by experts
for experts

Transformation Manager
Version 5.2

Tutorial 18 - API Deployment

Transformation
Manager

Table of Contents

Tutorial 18

API Deployment

This tutorial shows you how to initiate your transformation projects using java code. This mechanism is known as "short lived" mode which mimics the execution of a project in TM Migrator; that is, the project is initiated and runs to completion. You will need Java development skills in order to implement this deployment and, in essence, the tutorial is more an explanation than a set of exercises to complete. By the end of the tutorial you should be familiar with the steps required to implement a transformation project. The example code used in this tutorial is built up step by step in a single java file and is distributed with the tutorial resources in the following directory.

```
[TMHOME]\Tutorials\Source_and_Target\Tutorial_18_API_Deployment_Short
```

The tutorial will begin by deploying a project that uses an XML data model for both the source and target. The deployment code will then be modified to use a different project with a database as the target data store. In the exercises where your code is being edited the text in bold represents the code added to your Java file at each step.

Prerequisites

Before starting this tutorial we recommend that you have completed the following tasks.

1) Transformation Manager has been installed.
2) An appropriate license has been installed.
3) The tutorial resources including data models, samples and source and target data stores have been downloaded and extracted to your Transformation Manager home directory.
4) You have completed tutorials 1, 2, 3, 4 and 6.
5) You should have completed one of the following tutorials, 5, 7, 8, 9, 11, 13 or 16.
6) You may have completed tutorial 14 or 15.

Information

The starting point for the tutorial is to create two projects in TM Designer that we will use to demonstrate initiating the projects from java code. The first project uses an XML source and and transforms to an XML target. The second project uses the same XML source and transforms to a database target. The projects will be created using the Import... option from the File menu bar.

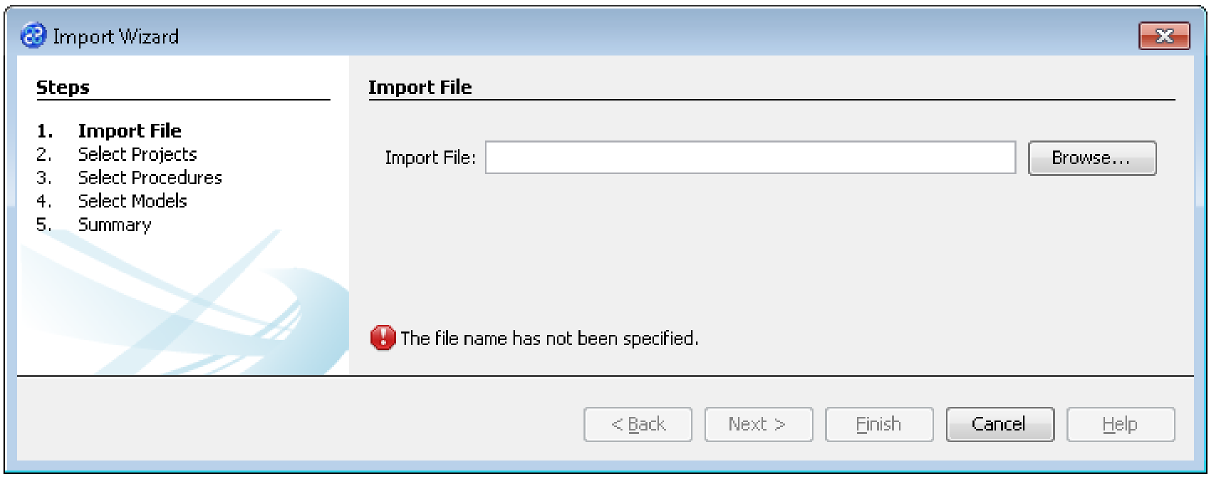
Throughout the exercises you will see the term [TMHOME] used in the java code. You will need to replace this with your own path to [TMHOME].

Exercise 1 - Create the Example Projects

This exercise will import the two example projects into your TM Designer repository which we will use for demonstration purposes in this tutorial.

1) Click on the File menu bar item.

2) Click on the Import... option to open the **Import Wizard** dialog as shown below.



Import File:

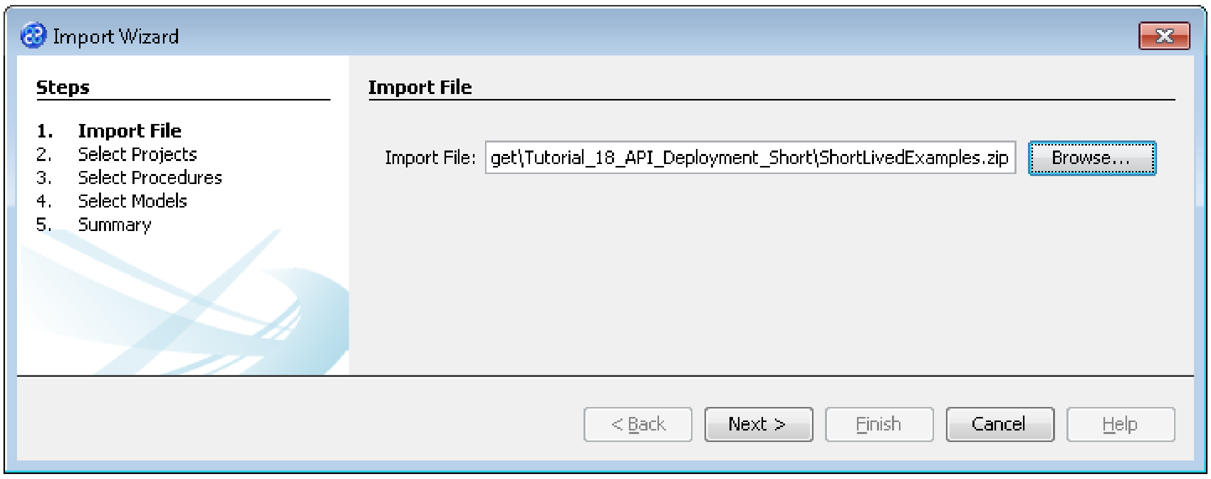
The file name has not been specified.

