



Cross Translator

Cross Translator v1.2x Developer Manual

Copyright © 1998-2007 Etasoft Inc.

Main website <http://www.etasoft.com>

Cross Translator website <http://www.ctranslator.com>

Purpose.....	2
What are plug-ins?.....	2
Plug-in types and development.....	2
Translator Integration	3
Custom Data Sources.....	5

Purpose

This document describes how to extend Cross Translator capabilities using Java™ programming language and how to execute translator maps from your application.

What are plug-ins?

Plug-ins are external functions used in Function, Condition, Validation, OnBegin, OnCommit and OnRollback properties. They are coded in Java™ and reside in package called “plugins”. However you are not limited to the set of pre-built plug-ins. You can create your own or modify existing ones.

Plug-in types and development

There are three types of plug-ins:

1. Data manipulation plug-ins used in Function property.
2. Conditional processing used in Condition and Validation properties.
3. Processing plug-ins used at transaction level.

Translator distinguishes them based on function (method) signature. Method signature is number and type of input parameters and return parameter type.

This is example of conditional plug-in that can be used in conditional processing or validation. It simply checks if input value equals value defined in Parameters property.

```
/**
 * Check input data value is equal parameter.
 */
public Boolean isEqualTo(Integer id, String strData, String strParameters) {
    if (strParameters == null || strParameters.length() == 0)
        throw new RuntimeException("Missing input parameter");
    return new Boolean(strParameters.equals(strData));
}
```

If value that is passed to segment or element is the same as value that is in Parameters property then result is Boolean.TRUE and condition will work.

```

package plugin;

/**
 * Plug-in functions for basic validation and conditional processing.
 */
public class Check {

    public Check() {

    }

    /**
     * Check input data value is equal parameter.
     */
    public Boolean isEqualTo(Integer id, String strData, String strParameters) {
        if (strParameters == null || strParameters.length() == 0)
            throw new RuntimeException("Missing input parameter");
        return new Boolean(strParameters.equals(strData));
    }

    // [... and so on]

```

This is the same isEqualTo conditional plug-in method wrapped into class so it could be used in translator Map Editor. We are missing last closing bracket in this screen shot and it is intentional as there might be more than one plug-in method in this class.

Data manipulation plug-ins have a slightly different method signature. They return String that is basically used instead of original data that is coming via input parameter strData.

```

/**
 * Trim spaces on both sides.
 */
public String trimSpaces(Integer id, String strData, String strParameters) {
    if (strData == null || strData.length() == 0) // return null if there
        return null;
    return strData.trim();
}

```

If nothing should be changed in the output plug-in could simply return strData it received as input parameter.

When you decide to develop your own plug-in simply take source code file of one of the samples provided and change class name and method names but leave method parameters and return type unchanged. This will guarantee that translator will be able to find and load your plug-ins into Map Editor. Your plug-in class should be in the Java™ CLASSPATH when translator Map Editor or Map Runner starts.

If you use translator engine packaged in JARs, it is easy, simply add classpath to your custom class to the CLASSPATH passed into JARs. However if you use translator engine or tools packaged in EXE in Windows OS then you can set classpath using special variable CT_PLUGINS in your system environment to point to custom class. Classpath in CT_PLUGINS is appended and passed to translator engine once Windows executable starts.

Translator Integration

All class and JAR files are in "integration" subdirectory under translator installation directory. Translator is written in Java™ and can be integrated in your Java™ based applications using just a few method calls.

IMPORTANT: if ctrancmd.jar is not included in the /lib directory please request it via email to technical support. This is due to source code privacy. We want to protect our Java source code from illegal use. ctrancmd.jar is only required if you want to run translator via Java™ environment.

JavaDocs are also provided. However we recommend using them only as a reference. JavaDocs include most of packages except:

com.etasoft.trans.runtime
com.etasoft.trans.mapgui
com.etasoft.gui

We reserve a right to change API in future versions of translator but calls indicated in Java™ source code examples are not expected to change.

There are two examples on how to call translator engine from your applications:

1. [axCmdLineRun.java](#) is simplified command line runner. Exception handling is at the end of the processing.
2. [axCmdLineRun2.java](#) is a bit more complex command line runner. Exception handling is at the end of the processing but some advanced techniques are added to change map properties during runtime: change DataPath or SQL statements.
3. [axBatchRun.java](#) is actual source code of translator command line utility. It is much more complex and installs callback mechanism before it calls map execution class. Callback is used to receive processing notifications from translator engine during map execution.

