



Transformer's Command Line Tools

*** CONFIDENTIAL ***

Produced by: Trace Financial Limited
224-232 St. John Street
London
EC1V 4QR

Version : 1.6
Date: 15/07/14



Trace





NO WARRANTIES OF ANY NATURE ARE IMPLIED OR EXTENDED BY THIS DOCUMENTATION.

Products and related material disclosed herein are only furnished pursuant and subject to the terms and conditions of a duly executed Contract or Agreement to license Software.

The only warranties made by Trace Financial Limited, if any, with respect to the products described in this document are set forth in such Contract or Agreement.

Trace Financial Limited cannot accept any financial or other responsibility that may result from use of the information herein or the associated software material, including direct, indirect, special or consequential damages.

You should be careful to ensure that this information and/or the associated software material complies with the laws and regulations of the jurisdictions with respect to which it is used.

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

CONFIDENTIALITY

The information contained within this document is confidential and unauthorised copying or reproduction by any means is prohibited.

OWNERSHIP

Trace Financial Limited reserves title to, and all copyright and other intellectual property rights in, the information contained herein and in the associated software material.

Correspondence regarding this publication should be addressed to Trace Financial Limited, 224-232, St. John Street, London EC1V 4QR.

Copyright © 2014 Trace Financial Limited



Contents

1.	Introduction	1
1.1.	Classpath	1
1.2.	Preferences	1
1.3.	ActionRequest or CmdLineAction	1
2.	Accept Test Results	2
3.	Build Library	3
4.	Build Service	4
5.	Build	6
6.	Check Files	6
7.	External Object Import	7
8.	Schema Export	8
9.	Schema Import	8
10.	Test Files	9
11.	Publish	10

1. Introduction

Transformer has an extensive API that allows a programmer to call mappings, validation rules etc. at runtime. The details of the API can be found in the “Transformer API Developer’s guide” which is distributed as a part of the Transformer release. That document focusses on the code required to call a mapping but there is another aspect of the API. Users of the Transformer GUI often wish to perform repetitive tasks such as the building of projects, the running of tests or the checking of files from scripts or as part of their continuous build process. To facilitate this we have provided a series of command line tools that allow a user to call these actions from a Java runtime environment.

1.1. Classpath

To call the command line tools a number of jar files must be added to the Java classpath. As well as adding the transformer-runtime-complete and currency jars to the classpath the programmer must also add the transformer-designtime jar. The name of the additional jar file takes the form:

```
transformer-designtime-<release number>.jar
```

and the design time jar can be found in:

```
C:\Program Files\Trace Financial Ltd\Transformer\<release number>\lib
```

1.2. Preferences

Transformer Projects can refer to libraries using relative paths. If the command line actions are being run on a machine which does not have the Transformer GUI installed, relative paths can now be resolved if a settings file is configured on that build machine.

The settings file must be called “settings.xml” and must be placed in a subdirectory within the home directory of the user account that is running the build, e.g.,:

```
/home/TRACEPLC/fredb/.transformer/3.4/settings.xml
```

The file syntax is like this:

```
<?xml version="1.0"?>
<Settings>
  <Setting name="libraries_directory" value="/some/path/on/my/machine"/>
  <Setting name="projects_directory" value="/some_other/path/on/my/machine"/>
  <Setting name="builds_directory" value="/yet_another/path/on/my/machine"/>
</Settings>
```

Note that it doesn’t matter whether the relative paths specified within the project contain forward or backslashes, the resolution will work anyway.

1.3. ActionRequest or CmdLineAction

Many of the actions that a user can invoke in the Transformer GUI can be called from the command line. There are two levels to how the actions are exposed and these lend themselves to different calling styles. The lowest level is a series of ActionRequest classes. Each action request represents and corresponds to a single command that the user can perform in the Transformer GUI. To invoke a request the user instantiates an object of the appropriate class, calls some set methods to set various options and then calls the process method.

To simplify this further we have also created some command line action classes which wrap up the ActionRequests and provide a main method. The various properties are set via a series of command line flags. The following table lists all of the available actions and their associated ActionRequests and CmdLineActions.

Action	ActionRequest	CmdLineAction
Accept Test Results	ReqSetExpectedToOutput	AcceptResults
Build Library	ReqBuildLibrary *	BuildLibrary
Build Service	ReqBuildApplicationEntryPoint	BuildExposedService
Build	ReqBuildAll	Build
Check Files	ReqCheckFiles	CheckFiles
External Object Import	ReqExternalObjectImport	ExternalObjectImport
Schema Export	ReqSchemaExporter	SchemaExport
Schema Import	ReqSchemaImporter	SchemaImport
Test Files	ReqTestFiles	TestFiles
Publish	ReqPublish	Publish

All of the command line tools can be found in the `com.tracegroup.transformer.tools` package.