3) Click the Browse... button to open the **Select File** window.

4) Navigate to the **ShortLivedExamples.zip** file. This can be found in the following directory.

```
[TMHOME]\Tutorials\Source_and_Target\Tutorial_18_API_Deployment_Short
```

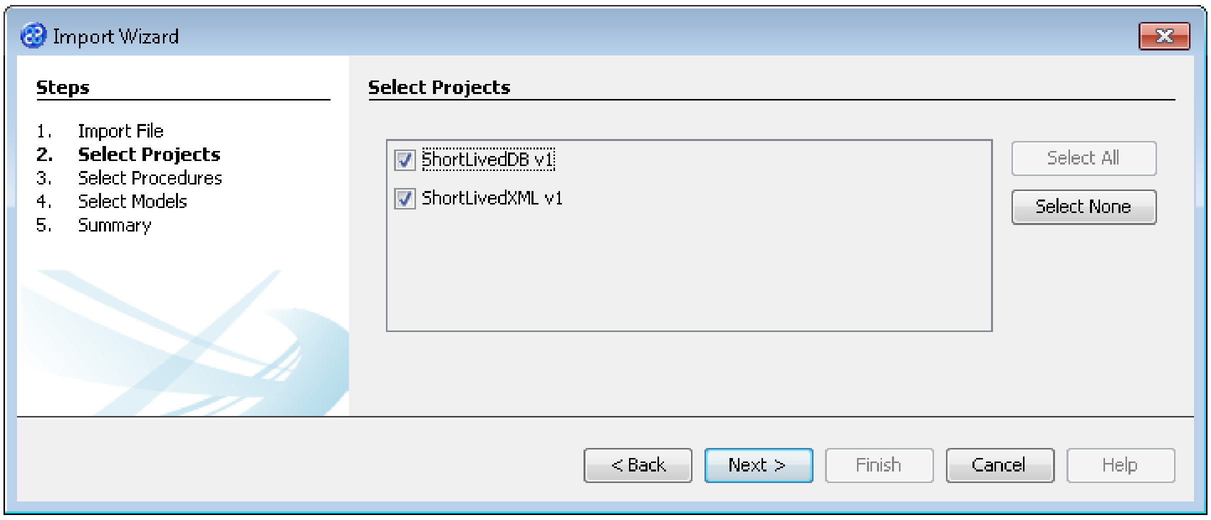
5) Click the Select File button to paste the destination and file name into the **Import File** field. The window should look like the one below.



Import File: get\Tutorial_18_API_Deployment_Short\ShortLivedExamples.zip

6) Click the Next > button to move to the **Select Projects** pane.

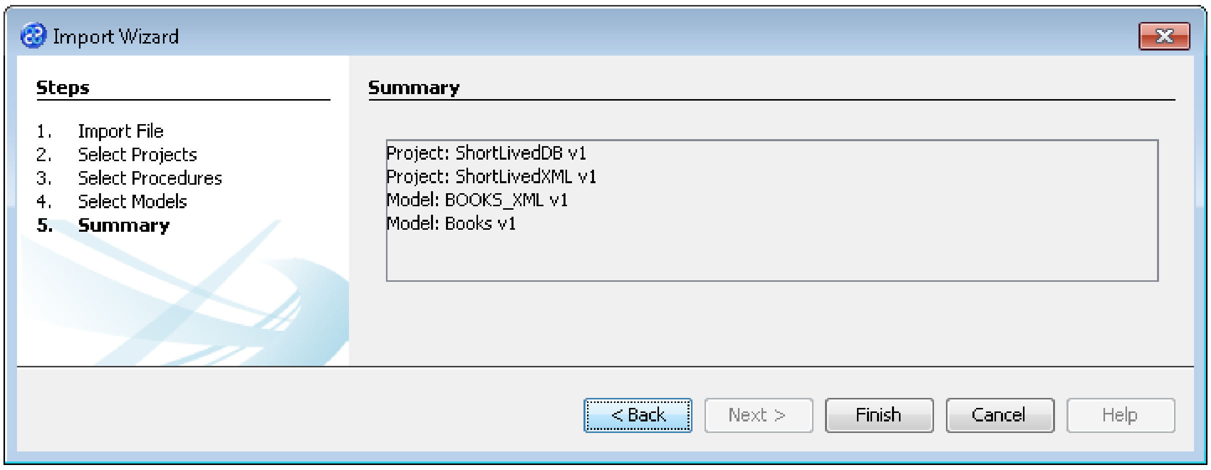
7) Click the Select All button to select both of the projects listed, **ShortLivedXML** and **ShortLivedDB** as shown below.



