformer;
    TMXMLAdapter myTMXMLSourceAdapter;

    ...<code omitted due to space>.

    public void initSystem()
    {
        System.out.println("Long-lived XML to Database Deployment Example");

        try
        {
            //Long-lived
```



```
myTMTransformer =
    TMTransformerFactory.getNewTMTransformExecutor(this, "longLivedProj", 1);

//Create SOURCE Adapter
myTMXMLSourceAdapter =
    TMBuiltInAdaptorFactory.makeNewReadTMXMLAdapter(myTMTransformer);

//Set the source adapter for the TMTransformExecutor
myTMTransformer.setSource("longLivedProj", myTMXMLSourceAdapter);

}
catch (TMException aTMException)
{
    aTMException.printStackTrace();
    return;
}
}
```

1. The source XML adapter is created in a single line directly from TM's own adapter factory. Further detail can be found in the API documentation.
2. The second line entered sets the source adapter for the TMTransformExecutor object created in Step 4. Hence, the TMTransformExecutor will be able to use the TMXMLAdapter when accessing the source data.
3. The third setting in simple examples is usually the setting of the instance connection. However, as this example requires the source from a number of files being retrieved in real-time from a directory we simply set the session to connect. Sessions are used to group operations together within the adapter. Hence, in this example when the time comes to connect to a particular file and transform it, the session '0' will be started with the particular file to transform passed into the TMXMLAdapter at that point (As shown in Step 9).

Step 6 – Target Adapter

The target adapter is created in a similar fashion to the source adapter.

1. An Additional import is required for the Hashmap in the form of java.util.
2. As the target for the transformation will be a database, a TMJDBCAdapter is created in a single line directly from TM's own adapter factory. Further detail can be found in the API documentation.
3. myTMTransformer has the setTarget method called to indicate to the transformer which adapter will be used for the writing of information to the target.
4. In order to pass the database connection information into the adapter a hash map of values is used. Common database parameters are added to the hash map using pre-defined static variables (such as NAME_USER, NAME_PASSWORD, NAME_URL and NAME_DRIVER) which can be found in TMJDBCAdapter. These are described in greater detail within the API documentation found within the installation directory \TransMan\docs\api-docs\index.html.

Note: As a MySQL database is being used as the target, an ODBC connection has to be set up to the database (Data Sources (ODBC) from the Administrative Tools within Windows).

```
package longlived;

import net.etlrm.*;
import java.util.*;

public class LongLived
{
    //Global Variables
```

```
TMTransformExecutor myTMTransformer;
TMXMLAdapter myTMXMLSourceAdapter;

...<code omitted due to space>.

public void initSystem()
{
    System.out.println("Long-lived XML to Database Deployment Example");

    try
    {
        //Long-lived
        myTMTransformer =
            TMTransformerFactory.getNewTMTransformExecutor(this,"longLivedProj",1);

        //Create SOURCE Adapter
        myTMXMLSourceAdapter =
            TMBuiltInAdaptorFactory.makeNewReadTMXMLAdapter(myTMTransformer);

        //Set the source adapter for the TMTransformExecutor
        myTMTransformer.setSource("longLivedProj",myTMXMLSourceAdapter);

        //TARGET
        TMJDBCAdapter aTMJDBCAdapter =
            TMBuiltInAdaptorFactory.makeNewWriteTMJDBCAdapter(myTMTransformer);

        //Set the source adapter for the TMTransformExecutor
        myTMTransformer.setTarget("longLivedProj",aTMJDBCAdapter);

        //Set up database connection properties
        Map params = new HashMap();
        params.put(TMJDBCAdapter.NAME_USER,"root");
        params.put(TMJDBCAdapter.NAME_PASSWORD,"root");
        params.put(TMJDBCAdapter.NAME_URL,"jdbc:odbc:mysqlTest");
        params.put(TMJDBCAdapter.NAME_DRIVER,"sun.jdbc.odbc.JdbcOdbcDriver");
        aTMJDBCAdapter.setInstanceConnectionData(params);

    }
    catch (TMException aTMException)
    {
        aTMException.printStackTrace();
        return;
    }
}
}
```

Step 7 – Initialise TMTransformExecutor

The resource and target information are opened separately within the TMTransformExecutor, as the source will be handled separately for the files being received.

Finally add the loop which will run the transformations every 10 seconds.