All of the ActionRequests can be found in the `com.tracegroup.transformer.actionrequests` package which is part of the `transformer-designime-<version number>.jar` file (see section 1.1).

Each of these actions will be looked at in turn.

* Note that `ReqBuildLibrary` calls two further Action Requests - `ReqBuildProject` for Included libraries or `ReqBuildDeployedLibrary` for Deployed Libraries (see section 3 for a fuller explanation).

Many of the command line actions have common parameters that must be supplied for the action to operate successfully, these include:

Project - the location of the project file (.tpj file) on disk. E.g. `c:/Projects/NewEd/NewEd.tpj`

ConfigFileGroup - this is the Message Definition or Mapping Definition Group that you wish the command to operate over.

ServiceName - this is the name of the Service that the command is being applied to.

2. Accept Test Results

Accept test results takes the output file for each test and copies its contents to the expected results file for that test. There is an option to create an expected result file if one does not exist.

Usage: `AcceptResults -p project [-c ConfigFileGroup] [-e] [-r]`

If no Config File Group is given the whole project will be changed

-e create expected results files if they do not exist

-r do not run the tests before updating the results

So to run it correctly you could do:

```
AcceptResults -p c:/Projects/NewEd/NewEd.tpj -c MyMessages
```

The associated ActionRequest is `ReqSetExpectedToOutput` and it has the following methods to configure it:

- `setConfigFile` – if you just want to accept the result for a single config file

- setConfFileGroup – the ConfigFileGroup to operate over. If this is not specified the entire project will be changed.
- setCreateExpected – should the expected results file be created if it doesn't exist (the default is false)
- setRunTests – should the tests be run before the contents are copied
- setTpjFileName – the location of the project file

So to alter an entire project the user may call:

```
ReqSetExpectedToOutput action = new ReqSetExpectedToOutput("Set Expected");
action.setTpjFileName("c:/projectDir/Project57.tpj");
action.setCreateExpected(true);
action.prepare();
action.process(new CmdLineProgressMeter());
```

3. Build Library

Build Library collects all of the configuration files in a project and copies them to a specified Jar file. This action was previously called ArchiveFiles.

Usage: BuildLibrary -p project [-ln library name] [-pk project key] [-pv version] [-ps status] [-mtv version] [-o output_directory] [-c ConfigFileGroup] [-l] [-v] [-d service_builder]

If no Config File Group is given the whole project will be archived

-ln the name of the library (stored in the prolog)

-pk the project key used to form the jar file name and the prefix for classloading

-pv the version. If this terminates with -SNAPSHOT then the first part will be used as the version and the project status will be set to TEST.

-ps the status

-mtv the minimum version of Transformer that the archive is valid to work with, if omitted then the version will be taken current Transformer runtime.

-o the output directory for the jar.

-c the Config File Group to archive. If no Config File Group is given the whole project will be archived

-l option when specified includes files which are part of libraries in the archive.

-v option when specified produces verbose output

-d specifies that the library is to be a separately deployed jar built with a service builder

The project key, version and status can only be specified if they are not already specified on the project's Build Configuration. The output directory has to be specified if it is not set on the project's properties.

It will not be possible to build a project as an included library if the build configuration says "deployed library", or vice versa. Note that the Project will be built as a library even if the Build Option in the project's Build Configuration is set to "Don't Build".

There are two modes of operation, if the library is to be separately deployable then a service builder must be set using the -d option if this option is not taken the library will be considered to be an "Included Library".

So to run it correctly you could do:

```
BuildLibrary -p c:/temp/newProj/newProj.tpj -ln newProj -o
c:/temp/newProj/
```

The Build Library calls through to the ReqBuildLibrary action request. Depending on the mode of operation that action requests calls through to two others. To build an included library it calls ReqBuildProject or to create a deployed library it uses ReqBuildDeployedLibrary.

