The Java Foreign Import Tool (JFit)

Edward Lam

The Java Foreign Import Tool (JFit)

Edward Lam

Last modified: November 29, 2006

Overview

This document describes how to use the Java Foreign Import Tool (JFit). JFit is a tool that allows Java classes to be imported into CAL and used in CAL source development and the Gem Cutter. This allows reuse of existing Java code when working with CAL, and eliminates the need for costly rewrites of modules that have already been designed and tested.

Using the Java Foreign Input Tool

1. Using the Java Foreign Input Factory (JFit) in the Gem Cutter

This section describes how to use the Java Foreign Import Module Factory UI to invoke JFit.

Invoke the Java Foreign Import Module Factory

From the Gem Cutter menu bar, select **Generate → Java Foreign Import Module**.

The factory dialog will be displayed, shown in *Figure 1*:

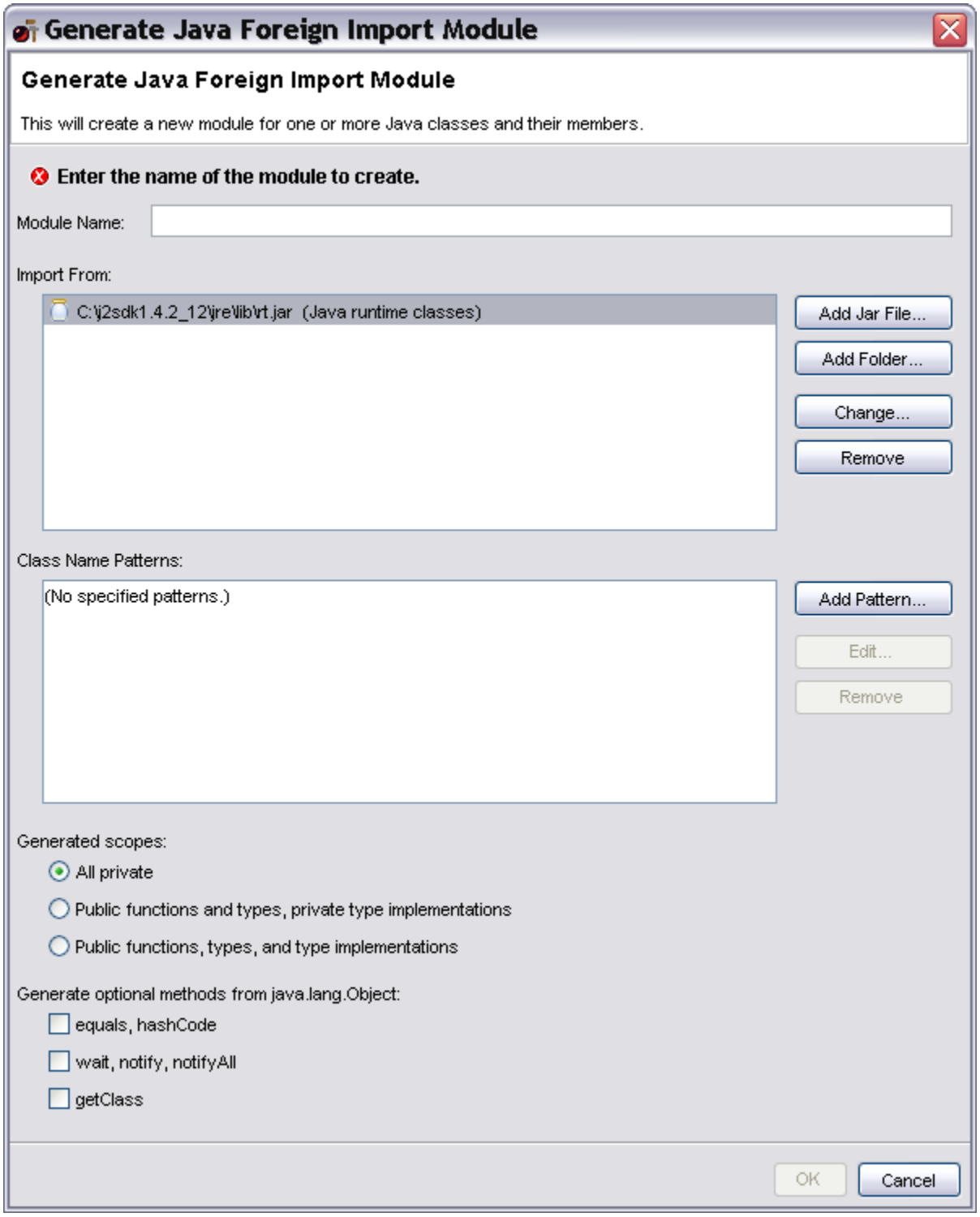

Figure 1. Generate Java Foreign Import Module dialog box

Enter the Module Name

This will be the name of the generated module. For example, if “Test” is entered here, the generated module will be module `Test`, and the file will be `Test.cal`.