8) Click the **Next >** button to move to the **Select Procedures** pane. There should be none to select.

9) Click the **Next >** button to move to the **Select Models** pane. The required models will be selected automatically.

10) Click the **Next >** button to move to the **Summary** pane. Review the information present. It should look like the image below.



```
Project: ShortLivedDB v1
Project: ShortLivedXML v1
Model: BOOKS_XML v1
Model: Books v1
```

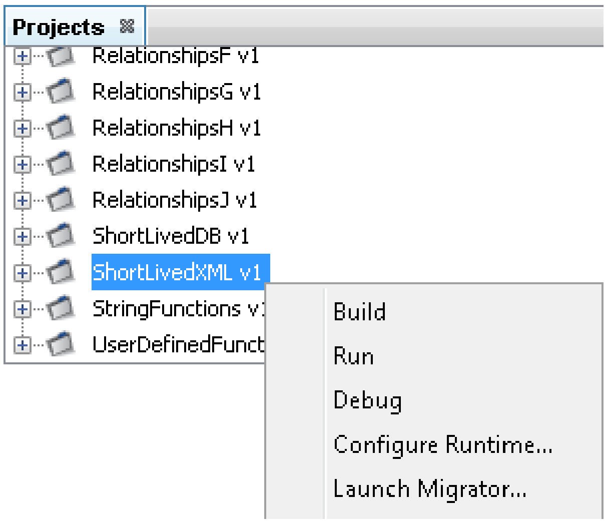
The items listed in the Summary pane may not be exactly as shown below. This is because the items listed here must not already exist in the currently open repository. You may notice that the Books v1 model is not displayed because this will probably already exist in your repository.

11) Click the **Finish** button to complete the import process. Our projects are now in the repository and we can start to create the java file.

Exercise 2 - Build Your Projects

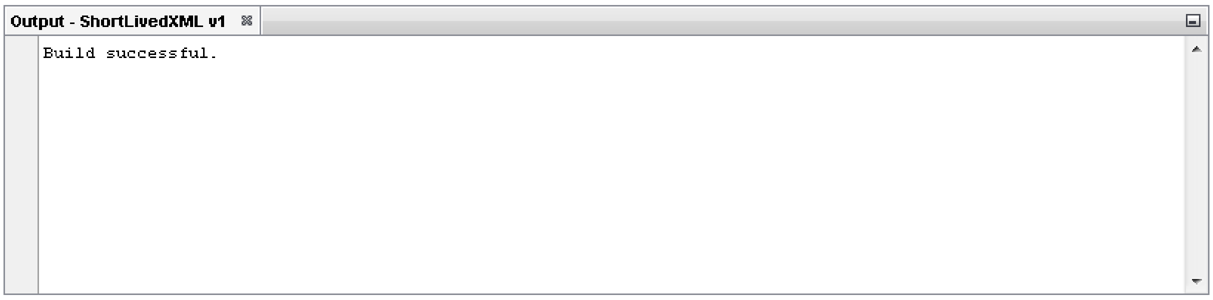
Now we must build our projects so we create the jar files for each of our projects.

1) Move your mouse pointer so that it is over the title of the project **ShortLivedXML v1** and use the right mouse or secondary mouse button to display the context menu.



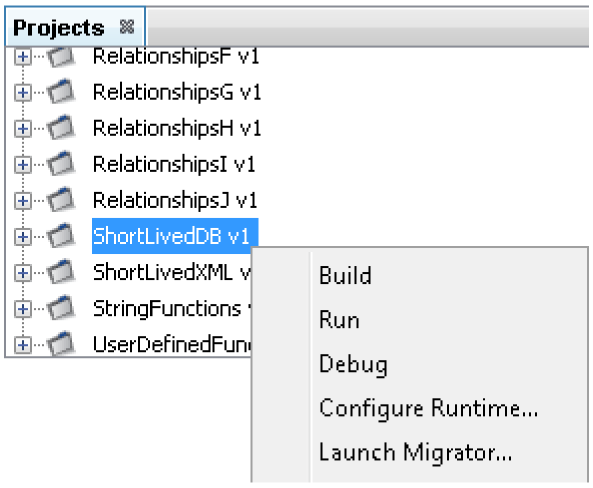
2) Click once on the Build option from the menu.

3) The **Output** pane will open and show you if there are any errors in your transform code. In this exercise you should not have any error messages and you should see a message stating **Build successful**.