ReqBuildLibrary has the following methods to configure it:

- setBuildNumber – the build number which will appear in the library's prolog
- setConfFileGroup – the ConfigFileGroup to operate over. If this is not specified the entire project will be archived.
- setDescription – the description of the library that will appear in the library's prolog
- setIncludeTests – should the test files be included in the archive. The default is false.
- setMinVersionOfTransformer - the minimum version of Transformer that the archive is valid to work with
- setOutputDirectory – the output directory location
- setProcessLibraries – should libraries be included in the Jar
- setProject – set the Project to operate on directly (the user cannot call both setProject and setTpjFileName)
- setProjectJarFileName – override the name of the generated Jar file name
- setPrologLibraryName – set the name of the Library
- setPrologLibraryVersion – set the version number of the Library (if not specified on the Project's properties)
- setPrologProjectKey – the project key of the library (if not specified on the Project's properties)
- setPrologProjectStatus – set the status of the library (if not specified on the Project's properties)
- setTpjFileName - the location of the project file (the user cannot call both setProject and setTpjFileName)
- setVerboseOutput – should the action log every action.
- setServiceBuilderRef which allows the user to specify the name of the service builder to use when creating the deployed library.

ReqBuildProject and ReqBuildDeployedLibrary have similar methods available.

An example for the Build Action is given below.

```
ReqBuildLibrary action = new ReqBuildLibrary("Build Library");
action.setTpjFileName("c:/projectDir/Project57.tpj");
action.setOutputFile(new File("C:/outputDir/Project57.jar"));


//If you want to set the project key uncomment the following:
//if (_prefix != null) {
//  action.setPrologProjectKey("projKey57");
//}

action.prepare();
action.process(new CmdLineProgressMeter());
```

4. Build Service

The build service action will build a specified service. The output from the service depends on the Service Builder that is associated with the service.

Usage: BuildExposedService -p project -s serviceName [-pk project key] [-pv version] [-ps status] [-j level] [-l] [-v] [-o outputDirectory]



-pk the project key used to form the jar file name and the prefix for classloading. This can only be set if it is not set on the project
-pv the version (if not specified on the Project's properties)
-ps the status (if not specified on the Project's properties)
-j the Java language level
-l build the entire project including libraries
-v option when specified produces verbose output

So to run it correctly you could do:

```
BuildExposedService -p c:/Projects/NewEd/NewEd.tpj -s ES1
```

The associated ActionRequest is ReqBuildApplicationEntryPoint and it has the following methods to configure it:

- setBuildAll – should the entire project be built (rather than the service work out which config files it requires). Default is false.
- setExposedServiceReference – the name of the Service to Build
- setLanguageLevel – the Java language level to use for the compilation of generated code
- setOutputDirectory – where to write the output to. If this is not specified then it will use the Output Directory property specified for the service.
- setProjectJarFileName – the name of output Jar file
- setProjectJarFilePath - allows the project jar file path to be set explicitly. Normally the jar file path is formed by combining the jar file name with the output directory
- setPrologLibraryName – set the name of the resulting Jar into its Prolog
- setPrologLibraryVersion – set the version number of the Jar
- setPrologProjectKey – the project key of the library (if not specified on the Project's properties)
- setPrologProjectStatus – set the status of the library (if not specified on the Project's properties)
- setPrologPrefix – set the prefix for the service
- setTpjFileName - the location of the project file to use
- setVerboseOutput – should the action log every action.

Some sample code to call the Service Builder:

```
ReqBuildApplicationEntryPoint req = new ReqBuildApplicationEntryPoint("Build  
Service");  
req.setTpjFileName("c:/Projects/NewEd/NewEd.tpj");  
req.setExposedServiceReference("ES1");  
req.prepare();  
req.process(new CmdLineProgressMeter());
```

This will produce a jar in the default place for that service. You can override some of the service's properties though - for example if you want to change the output directory you can call this before the prepare:

```
req.setOutputDirectory(new File("c:/temp/es46"));
```

5. Build

The Build action builds the Project as a Library (if necessary) and builds all of the Exposed Services defined within the Project. Note that for the Build action a Library Jar will not be created if the Library Build Configuration Build Option is set to “Don’t Build”.

The permitted usage of the command-line options depends on whether or not there is exactly one artefact to be produced.

The command line options are:

Usage: Build -p project [-pk project key] [-pv version] [-ps status] [-o output_directory]

-pk the project key used to form the jar file name and the prefix for classloading. If there is exactly one build artefact to be produced and there is no Project Key defined in the properties, the -pk parameter must be specified. In all other circumstances it must not be specified.

-pv the version. If this terminates with -SNAPSHOT then the first part will be used as the version and the project status will be set to TEST. If there is exactly one build artefact to be produced and there is no Version defined in the properties, the -pv parameter must be specified. In all other circumstances it must not be specified.

