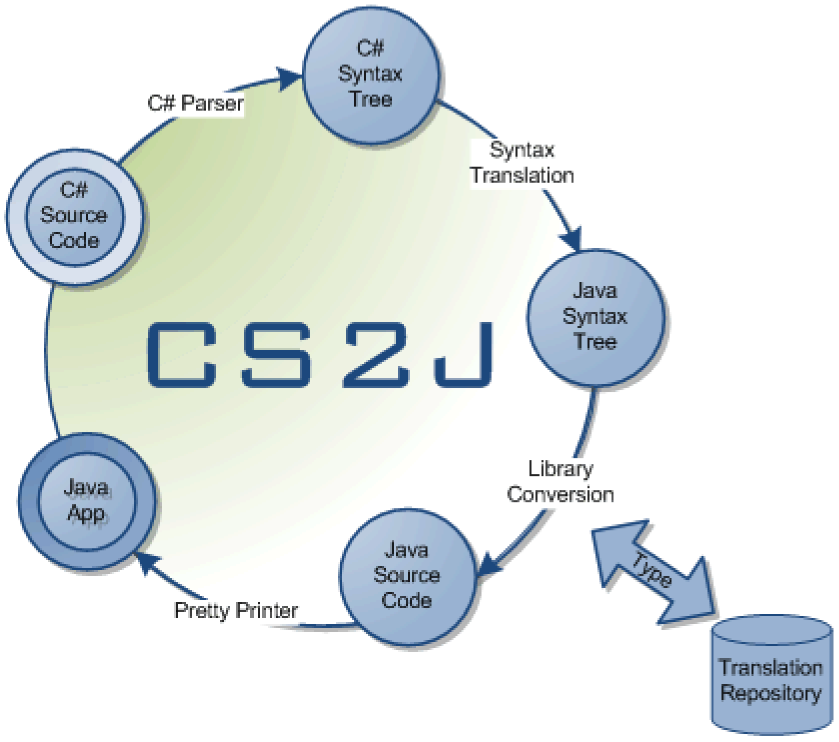
cs2j: A Developer's Guide



Trial Version

The trial version of CS2J may be used for evaluation purposes only. It has some usage restrictions compared to the full product:

1. Java classes are truncated at 150 lines.
2. The XML translation files are signed. You can make modifications, or add new translation files, and these will be used by CS2J as long as there are 5 or less translation files without valid signatures. If you hit this limit, then just restore some of the original translation files and try again.

Overview

CS2J is a C# application that converts C# classes to Java classes. This document is a brief overview of CS2J usage.

The translator first crawls over the whole of the C# application and builds up an internal data structure called the translation repository that stores translation information for all the application's classes, structs, enums, etc. It then extends this repository from XML files that add translation information for .NET System calls and third party libraries used by the application. Using this translation repository it then takes each class, struct, and enum from the application and translates it to Java:

1. Translate the C# source into a C# parse tree.
2. Translate the C# parse tree into a Java(ish) parse tree. This converts C# syntax into Java syntax, it doesn't translate method calls or do any translations that depend on types.
3. Typecheck the Java(ish) parse tree and use the translation repository to translate types and method calls into their Java equivalent.
4. Pretty print the Java parse tree to Java source files (one per top level type in the C# source file).

Running the translator

All directory paths in the following are relative to the root of the unpacked source code you received along with this release of CS2J.

To run the translator there are three main arguments:

- The directory where the manual .Net Framework translation files are held. e.g (NetFramework).
- The directory where the java classes will be written (e.g. JavaProject/src).
- The directory that is the root of the C# component that is to be translated.

CS2J is a Windows executable that can be run from the command line. (There is also a gui launcher which is not yet described in this document, ask for details). If you run it with no arguments you will get the following usage message.

To run the translator there are three main arguments:

- The directory where the manual .Net Framework translation files are held. e.g (NetFramework).
- The directory where the java classes will be written (e.g. JavaProject/src).
- The directory that is the root of the C# component that is to be translated.