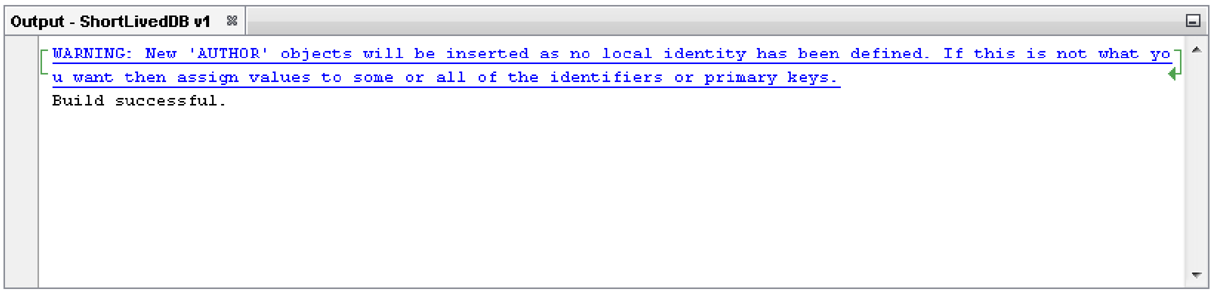
```
Build successful.
```

4) Move your mouse pointer so that it is over the title of the project **ShortLivedDB v1** and use the right mouse or secondary mouse button to display the context menu.



5) Click once on the Build option from the menu.

6) The **Output** pane will open and show you if there are any errors in your transform code. In this exercise you should not have any error messages and you should see a message stating **Build successful**.



```
WARNING: New 'AUTHOR' objects will be inserted as no local identity has been defined. If this is not what yo
u want then assign values to some or all of the identifiers or primary keys.
Build successful.
```

7) This will have created two jar files called **ETL_ShortLivedXML_1_0.jar** and **ETL_ShortLivedDB_1_0.jar** in the [TMHOME]\jar directory.

Exercise 3 - Create the File

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 shortlived;

public final class ShortLived
{
        public ShortLived()
        {
        }
}
```

Exercise 4 - Add the Main Method

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 ShortLived object. Notice that the constructor has been made private, as we are only calling it from the main method.

```
package shortlived;

public final class ShortLived
{
        public static void main(String[] args)
        {
                ShortLived aShortLived = new ShortLived();
        }

        private ShortLived()
        {
        }
}
```

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

Running the Transformation System

As running transformations in 'Short Lived' mode is straightforward, we will only require a single method that will run the transforms.

For the purpose of this example, the source will be taken from an XML file, with the data also being written to a target XML file.

Exercise 5 - Initialisation

Add the outline of the method `runProject` and configure the Transformation Manager Licence.

```
package shortlived;

import org.apache.log4j.Logger;
import net.etltm.*;

public final class ShortLived
{
        private static final String TM_LICENCE_FILE =
        "[TMHOME]\\Tutorials\\Source_and_Target\\Tutorial_18_API_Deployment_ShortDe
        ployment\\etltm.lic";

        private ShortLived()
        {
        TMTransformerFactory.setTMLicenseLocation(TM_LICENCE_FILE);
        }

        public static void main(String[] args)
        {
                ShortLived aShortLived = new ShortLived();
                aShortLived.runProject();
        }


        private void runProject()
        {
                Logger.getLogger(ShortLived.class).info("Short Lived Deployment
                Example");
        }
}
```

The code above uses Log4j to manage logging. The jar file for Log4j is included in the Transformation Manager installation. If Transformation Manager is installed in its default location the jar will be located in `C:\Program Files\TM Suite\libs\log4j-1.2.15.jar`. This jar must be included in the class path.

Along with creating a skeleton method that will contain the code to run the project, the constructor of the ShortLived class has been modified to setup the Transformation Manager licence. Transformation Manager licensing needs to be set before the rest of the deployment API is used.

In this tutorial the folder `[TMHOME]\Tutorials\Source_and_Target\Tutorial_18_API_Deployment_Short` will be used to store resources. If you already have Transformation Manager installed and licensed then it is possible to make a copy of the licence. The default location for the etltm.lic file is `[TMHOME]\etc\etltm.lic`. Once the etltm.lic file has been located copy etltm.lic to the directory. You may also create your own directory elsewhere for this purpose.

Exercise 6 - Create the New Transformer

This step creates a new “Short-lived” TMTransformer object. This is the main object used to carry out the transformations that have been created using TM Designer.

```
package shortlived;

import org.apache.log4j.Logger;
import net.etltm.*;

public final class ShortLived
```

```
{
        private static final String TM_LICENCE_FILE =
        "[TMHOME]\\Tutorials\\Source_and_Target\\Tutorial_18_API_Deployment_ShortDe
        ployment\\etltm.lic";
        private static final String PROJECT_NAME = "ShortLivedXML";

        private ShortLived()
        {
        TMTransformerFactory.setTMLicenseLocation(TM_LICENCE_FILE);
        }

        public static void main(String[] args)
        {
                ShortLived aShortLived = new ShortLived();
                aShortLived.runProject();
        }

        private void runProject()
        {
                Logger.getLogger(ShortLived.class).info("Short Lived Deployment
                Example");

                try {
                        //Create Short-lived TM Transformer
                        TMTransformer aTMTransformer =
                        TMTransformerFactory.getNewTMTransformer(this,
                        PROJECT_NAME, 1);

                } catch (TMException aTMException) {
                        Logger.getLogger(ShortLived.class).error("Error detected
                        when running transform", aTMException);
                }
        }
}
```

A TMTransformer is created using the same factory class that was previously used to set the Transformation Manager licensing. The `getNewTMTransformer` method takes three arguments. The first argument can be any Java object. The Java object supplied can be accessed from within your project using the built-in SML functions. In this example we are just going to supply the instance of the ShortLived object. The second argument is a String containing the name of the Transformation Manager project that will be run. In this example the project will be `ShortLivedXML`. Finally the version of the project is given.

The `getNewTMTransformer` method throws TMExceptions so the method call is wrapped with a try catch statement and any errors logged using Log4j.

Exercise 7 - Set Up the Source Adapter

The next step is to set up the source adapter. The source adapter is used to access the source data store for transformation.

