XMLVM User Manual

1 Overview

2 Invoking XMLVM

XMLVM can be invoked via the `xmlvm` command line tool. Its behavior is controlled by numerous command line arguments. `xmlvm` reads in one or more source files, processes them according to the command line options, and then writes out one or more destination files.

`--in=<path>`

> The source files are specified via one or more `--in` options. If the argument passed to `--in` is a directory, then this directory is traversed recursively and all files with the suffix `.class`, `.exe`, or `.xmlvm` are processed. Files with other suffixes are ignored. It is possible to use wildcards to filter out certain files. It is possible to specify multiple `--in` parameters. At least one `--in` parameter is required.

`--out=<path>`

> The output generated by `xmlvm` is written to a directory specified by the `--out` parameter. The argument `<path>` has to denote a directory name. If the directory does not exist, `xmlvm` will create it. All files generated by `xmlvm` will be written to this directory. The only exception is when using `--target=class`. In this case the resulting Java class files (ending in suffix `.class`) are written to appropriate sub-directories matching their package names. Already existing files with the same name will be overwritten. If the `--out` parameter is omitted, the current directory is the default.

`--target=[xmlvm|jvm|clr|dfa|class|exe|js|cpp|python|objc|iphone]`

This option defines the output format of the target. These correspond with the various backends for code generation supported by XMLVM. The different targets are explained in the following:

`xmlvm`: The input files are cross-compiled to XMLVM. `*.class` files will be cross-compiled to XMLVM$_{JVM}$. `*.exe` files will be cross-compiled to XMLVM$_{CLR}$. `*.xmlvm` files will be copied unchanged. This option is the default for `--target`.

`jvm`: The input files are cross-compiled to XMLVM$_{JVM}$.

`clr`: The input files are cross-compiled to XMLVM$_{CLR}$

`dfa`: A DFA (Data Flow Analysis) is performed on the input files. Currently the DFA will only be performed for XMLVM$_{CLR}$ programs. This option cannot be used in conjunction with any other code generating option.

`class`: The input files are cross-compiled to Java class files.

`exe`: The input files are cross-compiled to a .NET execurable.

`js`: The input files are cross-compiled to JavaScript.

`cpp`: The input files are cross-compiled to C++.

`python`: The input files are cross-compiled to Python.

`objc`: The input files are cross-compiled to Objective-C.

`iphone`: Cross-compiles an application to the iPhone. The output directory specified by `--out` will contain a ready to compile iPhone application. The resulting iPhone application can be compiled via "make" using Apple's SDK for the iPhone. This option requires the option `--iphone-app`.

`--iphone-app=<app_name>`

This option can only be used in conjunction with option `--target=iphone`. It specifies the name of the iPhone application whose name will be `<app_name>`.

`--android2iphone`

Cross-compiles an Android application to the iPhone. This option requires `--target=iphone`..

`--qx-app=<app_name>`

Cross-compiles an application to a Qooxdoo application. The environment variable `QOOXDOO_HOME` needs to point to the base directory of the Qooxdoo installation. The application will be called `<app_name>`. The output directory specified by `--out` will contain a ready to run Qooxdoo application. This option implies `--target=js` and requires option `--qx-main`.

`--qx-main=<main-class>`

This option denotes the entry point of the generated Qooxdoo application. It requires a full qualified name as a parameter. This option can only be used in conjunction with option `--qx-app`.

`--qx-debug`

Creates a debug version of the Qooxdoo application. If not specified, a ready-to-deploy version will be generated. Requires option `--qx-app`.

`--version`

Prints the version of XMLVM.

`--quiet`

No diagnostic messages are printed.

3 Examples

`xmlvm --in=/foo/bar`

The directory `/foo/bar` is searched recursively for `*.class`, `*.exe`, and `*.xmlvm` files. The default target is `xmlvm`. For `*.class` files, XMLVM_{JVM} is generated. For `*.exe` files, XMLVM_{CLR} is generated. Files with suffix `*.xmlvm` are copied to the output directory. Other files with different suffices are ignored. Since no `--out` parameter was given, the default output directory is "." (the current directory).

`xmlvm --in=/foo/*.class --in=/bar/*.exe --out=/bin`

The directory `/foo` is searched recursively for `*.class` and the directory `/bar` is searched recursively for `*.exe` files. The default target is `xmlvm`. Files with other suffices are ignored. For `*.class` files, XMLVM$_{JVM}$ is generated. For `*.exe` files, XMLVM$_{CLR}$ is generated. The resulting `*.xmlvm` files are placed in directory `/bin`.

`xmlvm --in=/foo --target=jvm`

The directory `/foo` is searched recursively for `*.class`, `*.exe`, and `*.xmlvm` files. In all cases, the generated output will always be XMLVM$_{JVM}$. For `*.exe` files as well as `*.xmlvm` files containing something other than XMLVM$_{JVM}$ will be cross-compiled XMLVM$_{JVM}$.

`xmlvm --in=/foo --target=class`

Same as the previous example, however instead of generating XMLVM$_{JVM}$ files, Java `*.class` files that can be executed by a Java virtual machine will be generated. The class files will be placed in appropriate sub-directories matching their package names.

`xmlvm --in=/foo --target=iphone --iphone-app=TheApplication`

Same as the previous example, however instead of creating Java `*.class` files, an iPhone application will be generated. The output directory will contain the ready to compile Objective-C source code including all necessary auxiliary files such as `Info.plist` and a `Makefile`. The iPhone application will be called `TheApplication` using a default icon.

`xmlvm --in=/foo --target=iphone --android2iphone --iphone-app=TheApplication`

Same as the previous example, but will also copy the Android compatibility libraries to the output directory. This effectively allows Java-based Android applications to be cross-compiled to the iPhone.

`xmlvm --in=/foo --qx-app=TheApplication --qx-main=com.acme.Main`

The directory `/foo` is searched recursively for `*.class`, `*.exe`, and `*.xmlvm` files. This option implies `--target=js`. All files will be cross-compiled to JavaScript. With the help of the Qooxdoo build scripts, the

output directory will contain a ready to be deployed AJAX application. The main entry point of the application is `com.acme.Main`.