CS2J is a Windows executable that can be run from the command line. (There is also a gui launcher which is not yet described in this document, ask for details). If you run it with no arguments you will get the following usage message.

```
Usage: cs2j
 [-help]                          (this usage message)
 [-v]                             (be [somewhat more] verbose, repeat for more verbosity)
 [-D <macroVariable>]             (define <macroVariable>, option can be repeated)
 [-showtokens]                    (the lexer prints the tokenized input to the console)
 [-dumpcsharp] [-dumpjavasyntax] [-dumpjava]      (show parse tree at various stages of the translation)
 [-dumpxml] [-xmldir <directory to dump xml database>]        (dump the translation repository as xml files)
 [-dumpenums <enum xml file>]                 (create an xml file documenting enums)
 [-odir <root of translated classes>]
 [-cheatdir <root of translation 'cheat' files>]
 [-netdir <root of .NET Framework Class Library translations>+]          (can be multiple directories, separated by semi-
colons)
 [-exnetdir <directories/files to be excluded from translation repository>+] (can be multiple directories/files, separated by
semi-colons)
 [-appdir <root of C# application>]
 [-exappdir <directories/files to be excluded from translation repository>+] (can be multiple directories/files, separated by
semi-colons)
 [-exclude <directories/files to be excluded from translation>+]          (can be multiple directories/files, separated by
semi-colons)
 [-translator-keep-parens <true/false>]          (keep parens from source, default true)
 [-debug <level>]                          (set debug level, default 0)
 [-debug-template-extraction <true/false>]        (show debug messages during template extraction, default true)
 [-warnings <true/false>]                      (show warnings, default true)
 <directory or file name to be translated>
```

A nearly minimal command line would be:

```
cs2jTranslator\bin\cs2j.exe -odir <java project source> -netdir NetFramework <cs application root>
```

This will translate all cs files below <application root> and place the resultant java files below <java project source>. (The directory structure of the java files will not match the directory structure of the cs files, instead it will match the java namespaces). To translate calls to the .NET framework the translator will use the translation templates found below NetFramework.

We now briefly discuss some of the other options to the translator.

Visualizing the translation

The -dumpXXXX options will show the internal data structure during processing. There are options to display the parse tree at each stage.

Dumping the translation repository

The translation database can be dumped to a set of xml files with the -dumpxml option. This produces a directory structure matching the application and .NET framework namespaces. Leaf XML files show the translation for a C# class. These translation files are discussed in more detail in the next section.

Guiding the translation process (adding Cheats)

The -cheatdir option points to a directory hierarchy that matches the target java output directory structure. You can put two types of file here:

files with extension .none: If file nothankyou.none exists in the cheats area then the translator won't produce a class file for nothankyou.
files with extension .java: If file manualisbetter.java exists in the cheats area then the translator will copy manualisbetter.java instead of its own translation.

You should consider carefully before using the cheats facility. We do not use it for our main translated product. For code that we can't translate we move it into a separate (untranslated) namespace (e.g. RusticiSoftware.Utils) and write separate versions for .NET and Java.

Excluding Paths

The -exXXX options allow you to exclude files and whole sub-trees (by giving the root directory) from consideration. You can block parts of the .NET translation area; parts of the application when generating the translation repository; and part of the source being translated. For these options you can specify multiple exclusion paths separated by semi-colons.

.NET Framework Translations

The provided .NET framework translations are in the NetFramework\System sub-folder. This is structured in the same way as the .NET framework namespace, i.e., the translation for "System.Collections.ArrayList" is the file System\Collections\ArrayList.xml. The translation file's location must match the position in the .NET namespace or the translator won't find it.

The format of the translation file deserves another document, unfortunately it isn't written (yet), so you will have to divine it from the provided translations, and asking questions.

These files are xml and the translator is a bit finicky about their structure and if it sees something it doesn't recognise it often fails silently. There is not yet a schema file to check correctness.

A useful trick when a translation doesn't seem to be picked up is to ask the translator to dump the translation database (option -dumpxml) and look at the resultant xml files to see if it recognized the translation template.