-ps the status. If there is exactly one build artefact to be produced and there is no Status defined in the properties, the -ps parameter must be specified. In all other circumstances it must not be specified.

-o the output directory for the jar. Optional if the project has a build directory specified, mandatory if it doesn’t.

The corresponding Action Request is called ReqBuildAll. It has the following options:

- setPrologLibraryVersion – set the version number of the Jar
- setPrologProjectKey – the project key of the library (if not specified on the Project’s properties)
- setPrologProjectStatus – set the status of the library (if not specified on the Project’s properties)
- setOutputDirectory – where to write the output to. If this is not specified then it will use the Output Directory property specified for the service.

6. Check Files

The check files action validates all of the files in the specified project.

Usage: CheckFiles -p project [-c ConfigFileGroup] [-o output file] [-a] [-d] [-l] [-nf]

If no Config file group is given the whole project will be checked

If the -a option is specified all files, including those with no errors, will be listed

If the -d option is specified, all data sources will be validated

If the -l option is specified files which are part of libraries will be checked

If the -nf option is specified the command will not fail even if there are files in error

Unless the -nf option is specified if the check files action detects any files in error it will exit with a non-zero error code.

The associated ActionRequest is ReqCheckFiles and it has the following methods to configure it:

- setActionFailIfChecksFail – should a failed check be considered as an action failure

- setConfigFile – if you just want to check a single config file
- setConfigFileGroup – the ConfigFileGroup to operate over. If this is not specified the entire project will be checked.
- setOutputFile – should the output be written to a file
- setProcessLibraries – should libraries be checked
- setProject - set the Project to operate on directly (the user cannot call both setProject and setTpjFileName)
- setShowAll – all files being checked are listed
- setTpjFileName - the location of the project file (the user cannot call both setProject and setTpjFileName)
- setValidateDataSources – should data sources be validated

So to check the whole of Project the required code is:

```
ReqCheckFiles req = new ReqCheckFiles("Check Project");
req.setTpjFileName("c:/Projects/NewEd/NewEd.tpj");
req.prepare();
req.process(new CmdLineProgressMeter());
```

7. External Object Import

External Object Import creates Transformer definitions from the specified Java class. The Java class and all of its dependencies must be on the Java Classpath. The Import process relies on an External Object Manager definition - this describes how the import process will proceed i.e. it describes how a field will be detected in a given Java class, how the name of the Java class will be created in Transformer and how there various Java inbuilt types will be transformed to Transforemr BOTs. If no External Object Manager Definition is specified the Basic one will be used.

Usage: ExternalObjectImport -p project -c className [-e External Object Manager Definition]
-c defines the fully qualified class that will be imported

If the -e option is specified, then that External Object Manager Definition will be used to perform the import.

The associated ActionRequest is ReqExternalObjectImport and it has the following methods to configure it:

- setClassName – the fully qualified class to import
- setExternalObjectManagerDefinitionName – the name of the external object manager definition to use to perform the import.
- setProject - set the Project to operate on directly (the user cannot call both setProject and setTpjFileName)
- setTpjFileName - the location of the project file (the user cannot call both setProject and setTpjFileName)

So to import a class using the default External Object Manager the required code is:

```
ReqExternalObjectImport req = new ReqExternalObjectImport ("External Import");
req.setTpjFileName("c:/Projects/NewEd/NewEd.tpj");
req.setClassName("org.acme.MyClass");
req.prepare();
req.process(new CmdLineProgressMeter());
```

8. Schema Export

Perform a schema export on the specified Message Definition Group. The user can choose to export the entire group or individually specified messages.

Usage: SchemaExport -p project -m mdgName [-o outputFile] [messages...]

-m specifies the name of the Message Definition Group being exported

-o specifies the location of the Schema file being created. If no output is specified the default will be taken from the schema location defined in the Project Properties.

All the rest of the arguments are treated as being a list of the messages that are being exported. If no messages are listed then the entire Group will be exported.

The associated ActionRequest is ReqSchemaExporter and it has the following methods to configure it:

- setMDGName – the name of the Message Definition Group to Export.
- setMessageNames - a collection of message name to export
- setOutputFile – the name of the Schema file being created
- setProject - set the Project to operate on directly (the user cannot call both setProject and setTpjFileName)
- setTpjFileName - the location of the project file (the user cannot call both setProject and setTpjFileName)

To export an entire group the following code is required:

```
ReqSchemaExporter req = new ReqSchemaExporter("Schema Export");
req.setMDGName ("MyCSV");
req.setOutputFile ("c:/temp/MyCSV.xsd");
req.prepare();
req.process(new CmdLineProgressMeter());
```

9. Schema Import

Create a new Message Definition Group from a specified Schema.