```
package longlived;

import net.etl.*;
import java.util.*;

public class LongLived
{
    //Global Variables
    TMTransformExecutor myTMTransformer;
    TMXMLAdapter myTMXMLSourceAdapter;
```



```
...<code omitted due to space>.

public void initSystem()
{
    System.out.println("Long-lived XML to Database Deployment Example");

    try
    {
        //Long-lived
        myTMTransformer =
            TMTransformerFactory.getNewTMTransformExecutor(this, "longLivedProj", 1);

        //Create SOURCE Adapter
        myTMXMLSourceAdapter =
            TMBuiltInAdaptorFactory.makeNewReadTMXMLAdapter(myTMTransformer);

        //Set the source adapter for the TMTransformExecutor
        myTMTransformer.setSource("longLivedProj", myTMXMLSourceAdapter);

        //TARGET
        TMJDBCAdapter aTMJDBCAdapter =
            TMBuiltInAdaptorFactory.makeNewWriteTMJDBCAdapter(myTMTransformer);

        //Set the source adapter for the TMTransformExecutor
        myTMTransformer.setTarget("longLivedProj", aTMJDBCAdapter);

        //Set up database connection properties
        Map params = new HashMap();
        params.put(TMJDBCAdapter.NAME_USER, "root");
        params.put(TMJDBCAdapter.NAME_PASSWORD, "root");
        params.put(TMJDBCAdapter.NAME_URL, "jdbc:odbc:mysqlTest");
        params.put(TMJDBCAdapter.NAME_DRIVER, "sun.jdbc.odbc.JdbcOdbcDriver");
        aTMJDBCAdapter.setInstanceConnectionData(params);

        //As the source will be handled differently, resources and target information
        //are opened seperately
        myTMTransformer.openOnlyProjectResources();
        myTMTransformer.openPrimaryTarget();

        //Loop to run the transformations every 10 seconds
        while (true)
        {
            runTransform();
            try
            {
                Thread.sleep(10000);
            }
            catch (InterruptedException ie)
            {
                System.out.println("Iterupt");
            }
        }
    }
    catch (TMException atMException)
    {
        atMException.printStackTrace();
        return;
    }
}
}
```

The following steps will describe how **runTransform()** is created.

Running the Transformations

Step 8 – Accessing the Source XML files

1. Add an import line at the top of the file for java.io which will be required for the XML file handling.

```
import java.util.*;
import net.etltm.*;
import java.io.*;
```

The runTransform() method is located below the initSystem() method however in the interest of space only the runTransform() method is shown below.

For this example it is assumed that the directory specified will contain only XML and no extension filtering is carried out.

2. Create an array of the files in the directory and prepare to loop over each file.

```
private void runTransform()
{
    //Get a list of files in the specified directory
    String filename;
    File[] files;
    File f = new File("d:\\xml\\");
    files = f.listFiles();

    //LOOP for all files currently in the directory
    for(int i=0; i< files.length; i++)
    {
        filename = files[i].getAbsolutePath();
        System.out.println(filename);
    }
}
```

Step 9 – Setting the Source File

1. Create a hash map to contain the source file to be transformed, using the pre-defined static key TMXMLAdapter.NAME_URL for identification.
2. A session is begun for the TMXMLAdapter with an identity of '1'. The parameter hash map denoting the file to be transformed is also passed in at this time. Sessions are used to group operations together within the adapter and hence a new session will be started for each individual XML file.

Further code will be added in Step 10 to run the transformation, which is followed by the ending of the session for the TMXMLAdapter.

```
private void runTransform()
{
    //Get a list of files in the specified directory
    String filename;
    File[] files;
    File f = new File("d:\\xml\\");
    files = f.listFiles();

    //LOOP for all files currently in the directory
    for(int i=0; i< files.length; i++)
    {
        filename = files[i].getAbsolutePath();
        System.out.println(filename);

        try
        {
```

```

//Add the source XML file to the hash map
Map param = new HashMap ();
param.put(TMXMLAdapter.NAME_URL, filename);

//begin the session
myTMXMLSourceAdapter.beginSession(1,param);

//CODE FOR STEP 10

//end the session
myTMXMLSourceAdapter.endSession(1);
}
catch (TMException aTMException)
{
    aTMException.printStackTrace();
    return;
}
}
}

```

Step 10 – Transforming the Data

For this example the source instance data follows the ‘BOOKS’ dtd structure within the TransMan\Samples\XML directory.

1. A new import is required at the top of the file for the TMDHIterator described below.