```
package shortlived;

import org.apache.log4j.Logger;
import net.etltm.*;
import java.util.*;

public final class ShortLived
{
        private static final String TM_LICENCE_FILE =
        "[TMHOME]\\Tutorials\\Source_and_Target\\Tutorial_18_API_Deployment_ShortDe
        ployment\\etltm.lic";
        private static final String PROJECT_NAME = "ShortLivedXML";

        private ShortLived()
        {
        TMTransformerFactory.setTMLicenseLocation(TM_LICENCE_FILE);
        }

        /**
```

```
         * Create Source TMAdapter for supplied TMTransformer
         *
         * @param aTMTransformer - TMTransformer to use
         * @return TMAdapter created
         * @throws TMException
         */

         private TMAdapter createSourceAdapter(TMTransformer aTMTransformer) throws
         TMException {
         TMXMLAdapter aXMLSourceAdapter =
         TMBuiltInAdaptorFactory.makeNewReadTMXMLAdapter(aTMTransformer);

         Map param = new HashMap();
         param.put(TMXMLAdapter.NAME_URL,
         "[TMHOME]\\Tutorials\\Source_and_Target\\Tutorial_18_API_Deployment_Short\\
         Source\\books.xml");
         aXMLSourceAdapter.setInstanceConnectionData(param);

         return aXMLSourceAdapter;
         }

         public static void main(String[] args)
         {
                 ShortLived aShortLived = new ShortLived();
                 aShortLived.runProject();
         }

         private void runProject()
         {
                 Logger.getLogger(ShortLived.class).info("Short Lived Deployment
                 Example");

                 try {
                         //Create Short-lived TM Transformer
                         TMTransformer aTMTransformer =
                         TMTransformerFactory.getNewTMTransformer(this,
                         PROJECT_NAME, 1);

                         //Set Source
                         aTMTransformer.setSource(PROJECT_NAME,
                         createSourceAdapter(aTMTransformer));

                 } catch (TMException aTMException) {
                         Logger.getLogger(ShortLived.class).error(null,
                         aTMException);
                 }
         }
}
```

The source for the ShortLivedXML project is an XML data store so we will need to create and configure a `TMXMLAdapter`. This will be done in a new method called `createSourceAdapter`. The method takes a `TMTransformer` as an argument and returns a `TMAdapter`. A `TMXMLAdapter` source object is created using the `makeNewReadTMXMLAdapter` method on the `TMBuiltInAdaptorFactory` factory class. The method has only one argument, which is a `TMTransformer` object and returns a new `TMXMLAdapter` object. Now that a `TMXMLAdapter` has been created for the `TMTransformer` it needs to be configured. The adapter is configured using the `setInstanceConnectionData` method on the adapter object. The method takes a Map that contains the configuration options. For the `TMXMLAdapter` we only need to specify the XML file that should be used by the adapter. Let's use the source for the tutorial which is in `C:\Users\uc\TM\Tutorials\Source_and_Target\Tutorial_18_API_Deployment_Short\Source`. Now the adapter is configured the final step is to set it as the source adapter on the `TMTransformer` object. The source adapter is set using the `setSource` method on the `TMTransformer` object. The `setSource` method takes two arguments, project name and the `TMAdapter` object.

Exercise 8 - Set Up the Target Adapter

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

```
package shortlived;

import org.apache.log4j.Logger;
import net.etltm.*;
import java.util.*;

public final class ShortLived
{
        private static final String TM_LICENCE_FILE =
        "[TMHOME]\\Tutorials\\Source_and_Target\\Tutorial_18_API_Deployment_ShortDe
        ployment\\etltm.lic";
        private static final String PROJECT_NAME = "ShortLivedXML";

        private ShortLived()
        {
        TMTransformerFactory.setTMLicenseLocation(TM_LICENCE_FILE);
        }

        /**
        * Create Source TMAdapter for supplied TMTransformer
        *
        * @param aTMTransformer - TMTransformer to use
        * @return TMAdapter created
        * @throws TMException
        */
        private TMAdapter createSourceAdapter(TMTransformer aTMTransformer) throws
        TMException {
                TMXMLAdapter aXMLSourceAdapter =
                TMBuiltInAdaptorFactory.makeNewReadTMXMLAdapter(aTMTransformer);

                Map param = new HashMap();
                param.put(TMXMLAdapter.NAME_URL,
                "[TMHOME]\\Tutorials\\Source_and_Target\\Tutorial_18_API_Deployment
                _Short\\Source\\books.xml");
                aXMLSourceAdapter.setInstanceConnectionData(param);
                return aXMLSourceAdapter;
                }
        /**
        * Create Target TMAdapter for supplied TMTransformer
        *
        * @param aTMTransformer- TMTransformer to use
        * @return TMAdapter created
        * @throws TMException
        */
        private TMAdapter createTargetAdapter(TMTransformer aTMTransformer) throws
        TMException {
                TMXMLAdapter aXMLTargetAdapter =
                TMBuiltInAdaptorFactory.makeNewWriteTMXMLAdapter(aTMTransformer);