Select Import Sources

In order for JFit to know the classes for which foreign imports should be generated, one or more import sources should be specified. There are two types of sources:

Jar File
: The class(es) to import exist in the selected .jar file.

Folder
: The class(es) to import are in .class file form, and this folder represents the parent of the root package component. For instance, if the files to import are subpackages of “com.businessobjects”, the folder here would be the parent of the “com” folder.

Note

The classes which comprise the standard Java distribution are available in .jar form with the Java SDK. These can typically be found in the folder: `<java.home>\lib`. For instance, the `java.lang.StringBuffer` class from a recent Java distribution might be found at a path similar to: `C:\j2sdk1.4.2_07\jre\lib\rt.jar`.

Add Import Patterns

Import patterns can optionally be provided to widen or narrow the scope of the classes of interest within the scope of the selected import sources. There are two types of import patterns:

Exclude patterns
: Classes whose fully-qualified names match the given pattern will be excluded from the classes for which types and functions will be generated.

Include patterns
: Classes whose fully-qualified names match the given pattern will be included in the classes for which types and functions will be generated, provided that their names do not match any exclude patterns.

In both types of patterns, wildcards may be used to match groups of classes. For instance:

`java.lang.StringBuffer`	matches the class `java.lang.StringBuffer`
`java.math.*`	matches all classes in all packages starting with `java.math.`
`java.lang.Illegal*Exception`	matches classes in the `java.lang` package starting with `Illegal` and ending with `Exception`
`com.businessobjects.test.classes.MyClass$?`	Matches inner classes of `com.businessobjects.test.`

	`classes.MyClass` whose unqualified names are one character long

Select Generated Scopes

This section allows the visibility of the generated types and functions to be specified. There are three options:

All private
: The visibility of all generated entities is private

Public functions and types, private type implementations
: The visibility of generated functions and types is public. The implementation scopes for the generated types will be private. This indicates that other modules can not "know" that the implementation of the type is as a foreign type, and thus cannot also define foreign functions which operate on the type.

Public functions, type, and type implementations
: The visibility of all generated entities is public.

Select Optional Methods to Generate from `java.lang.Object`

By default, functions are not generated for a number of methods inherited by all java classes from the class `java.lang.Object`. These methods are: `equals()`, `hashCode()`, `wait()`, `notify()`, `notifyAll()`, and `getClass()`.

This section can be used to override the default settings, and optionally generate functions for a number of these methods for all generated types.

Generate the Module

Click "OK" to accept the user-defined inputs and generate the foreign import module.

2. Using the Command-line Tool

This section describes how to invoke JFit from the command line.

Command-line options

-cp, -cpf
: JFit uses Java reflection to analyze selected classes, so that it may calculate the functions and types to generate for the selected classes, and their constructors, methods, and fields. In order for this to succeed, any referenced classes must exist on the classpath, either by adding to the classpath with which JFit is invoked, or adding entries via the -cp or -cpf options.

-p, -pf
: Specify class name patterns. These correspond to the **Patterns** area in the factory UI.

-xm, -xmf
: Specify methods to exclude, qualified by class name. For instance, "-xm java.lang.Exception.toString; java.lang.Exception.hashCode" will exclude the corresponding methods from `java.lang.Exception` and its subclasses.

In the factory UI, these options are available in a limited manner for some methods from `java.lang.Object`, in the **Generate optional methods** section.

-o
Specify the folder to which the.cal file will be generated. By default, the .cal file will be generated to the current folder.

-f
Specify the .jar file(s) containing the desired class(es). These correspond to the jar file import source in the factory UI. In the absence of any import source options, the tool will use the current folder as the import source.

-d
Specify the folder(s) containing the desired class(es). These correspond to the folder import source in the factory UI. In the absence of any import source options, the tool will use the current folder as the import source.

-ws
Specify the workspace which will be used as the basis for the generation. This option will affect whether the generated module will be able to reuse an existing type from an existing module. If module M contains type T, the generated module can reuse type T only if the workspace contains module M.

The default workspace is `gemcutter.default.cws`.

-public, -private, privateTypeImpl
Specify the scopes of the generated types and functions. This corresponds to the generated scopes in the factory UI.

The default visibility is “-private”.

-quiet, -verbose
These options can be used to control the amount of console output which is given by the tool.

-h
Dump a help message to the console.