```

package longlived;

import java.util.*;
import net.etltn.*;
import java.io.*;
import net.etltn.qp.TMQueryObjectsFactory;

```

2. A TMDHIterator is created to iterate through the instances of ‘BOOK’ within the XML file. This is created by getting the objects of interest from the source TMXMLAdapter which in turn allows the creation of a query set (similar to a where clause in SQL) in this case passing in the parameter ‘BOOK’ – the item to be transformed.

```

private void runTransform()
{
    //Get a list of files in the specified directory
    String filename;
    File[] files;
    File f = new File("d:\\xml\\");
    files = f.listFiles();

    //LOOP for all files currently in the directory
    for(int i=0; i< files.length; i++)
    {
        filename = files[i].getAbsolutePath();
        System.out.println(filename);

        try
        {
            //Add the source XML file to the hash map
            Map param = new HashMap ();
            param.put(TMXMLAdapter.NAME_URL, filename);

            //begin the session
            myTMXMLSourceAdapter.beginSession(1,param);

            //Create an iterator to select the object for mapping
            TMDHIterator aTMDHIterator =

```

```
myTMXMLSourceAdapter.getObjectsOfInterest
(TMQueryObjectsFactory.createTMQuerySet("BOOK"));
//Loop through each 'Book'
while (true)
{
    TMDH aTMDH = aTMDHIterator.getNextDH();
    if (null==aTMDH)
    {
        break;
    }
    //Transform the data
    myTMTransformer.transform(aTMDH);
}

//end the session
myTMXMLSourceAdapter.endSession(1);
}
catch (TMException aTMException)
{
    aTMException.printStackTrace();
    return;
}
//DELETE FILES IF REQUIRED
}
}
```

3. A simple while loop is then used to iterate through the objects of interest retrieved within the TMDHIterator.
4. A TMDH object is used to store each object of interest within the TMDHIterator.
5. Finally the TMTransformExecutor (myTMTransformer) calls the 'transform' method to transform the object of interest currently retrieved within the while loop.
6. Additional code can be added at the end of the method to delete the files which have been processed in this run.

Execution

Step 11 – Running the Program

In order to run the java file created using the above steps, ensure you have the following in your classpath.

1. ofetsr.jar (Which can be found in the installed 'TransMan\lib' directory)
2. ETL_longLivedProj_1_0.jar (The jar file created from your TM Design Tool mappings, located by default in 'TransMan\com\prismt\transform\maps\longLivedProj\VI\m0')



Differences for Writing to Multiple XML Targets

The following describes the changes required if writing to multiple XML target files instead of the database as shown in the previous example.

Step 6 (Replacement) – XML Target Adapter

The target adapter is created identically to the source adapter.

1. The source XML adapter is created in a single line directly from TM's own adapter factory. Further detail can be found in the API documentation.
2. myTMTransformer has the setTarget method called to indicate to the transformer which adapter will be used for the writing of information to the target.
3. The third setting in simple examples is usually the setting of the instance connection. However, as this example requires the data to be written to a number of different target files to be specified when the transform occurs, we simply set the session to connect to. Sessions are used to group operations together within the adapter, as shown in this example. When the time comes to write to a particular file, the session '1' will be started for the target XML Adapter with the particular file name to be written passed into the TMXMLAdapter at that point (As shown in Step 9b – Setting the XML Target File).

```
package longlived;

import net.etl.*;
import java.util.*;

public class LongLived
{
    //Class members
    TMTransformExecutor myTMTransformer;
    TMXMLAdapter myTMXMLSourceAdapter;

    ...<code omitted due to space>.

    public void initSystem()
    {
        System.out.println("Long-lived XML to Database Deployment Example");

        try
        {
            //Long-lived
            myTMTransformer =
                TMTransformerFactory.getNewTMTransformExecutor(this, "longLivedProj", 1);

            //Create SOURCE Adapter
            myTMXMLSourceAdapter =
                TMBuiltInAdaptorFactory.makeNewReadTMXMLAdapter(myTMTransformer);

            //Set the source adapter for the TMTransformExecutor
            myTMTransformer.setSource("longLivedProj", myTMXMLSourceAdapter);

            //set the session level for the source adapter
            myTMXMLSourceAdapter.setSessionLevelUsedToConnect(1);