Most important class is axTransactionRunner. It does all the map loading, execution and basic exception handling. This is part of axCmdLineRun:

```
try {
    String mapFileName = args[0];
    axInputParameters inputParameters = new axInputParameters();
    inputParameters.addOver(args, true);

    // run map
    axTransactionRunner runner = new axTransactionRunner(mapFileName,
        inputParameters);
    runner.runMap();

    System.out.println("COMPLETE");
}
catch (InterruptedException exp) {
    System.out.println("Translation interrupted.");
}
catch (Throwable exp2) {
    System.out.println("PROCESSING FAILED");
    System.out.println("Error: " + exp2.getMessage());
}
```

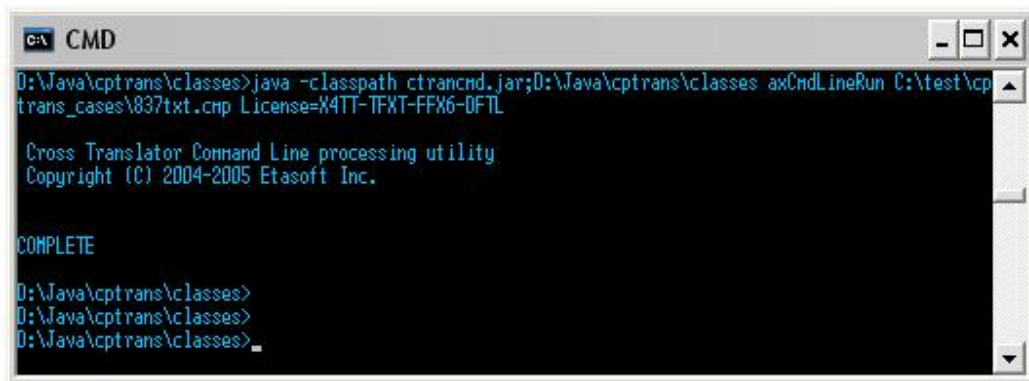
Complete source code for axCmdLineRun.java is provided with software downloads.

axTransactionRunner takes two parameters. First is map file name that should point to actual map file with extension *.cmp. Second is an instance of axInputParameters that can hold additional parameters passed to translator engine. Other constructors can also be used. However if map file name is not supplied in the constructor, then logging to log file has to be turned off. Translator uses

map file name and adds “.log” to it in order to produce log files specific for each map.

Below is a sample of execution for axCmdLineRun. You have to pass **map file name and license key** in order for it to run. Map file name will be used as parameter directly, but license key parameter will be added to axInputParameters. Other command line parameters can be passed to this utility and will be added to axInputParameters by default and passed into translator engine. Another parameter is Log and can point to log file to be used for translation logging messages. If parameter is not supplied it defaults to “log.txt”.

Classpath parameter has to contain ctrancmd.jar as well as path to location where axCmdLineRun is.



```

C:\> CMD
D:\Java\cptrans\classes>java -classpath ctrancmd.jar;D:\Java\cptrans\classes axCmdLineRun C:\test\cptrans_cases\837txt.cmp License=X4TT-TFXT-FFX6-DFTL

Cross Translator Command Line processing utility
Copyright (C) 2004-2005 Etasoft Inc.

COMPLETE

D:\Java\cptrans\classes>
D:\Java\cptrans\classes>
D:\Java\cptrans\classes>_

```

axCmdLineRun execution.

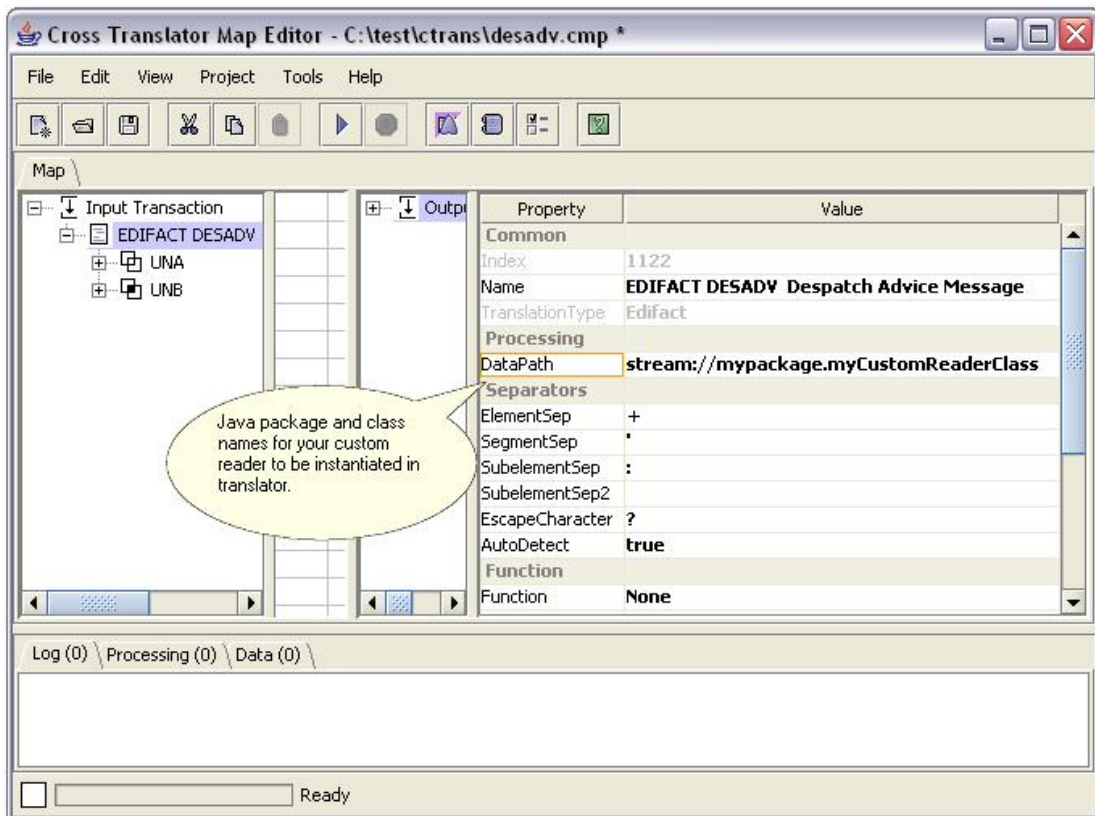
```
>java -classpath ctrancmd.jar;D:\Java\cptrans\classes axCmdLineRun
C:\test\cptrans_cases\837txt.cmp License=X4TT-TFXT-FFX6-DFTL
```