Usage: SchemaImport -p project -m mdgName -s schemaFile

-m names the Message Definition Group being created

-s specifies the location of the schema file that is being imported

The associated ActionRequest is ReqSchemaImporter and it has the following methods to configure it:

- setAutoGenerate – should the name of the Message Definition Group being created be generated from the Schema's name. If this is set to true setMDGName does not need to be specified
- setMDGName – the name of the Message Definition Group being created
- setProject - set the Project to operate on directly (the user cannot call both setProject and setTpjFileName)
- setSchemaLocationsAndNames – set the names and locations of the Schemas being imported. Using this version more than one schema can be imported at a time.
- setTpjFileName - the location of the project file (the user cannot call both setProject and setTpjFileName)
- setXsdFileName – set the name of the schema file being imported

ReqSchemaImporter and can be called like so:

```
ReqSchemaImporter req = new ReqSchemaImporter ("Import");
req.setTpjFileName("c:/Projects/NewEd/NewEd.tpj");
req.setMDGName ("NewGroup");
req.setXsdFileName("c:/Schemas/Blah.xsd ");
req.prepare();
req.process(new CmdLineProgressMeter());
```

10. Test Files

Run all the test in the project or in the specified Config File Group.

Usage: TestFiles -p project[-c ConfigFileGroup] [-a] [-l] [-nf] [-o output file] [-k true|false]

If no Config file group is given the whole project will be checked

If the -a option is specified all tests, including those which succeed, will be listed

If the -l option is specified tests which are part of libraries will be run

If the -nf option is specified the command will not fail even if there are test failures

The -k option specifies whether config files should be kept open during the testing. Keeping files open will mean that the tests will run more quickly but the memory usage might be higher.

Unless the -nf option is specified if the test files action detects any failed tests it will exit with a non-zero error code.

The associated ActionRequest is ReqTestFiles and it has the following methods to configure it:

- setActionFailIfChecksFail – should a failed check be considered as an action failure
- setConfigFile – if you just want to test a single config file
- setConfigFileGroup – the ConfigFileGroup to operate over. If this is not specified the entire project will be tested.
- setDoLibraries – should libraries be tested too. The default is false
- setOutputFile – should the output be written to an output file.
- setProject - set the Project to operate on directly (the user cannot call both setProject and setTpjFileName)
- setShowAll – should the name of every file being tested be printed
- setTpjFileName - the location of the project file (the user cannot call both setProject and setTpjFileName)
- setKeepFilesOpen – should the test files be kept open during the test

To run all the tests in a project the user needs the following code:

```
ReqTestFiles action = new ReqTestFiles("Test Project");
action.setTpjFileName("c:/projectDir/Project57.tpj");
action.prepare();
action.process(new CmdLineProgressMeter());
```

11. Publish

Produce documentation for a specified project.

```
Usage: Publish -p Project
        [-c ConfigFileGroup]
        [-pb Publisher]
        [-cs Stylesheet]
        [-l]
        -o output_directory
```

-c the Config File Group to publish. If no Config File Group is given the whole project will be published

-pb the publisher to be used. If no publisher is specified 'SummaryHTML' will be used.

-cs the stylesheet to apply to the output. If no stylesheet is specified 'defaultMappingStyleSheet.css' will be used.

-l publish libraries. If not specified, libraries will not be published.

-o the output directory for the jar.

So to run it correctly you could do:

```
Publish -p c:/temp/newProj/newProj.tpj -o c:/temp/newProj/
```

The associated ReqPublish action has the following methods to configure it:

- setPublishLibraries – should libraries be published
- setPublishingProject – should the whole project be published
- setPublisherName – the name of the publisher to use (e.g. SummaryHTML)
- setCssName – the stylesheet to be used
- setOutputDirectory – the output directory location

The Publisher is created using PublisherInvocation

An example the Publish Action is given below.

```
ReqPublish action = new ReqPublish("Publish");
    // Project
action.setTpjFileName("c:/projectDir/Project57.tpj");
action.setPublishingProject(true);

    // Output Directory
String outputDir = "c:/projectDir/publishingoutput";
action.setOutputDirectory(outputDir);

    // Libraries
action.setPublishLibraries(false);

    // Schema
action.setCssName("defaultMappingStyleSheet");

action.setPublisherName("SummaryHTML");
action.prepare();
action.process(new CmdLineProgressMeter());
```