            //TARGET
            myTMXMLTargetAdapter =
                TMBuiltInAdaptorFactory.makeNewWriteTMXMLAdapter(myTMTransformer);

            //Set the source adapter for the TMTransformExecutor
            myTMTransformer.setTarget("longLivedProj", myTMXMLTargetAdapter);
        }
    }
}
```

```

        //set the session level for the target adapter
        myTMXMLTargetAdapter.setSessionLevelUsedToConnect(1);
    }
    catch (TMException atMException)
    {
        atMException.printStackTrace();
        return;
    }
}
}

```

Step 7 (Replacement) – Initialise XML TMTransformExecutor

1. The resource information is opened separately within the TMTransformExecutor, as the source **and target** will be handled separately for the files being received and written respectively. The difference from the original Step 7 is the removal of the line:

```
myTMTransformer.openPrimaryTarget();
```

This is now described in Step 9b below.

2. Add the loop which will run the transformations every 10 seconds. This is described in the ‘Running the Transforms’ section.

```

package longlived;

import net.etlrm.*;
import java.util.*;

public class LongLived
{
    //Class members
    TMTransformExecutor myTMTransformer;
    TMXMLAdapter myTMXMLSourceAdapter;

    ...<code omitted due to space>.

    public void initSystem()
    {
        System.out.println("Long-lived XML to Database Deployment Example");

        try
        {
            //Long-lived
            myTMTransformer =
                TMTransformerFactory.getNewTMTransformExecutor(this,"longLivedProj",1);

            //Create SOURCE Adapter
            myTMXMLSourceAdapter =
                TMBuiltInAdaptorFactory.makeNewReadTMXMLAdapter(myTMTransformer);

            //Set the source adapter for the TMTransformExecutor
            myTMTransformer.setSource("longLivedProj",myTMXMLSourceAdapter);

            //set the session level for the source adapter
            myTMXMLSourceAdapter.setSessionLevelUsedToConnect(1);

            //TARGET
            TMJDBCAdapter aTMJDBCAdapter =
                TMBuiltInAdaptorFactory.makeNewWriteTMJDBCAdapter(myTMTransformer);

            //Set the source adapter for the TMTransformExecutor
            myTMTransformer.setTarget("longLivedProj",aTMJDBCAdapter);

            //Set up database connection properties

```

```

Map params = new HashMap();
params.put(TMJDBCAdapter.NAME_USER, "root");
params.put(TMJDBCAdapter.NAME_PASSWORD, "root");
params.put(TMJDBCAdapter.NAME_URL, "jdbc:odbc:mysqlTest");
params.put(TMJDBCAdapter.NAME_DRIVER, "sun.jdbc.odbc.JdbcOdbcDriver");
atMJDBCAdapter.setInstanceConnectionData(params);

//As the source and target will be handled differently,
//only the resources information is opened here
myTMTransformer.openOnlyProjectResources();

//Loop to run the transformations every 10 seconds
while (true)
{
    runTransform();
    try
    {
        Thread.sleep(10000);
    }
    catch (InterruptedException ie)
    {
        System.out.println("Iterupt");
    }
}
}
catch (TMException atMException)
{
    atMException.printStackTrace();
    return;
}
}
}

```

Step 9b (Additional) – Setting the XML Target File

Step 9 described above is required for setting the source XML file. However, an additional step is now required between steps 9 and 10 to set the target XML file also.

1. Create a hash map to contain the target file to be transformed, using the pre-defined static key `TMXMLAdapter.NAME_URL` for identification. Current time in milliseconds is added to the end of the file name in order to generate a unique file each time in this example.
2. Session is begun for the target `TMXMLAdapter` with an identity of '1'. The parameter hash map denoting the file to be transformed is also passed in at this time. Sessions are used to group operations together within the adapter and hence a new session will be started for each individual XML file.
3. IMPORTANT – The target XML Adapter is now opened after the target XML file has been specified. This has moved from Step 7 of the original example.
4. The final line to be added is the ending of the session for the target `TMXMLAdapter` in an identical way to the source XML adapter.