                Map param = new HashMap();
                param.put(TMXMLAdapter.NAME_URL,
                "[TMHOME]\\Tutorials\\Source_and_Target\\Tutorial_18_API_Deployment
                _Short\\Target\\target.xml");
                aXMLTargetAdapter.setInstanceConnectionData(param);

                return aXMLTargetAdapter;
        }

        public static public void main(String[] args)
        {
                ShortLived aShortLived = new ShortLived();
                aShortLived.runProject();
        }


        private void runProject()
        {
                Logger.getLogger(ShortLived.class).info("Short Lived Deployment
                Example");
                try {
                        //Create Short-lived TM Transformer
                        TMTransformer aTMTransformer =
                        TMTransformerFactory.getNewTMTransformer(this,
                        PROJECT_NAME, 1);

                        //Set Source
                        aTMTransformer.setSource(PROJECT_NAME,
                        createSourceAdapter(aTMTransformer));

                        // Set Target
                        aTMTransformer.setTarget(PROJECT_NAME,
                        createTargetAdapter(aTMTransformer));

                } catch (TMException aTMException) {
                        Logger.getLogger(ShortLived.class).error(null,
                        aTMException);
                }
        }
}
```

The ShortLivedXML project uses an XML file for the target. Creating and configuring the target adapter will be done in a new method, `createTargetAdapter`. Like the `createSourceAdapter` it takes a `TMTransformer` object as an argument and returns a `TMAdapter`. The first difference is that the `makeNewWriteTMXMLAdapter` method is used on the `TMBuiltInAdaptorFactory` factory class, instead of the `makeNewReadTMXMLAdapter` used by the `createSourceAdapter` method. The other difference is that the value in the transform used to configure the adapter points to a different file.

Now that the target adapter has been created and configured we need to set it as the source adapter on the TMTransformer object. The target adapter is set using the `setTarget` method on the `TMTransformer` object. The `setTarget` method takes two arguments, project name and the `TMAdapter` object.

Exercise 9 - Initialise the data in TMTransformer

Now the Source and Target Adapters have been created and the source and target xml files have been specified to the respective adapters, the TMTransformer object can open the required information.

```
package shortlived;

import org.apache.log4j.Logger;
import net.etltm.*;
import java.util.*;

public final class ShortLived
{
        private static final String TM_LICENCE_FILE =
        "[TMHOME]\\Tutorials\\Source_and_Target\\Tutorial_18_API_Deployment_ShortDe
        ployment\\etltm.lic";
        private static final String PROJECT_NAME = "ShortLivedXML";

        private ShortLived()
        {
        TMTransformerFactory.setTMLicenseLocation(TM_LICENCE_FILE);
        }

        /**
        * Create Source TMAdapter for supplied TMTransformer
        *
        * @param aTMTransformer - TMTransformer to use
        * @return TMAdapter created
        * @throws TMException
        */

        private TMAdapter createSourceAdapter(TMTransformer aTMTransformer) throws
        TMException {
                TMXMLAdapter aXMLSourceAdapter =
                TMBuiltInAdaptorFactory.makeNewReadTMXMLAdapter(aTMTransformer);
                Map param = new HashMap();
                param.put(TMXMLAdapter.NAME_URL,
                "[TMHOME]\\Tutorials\\Source_and_Target\\Tutorial_18_API_Deployment
                _Short\\Source\\books.xml");
                aXMLSourceAdapter.setInstanceConnectionData(param);

                return aXMLSourceAdapter;
        }

        /**
        * Create Target TMAdapter for supplied TMTransformer
        *
        * @param aTMTransformer- TMTransformer to use
        * @return TMAdapter created
        * @throws TMException
        */
        private TMAdapter createTargetAdapter(TMTransformer aTMTransformer) throws
        TMException {
                TMXMLAdapter aXMLTargetAdapter =
                TMBuiltInAdaptorFactory.makeNewWriteTMXMLAdapter(aTMTransformer);

                Map param = new HashMap();
                param.put(TMXMLAdapter.NAME_URL,
                "[TMHOME]\\Tutorials\Source_and_Target\\Tutorial_18_API_Deployment_
                Short\\Target\\target.xml");
                aXMLTargetAdapter.setInstanceConnectionData(param);

                return aXMLTargetAdapter;
        }

        public static public void main(String[] args)
        {
                ShortLived aShortLived = new ShortLived();
                aShortLived.runProject();
        }

        private void runProject()
        {
                Logger.getLogger(ShortLived.class).info("Short Lived Deployment
                Example");

                try {
                        //Create Short-lived TM Transformer
                        TMTransformer aTMTransformer =
                        TMTransformerFactory.getNewTMTransformer(this,
                        PROJECT_NAME, 1);

                        //Set Source
                        aTMTransformer.setSource(PROJECT_NAME,
                        createSourceAdapter(aTMTransformer));

                        // Set Target
                        aTMTransformer.setTarget(PROJECT_NAME,
                        createTargetAdapter(aTMTransformer));

                        //Open the project information along with the source
                        //and target adapter information
                        aTMTransformer.openProjectAndResources();

                } catch (TMException aTMException) {
                        Logger.getLogger(ShortLived.class).error(null,
                        aTMException);
                }
        }
}
```

This is simply done in a single line that opens the transformation project information and the source and target file information.

Exercise 10 - Run the Transformation Project

The final line to be added executes the project and its transformations taking the data from the source data store and writing it to the target data store following the business logic defined using TM Designer.