Custom Data Sources

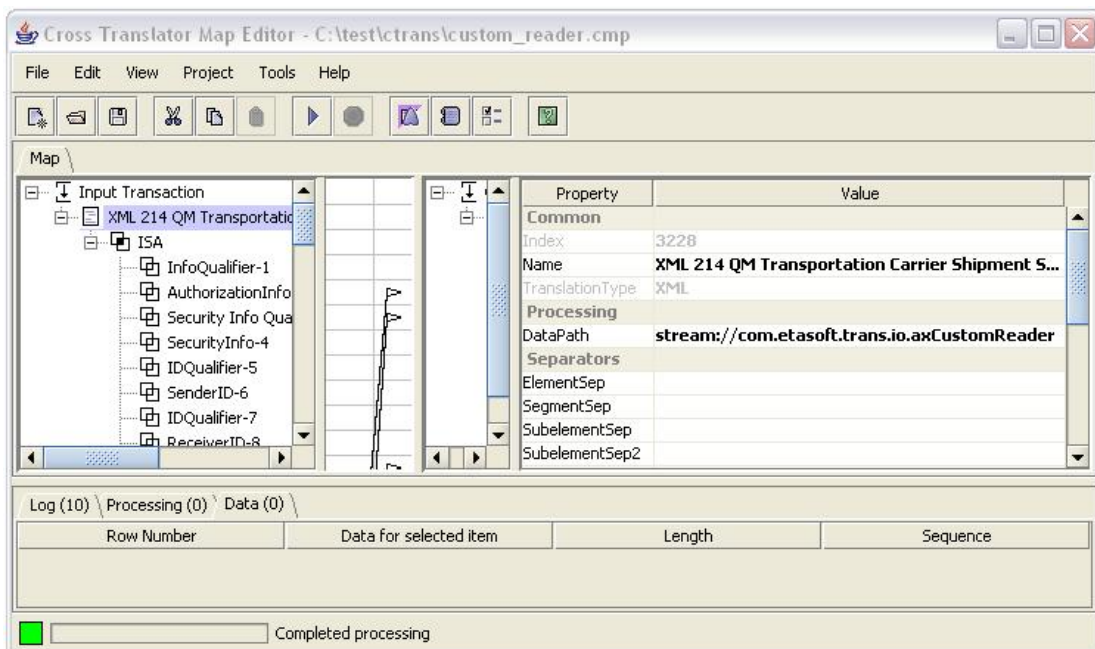
Translator supports some build-in data sources like files, FTP, HTTP and JDBC connections. It is all supported via prefix in DataPath property. Specific prefix instructs translator engine to construct different instance of class derived from axReader or axWriter.

There are cases when other data sources should be integrated in order to read data from specific data sources that do not have build-in support in translator, like message queue, web service, etc. In those rare cases custom reader or writer can be developed and integrated into translator. Integration folder contains examples of axCustomReader.java and axCustomWriter.java.

DataPath property prefix stream:// should be used to instantiate custom reader or writer. Custom reader or writer class should be in the CLASSPATH for the translator engine to find and load it. If you use translator engine packaged in JARs, it is easy, simply add classpath to your custom class to the CLASSPATH passed into JARs. However if you use translator engine or tools packaged in EXE in Windows OS then you can set classpath using special variable CT_PLUGINS in your system environment to point to custom class. Classpath in CT_PLUGINS is appended and passed to translator engine once Windows executable starts.



These are DataPath settings for custom reader integration in translator.



There is an example of using our sample com.etasoft.trans.io.axCustomReader.