```

private void runTransform()
{
    //Get a list of files in the specified directory
    String filename;
    File[] files;
    File f = new File("d:\\xml\\");
    files = f.listFiles();

    //LOOP for all files currently in the directory
    for(int i=0; i< files.length; i++)
    {
        filename = files[i].getAbsolutePath();
    }
}

```



```
System.out.println(filename);

try
{
    //Add the source XML file to the hash map
    Map param = new HashMap ();
    param.put(TMXMLAdapter.NAME_URL, filename);

    //Set the target adapters file to read in
    Map targetParam = new HashMap ();
    targetParam.put(TMXMLAdapter.NAME_URL,
        "d:\\xml\\target"+System.currentTimeMillis());

    //begin the session
    myTMXMLSourceAdapter.beginSession(1,param);
    myTMXMLTargetAdapter.beginSession(1, targetParam);

    //Target must now be opened after the specific target file
    //has been specified
    myTMTransformer.openPrimaryTarget();

    //CODE FOR STEP 10

    //end the session
    myTMXMLSourceAdapter.endSession(1);
    myTMXMLTargetAdapter.endSession(1);
}
catch (TMException aTMException)
{
    aTMException.printStackTrace();
    return;
}
}
```



Appendix A – Entire File XML to Database

```
package longlived;

import net.etltm.*;
import java.util.*;
import java.io.*;
import net.etltm.qp.TMQueryObjectsFactory;

public class LongLived
{
    //Class members
    TMTransformExecutor myTMTransformer;
    TMXMLAdapter myTMXMLSourceAdapter;

    static public void main(String[] args)
    {
        LongLived longLived = new LongLived();
        longLived.initSystem();
    }

    public LongLived()
    {
    }

    public void initSystem()
    {
        System.out.println("Long-lived XML to Database Deployment Example");

        try
        {
            //Long-lived
            myTMTransformer =
                TMTransformerFactory.getNewTMTransformExecutor(this,"longLivedProj",1);

            //Create SOURCE Adapter
            myTMXMLSourceAdapter =
                TMBuiltInAdaptorFactory.makeNewReadTMXMLAdapter(myTMTransformer);

            //Set the source adapter for the TMTransformExecutor
            myTMTransformer.setSource("longLivedProj",myTMXMLSourceAdapter);

            //set the session level for the source adapter
            myTMXMLSourceAdapter.setSessionLevelUsedToConnect(1);

            //TARGET
            TMJDBCAdapter aTMJDBCAdapter =
                TMBuiltInAdaptorFactory.makeNewWriteTMJDBCAdapter(myTMTransformer);

            //Set the source adapter for the TMTransformExecutor
            myTMTransformer.setTarget("longLivedProj",aTMJDBCAdapter);

            //Set up database connection properties
            Map params = new HashMap();
            params.put(TMJDBCAdapter.NAME_USER,"root");
            params.put(TMJDBCAdapter.NAME_PASSWORD,"root");
            params.put(TMJDBCAdapter.NAME_URL,"jdbc:odbc:mysqlTest");
            params.put(TMJDBCAdapter.NAME_DRIVER,"sun.jdbc.odbc.JdbcOdbcDriver");
            aTMJDBCAdapter.setInstanceConnectionData(params);
        }
    }
}
```



```
//As the source will be handled differently, resources and target information
//only, are opened here
myTMTransformer.openOnlyProjectResources();
myTMTransformer.openPrimaryTarget();

//Loop to run the transformations every 10 seconds
while (true)
{
    runTransform();
    try
    {
        Thread.sleep(10000);
    }
    catch (InterruptedException ie)
    {
        System.out.println("Iterupt");
    }
}
catch (TMException atMException)
{
    atMException.printStackTrace();
    return;
}
}

private void runTransform()
{
    //Get a list of files in the specified directory
    String filename;
    File[] files;
    File f = new File("d:\\xml\\");
    files = f.listFiles();

    //LOOP for all files currently in the directory
    for(int i=0; i< files.length; i++)
    {
        filename = files[i].getAbsolutePath();
        System.out.println(filename);

        try
        {
            //Add the source XML file to the hash map
            Map param = new HashMap ();
            param.put(TMXMLAdapter.NAME_URL, filename);

            //begin the session
            myTMXMLSourceAdapter.beginSession(1,param);

            //Create an iterator to select the object for mapping
            TMDHIterator aTMDHIterator =
                myTMXMLSourceAdapter.getObjectsOfInterest
                    (TMQueryObjectsFactory.createTMQuerySet("BOOK"));
            //Loop through each 'Book'
            while (true)
            {
                TMDH aTMDH = aTMDHIterator.getNextDH();
                if (null==aTMDH)
                {
                    break;
                }
                //Transform the data
                myTMTransformer.transform(aTMDH);
            }
        }
    }
}
```