```
package shortlived;

import org.apache.log4j.Logger;
import net.etltm.*;
import java.util.*;

public final class ShortLived
{
        private static final String TM_LICENCE_FILE =
        "[TMHOME]\\Tutorials\\Source_and_Target\\Tutorial_18_API_Deployment_ShortDe
        ployment\\etltm.lic";
        private static final String PROJECT_NAME = "ShortLivedXML";

        private ShortLived()
        {
        TMTransformerFactory.setTMLicenseLocation(TM_LICENCE_FILE);
        }

        /**
        * Create Source TMAdapter for supplied TMTransformer
        *
        * @param aTMTransformer - TMTransformer to use
        * @return TMAdapter created
        * @throws TMException
        */

        private TMAdapter createSourceAdapter(TMTransformer aTMTransformer) throws
        TMException {
                TMXMLAdapter aXMLSourceAdapter =
                TMBuiltInAdaptorFactory.makeNewReadTMXMLAdapter(aTMTransformer);

                Map param = new HashMap();
                param.put(TMXMLAdapter.NAME_URL,
                "[TMHOME]\\Tutorials\\Source_and_Target\\Tutorial_18_API_Deployment
                _Short\\Source\\books.xml");
                aXMLSourceAdapter.setInstanceConnectionData(param);

                return aXMLSourceAdapter;
        }

        /**
        * Create Target TMAdapter for supplied TMTransformer
        *
        * @param aTMTransformer- TMTransformer to use
        * @return TMAdapter created
        * @throws TMException
        */
        private TMAdapter createTargetAdapter(TMTransformer aTMTransformer) throws
        TMException {
                TMXMLAdapter aXMLTargetAdapter =
                TMBuiltInAdaptorFactory.makeNewWriteTMXMLAdapter(aTMTransformer);

                Map param = new HashMap();
                param.put(TMXMLAdapter.NAME_URL,
                "[TMHOME]\Tutorials\Source_and_Target\Tutorial_18_API_Deployment_Sh
                ort\Target\\target.xml");
                aXMLTargetAdapter.setInstanceConnectionData(param);

                return aXMLTargetAdapter;
        }

        public static public void main(String[] args)
        {
                ShortLived aShortLived = new ShortLived();
                aShortLived.runProject();
        }

        private void runProject()
        {
                Logger.getLogger(ShortLived.class).info("Short Lived Deployment
                Example");

                try {
                        //Create Short-lived TM Transformer
                        TMTransformer aTMTransformer =
                        TMTransformerFactory.getNewTMTransformer(this,
                        PROJECT_NAME, 1);

                        //Set Source
                        aTMTransformer.setSource(PROJECT_NAME,
                        createSourceAdapter(aTMTransformer));

                        // Set Target
                        aTMTransformer.setTarget(PROJECT_NAME,
                        createTargetAdapter(aTMTransformer));

                        //Open the project information along with the source
                        //and target adapter information
                        aTMTransformer.openProjectAndResources();

                        //run the transform
                        aTMTransformer.runAllTransforms();

                } catch (TMException aTMException) {
                        Logger.getLogger(ShortLived.class).error(null,
                        aTMException);
                }
        }
}
```

At this point you can now open the target.xml file to see how your project has transformed the data from the books.xml source data store. In order to run the Java file created using the above steps, ensure you have the following in your classpath.

- ofetsdo .jar

Default location `C:\Program Files\TM Suite\libs`

- tmglobals.jar

Default location `C:\Program Files\TM Suite\libs`

- tmsaxon-2.4.jar

Default location `C:\Program Files\TM Suite\libs`

- tmstats.jar

Default location `C:\Program Files\TM Suite\libs`

- tmutils.jar

Default location `C:\Program Files\TM Suite\libs`

- tmvalues jar

Default location `C:\Program Files\TM Suite\libs`

- log4j jar

Default location `C:\Program Files\TM Suite\libs`

- jackson-annotations-2.0.5.jar

Default location `C:\Program Files\TM Suite\libs`

- jackson-core-2.0.5.jar

Default location `C:\Program Files\TM Suite\libs`

- jackson-databind-2.0.5.jar

Default location `C:\Program Files\TM Suite\libs`

- jackson-dataformat-smile-2.0.5.jar

Default location `C:\Program Files\TM Suite\libs`

- ETL_ShortLivedXML_1_0.jar (assuming that the project was built in Exercise 2 of this tutorial)

Default location `[TMHOME]\jar\ETL_ShortLivedXML_1_0.jar`

Exercise 11 - Database Connections

We will now modify the deployment code to use a different project called `ShortLivedDB` which has a database as the target data store.

```
package shortlived;

import org.apache.log4j.Logger;
import net.etltm.*;
import java.util.*;

public final class ShortLived
{
        private static final String TM_LICENCE_FILE =
        "[TMHOME]\\Tutorials\\Source_and_Target\\Tutorial_18_API_Deployment_ShortDe
        ployment\\etltm.lic";
        private static final String PROJECT_NAME = "ShortLivedDB";

        private Shortlived()
        {
        TMTransformerFactory.setTMLicenseLocation(TM_LICENCE_FILE);
        }

        /**
        * Create Source TMAdapter for supplied TMTransformer
        *
        * @param aTMTransformer - TMTransformer to use
        * @return TMAdapter created
        * @throws TMException
        */

        private TMAdapter createSourceAdapter(TMTransformer aTMTransformer) throws
        TMException {
                TMXMLAdapter aXMLSourceAdapter =
                TMBuiltInAdaptorFactory.makeNewReadTMXMLAdapter(aTMTransformer);

                Map param = new HashMap();
                param.put(TMXMLAdapter.NAME_URL,
                "[TMHOME]\\Tutorials\\Source_and_Target\\Tutorial_18_API_Deployment
                _Short\\Source\\books.xml");
                aXMLSourceAdapter.setInstanceConnectionData(param);

                return aXMLSourceAdapter;
        }

        /**
        * Create Target TMAdapter for supplied TMTransformer
        *
        * @param aTMTransformer- TMTransformer to use
        * @return TMAdapter created
        * @throws TMException
        */
        private TMAdapter createTargetAdapter(TMTransformer aTMTransformer) throws
        TMException {
                TMJDBCAdapter aTMJDBCAdapter =
                TMBuiltInAdaptorFactory.makeNewWriteTMJDBCAdapter(aTMTransformer);

                Map param = new HashMap();
                param.put(TMJDBCAdapter.NAME_URL,
                "jdbc:derby:[TMHOME]\\Tutorials\\Source_and_Target\\Tutorial_18_API
                _Deployment_Short\\Target\\ShortLivedDB\\books");
                param.put(TMJDBCAdapter.NAME_DRIVER,
                "org.apache.derby.jdbc.EmbeddedDriver");
                aTMJDBCAdapter.setInstanceConnectionData(param);

                return aTMJDBCAdapter;
        }

        public static public void main(String[] args)
        {
                ShortLived aShortLived = new ShortLived();
                aShortLived.runProject();
        }

        private void runProject()
        {
                Logger.getLogger(ShortLived.class).info("Short Lived Deployment
                Example");

                try {
                        //Create Short-lived TM Transformer
                        TMTransformer aTMTransformer =
                        TMTransformerFactory.getNewTMTransformer(this,
                        PROJECT_NAME, 1);

                        //Set Source
                        aTMTransformer.setSource(PROJECT_NAME,
                        createSourceAdapter(aTMTransformer));

                        // Set Target
                        aTMTransformer.setTarget(PROJECT_NAME,
                        createTargetAdapter(aTMTransformer));

                        //Open the project information along with the source
                        //and target adapter information
                        aTMTransformer.openProjectAndResources();

                        //run the transform
                        aTMTransformer.runAllTransforms();

                } catch (TMException aTMException) {
                        Logger.getLogger(ShortLived.class).error(null,
                        aTMException);
                }
        }
}
```

As the creation and configuration of the target adapter is in its own method the changes to the target adapter are limited to that method. The first difference is that we use the `makeNewWriteTMJDBCAdapter` method on the `TMBuiltInAdaptorFactory` factory class. As the other methods we have used on the `TMBuiltInAdaptorFactory` factory class it takes a `TMTransformer` object as an argument and returns the adapter object.

Since the new target adapter is a database adapter the configuration option needs to be changed. The configuration now contains the JDBC settings required to connect to the database. The database we will use is in the following directory.

```
        [TM]\Tutorials\Source_and_Target\Tutorial_18_API_Deployment_Short\Target\Sh
        ortLivedDB\Books
```

We also need to change the constant that contains the project name from `ShortLivedXML` to `ShortLivedDB`.

After you have executed the transform you will be able to see how your data has been transformed between the books.xml source data store and the books target data store which is a Derby database. You can view the database using TM Designer or an external database query tool.

In order to run the updated Java file created using the above step the following jars need to be on the class path.

- ofetsdo .jar

Default location `C:\Program Files\TM Suite\libs`

- tmglobals.jar

Default location `C:\Program Files\TM Suite\libs`

- tmsaxon-2.4.jar

Default location `C:\Program Files\TM Suite\libs`

- tmstats.jar

Default location `C:\Program Files\TM Suite\libs`

- tmutils.jar

Default location `C:\Program Files\TM Suite\libs`

- tmvalues jar

Default location `C:\Program Files\TM Suite\libs`

- log4j jar

Default location `C:\Program Files\TM Suite\libs`

- jackson-annotations-2.0.5.jar

Default location `C:\Program Files\TM Suite\libs`

- jackson-core-2.0.5.jar

Default location `C:\Program Files\TM Suite\libs`

- jackson-databind-2.0.5.jar

Default location `C:\Program Files\TM Suite\libs`

- jackson-dataformat-smile-2.0.5.jar

Default location `C:\Program Files\TM Suite\libs`

- derby-10.4.2.0.jar

Default location `C:\Program Files\TM Suite\libs`

- ETL_ShortLivedDB_1_0.jar (assuming that the project was built in Exercise 2 of this tutorial)

Default location `[TMHOME]\jar\ETL_ShortLivedDB_1_0.jar`