```
        //end the session
        myTMXMLSourceAdapter.endSession(1);
    }
    catch (TMException aTMException)
    {
        aTMException.printStackTrace();
        return;
    }
    //DELETE FILES IF REQUIRED
}
}
```



Appendix B – Entire File XML to Multiple XML Files

```
package longlived;

import net.etltm.*;
import java.util.*;
import java.io.*;
import net.etltm.qp.TMQueryObjectsFactory;

public class LongLived
{
    //Class members
    TMTransformExecutor myTMTransformer;
    TMXMLAdapter myTMXMLSourceAdapter;

    static public void main(String[] args)
    {
        LongLived longLived = new LongLived();
        longLived.initSystem();
    }

    public LongLived()
    {
    }

    public void initSystem()
    {
        System.out.println("Long-lived XML to Database Deployment Example");

        try
        {
            //Long-lived
            myTMTransformer =
                TMTransformerFactory.getNewTMTransformExecutor(this, "longLivedProj", 1);

            //Create SOURCE Adapter
            myTMXMLSourceAdapter =
                TMBuiltInAdaptorFactory.makeNewReadTMXMLAdapter(myTMTransformer);

            //Set the source adapter for the TMTransformExecutor
            myTMTransformer.setSource("longLivedProj", myTMXMLSourceAdapter);

            //set the session level for the source adapter
            myTMXMLSourceAdapter.setSessionLevelUsedToConnect(1);

            //TARGET
            myTMXMLTargetAdapter =
                TMBuiltInAdaptorFactory.makeNewWriteTMXMLAdapter(myTMTransformer);

            //Set the source adapter for the TMTransformExecutor
            myTMTransformer.setTarget("longLivedProj", myTMXMLTargetAdapter);

            //set the session level for the target adapter
            myTMXMLTargetAdapter.setSessionLevelUsedToConnect(1);

            //As the source and target will be handled differently,
            //only the resources information is opened here
            myTMTransformer.openOnlyProjectResources();
        }
    }
}
```



```
//Loop to run the transformations every 10 seconds
while (true)
{
    runTransform();
    try
    {
        Thread.sleep(10000);
    }
    catch (InterruptedException ie)
    {
        System.out.println("Iterupt");
    }
}
}
catch (TMException atMException)
{
    atMException.printStackTrace();
    return;
}
}

private void runTransform()
{
    //Get a list of files in the specified directory
    String filename;
    File[] files;
    File f = new File("d:\\xml\\");
    files = f.listFiles();

    //LOOP for all files currently in the directory
    for(int i=0; i< files.length; i++)
    {
        filename = files[i].getAbsolutePath();
        System.out.println(filename);

        try
        {
            //Add the source XML file to the hash map
            Map param = new HashMap ();
            param.put(TMXMLAdapter.NAME_URL, filename);

            //Set the target adapters file to read in
            Map targetParam = new HashMap ();
            targetParam.put(TMXMLAdapter.NAME_URL,
                "d:\\xml\\target"+System.currentTimeMillis());

            //begin the session
            myTMXMLSourceAdapter.beginSession(1,param);
            myTMXMLTargetAdapter.beginSession(1, targetParam);

            //Target must now be opened after the specific target file
            //has been specified
            myTMTransformer.openPrimaryTarget();

            //Create an iterator to select the object for mapping
            TMDHIterator aTMDHIterator =
                myTMXMLSourceAdapter.getObjectsOfInterest
                    (TMQueryObjectsFactory.createTMQuerySet("BOOK"));

            //Loop through each 'Book'
            while (true)
            {
                TMDH aTMDH = aTMDHIterator.getNextDH();
                if (null==aTMDH)
                {

```

```
        break;
    }
    //Transform the data
    myTMTransformer.transform(atMDH);
}

//end the session
myTMXMLSourceAdapter.endSession(1);
myTMXMLTargetAdapter.endSession(1);
}
catch (TMException aTMException)
{
    aTMException.printStackTrace();
    return;
}
//DELETE FILES IF REQUIRED
}
}
}
```

Further Information

Website : www.etlsolutions.com
Email : info@etlsolutions.com
Telephone : +44 (0) 1912 894040