

Soot command-line options

Patrick Lam (plam@sable.mcgill.ca)

Feng Qian (fqian@sable.mcgill.ca)

Ondřej Lhoták (olhotak@sable.mcgill.ca)

John Jorgensen

June 18, 2003

Contents

1	SYNOPSIS	1
2	DESCRIPTION	1
3	OPTIONS	2
3.1	General Options	2
3.2	Input Options	3
3.3	Output Options	4
3.4	Processing Options	4
3.5	Application Mode Options	5
3.6	Input Attribute Options	5
3.7	Annotation Options	6
3.8	Miscellaneous Options	6

1 SYNOPSIS

Soot is invoked as follows:

```
java javaOptions soot.Main [ sootOption* ] classname*
```

2 DESCRIPTION

This manual documents the command line options of the Soot bytecode compiler/optimizer tool. In essence, it tells you what you can use to replace the *sootOption* placeholder which appears in the SYNOPSIS.

The descriptions of Soot options talk about three categories of classes: argument classes, application classes, and library classes.

Argument classes are those you specify explicitly to Soot. When you use Soot's command line interface, argument classes are those classes which are either listed explicitly on the command line or found in a directory specified with the `-process-dir` option. When you use the Soot's Eclipse plug-in, argument classes are those which you selected before starting Soot from the Navigator popup menu, or all classes in the current project if you started Soot from the Project menu.

Application classes are classes that Soot analyzes, transforms, and turns into output files.

Library classes are classes which are referred to, directly or indirectly, by the application classes, but which are not themselves application classes. Soot resolves these classes and reads `.class` or `.jimple`

source files for them, but it does not perform transformations on library classes or write output files for them.

All argument classes are necessarily application classes. When Soot is not in “application mode”, argument classes are the only application classes; other classes referenced from the argument classes become library classes.

When Soot is in application mode, every class referenced from the argument classes, directly or indirectly, is also an application class, unless its package name indicates that it is part of the standard Java runtime system.

Users may fine-tune the designation of application and library classes using the Application Mode Options.

Here is a simple example to clarify things. Suppose your program consists of three class files generated from the following source:

```
// UI.java
interface UI {
    public void display(String msg);
}

// HelloWorld.java
class HelloWorld {
    public static void main(String[] arg) {
        UI ui = new TextUI();
        ui.display("Hello World");
    }
}

// TextUI.java
import java.io.*;
class TextUI implements UI {
    public void display(String msg) {
        System.out.println(msg);
    }
}
```

If you run

```
java soot.Main HelloWorld
```

HelloWorld is the only argument class and the only application class. UI and TextUI are library classes, along with `java.lang.System`, `java.lang.String`, `java.io.PrintStream`, and a host of other classes from the Java runtime system that get dragged in indirectly by the references to `String` and `System.out`.

If you run

```
java soot.Main --app HelloWorld
```

HelloWorld remains the only argument class, but the application classes include UI and TextUI as well as HelloWorld. `java.lang.System` et. al. remain library classes.

If you run

```
java soot.Main -i java. --app HelloWorld
```

HelloWorld is still the only argument class, but the set of application classes includes the referenced Java runtime classes in packages whose names start with `java.` as well as HelloWorld, UI, and `textUI`. The set of library classes includes the referenced classes from other packages in the Java runtime.

3 OPTIONS

3.1 General Options

- `-h, -help` Display the textual help message and exit immediately without further processing.
- `-pl, -phase-list` Print a list of the available phases and sub-phases, then exit.
- `-ph phase, -phase-help phase` Print a help message about the phase or sub-phase named *phase*, then exit.
To see the help message of more than one phase, specify multiple phase-help options.
- `-version` Display information about the version of Soot being run, then exit without further processing.
- `-v, -verbose` Provide detailed information about what Soot is doing as it runs.
- `-app` Run in application mode, processing all classes referenced by argument classes.
- `-w, -whole-program` Run in whole program mode, taking into consideration the whole program when performing analyses and transformations. Soot uses the Call Graph Constructor to build a call graph for the program, then applies enabled transformations in the Whole-Jimple Transformation, Whole-Jimple Optimization, and Whole-Jimple Annotation packs before applying enabled intraprocedural transformations.

Note that the Whole-Jimple Optimization pack is normally disabled (and thus not applied by whole program mode), unless you also specify the Whole Program Optimize option.
- `-debug` Print various debugging information as Soot runs, particularly from the Baf Body Phase and the Jimple Annotation Pack Phase.

3.2 Input Options

- `-cp path, -soot-class-path path, -soot-classpath path` Use *path* as the list of directories in which Soot should search for classes. *path* should be a series of directories, separated by the path separator character for your system.

If no classpath is set on the command line, but the system property `soot.class.path` has been set, Soot uses its value as the classpath.

If neither the command line nor the system properties specify a Soot classpath, Soot falls back on a default classpath consisting of the value of the system property `java.class.path` followed *java.home/lib/rt.jar*, where *java.home* stands for the contents of the system property `java.home` and `/` stands for the system file separator.
- `-process-dir dir` Add all classes found in *dir* to the set of argument classes which is analyzed and transformed by Soot. You can specify the option more than once, to add argument classes from multiple directories.

If subdirectories of *dir* contain `.class` or `.jimple` files, Soot assumes that the subdirectory names correspond to components of the classes' package names. If *dir* contains `subA/subB/MyClass.class`, for instance, then Soot assumes `MyClass` is in package `subA.subB`.
- `-src-prec format` (default value: `c`)
Sets *format* as Soot's preference for the type of source files to read when it looks for a class.
Possible values:

c, class	Try to resolve classes first from .class files found in the Soot classpath. Fall back to .jimple files only when unable to find a .class file.
J, jimple	Try to resolve classes first from .jimple files found in the Soot classpath. Fall back to .class files only when unable to find a .jimple file.

-allow-phantom-refs Allow Soot to process a class even if it cannot find all classes referenced by that class. This may cause Soot to produce incorrect results.

3.3 Output Options

-d *dir*, -output-dir *dir* (default value: **./sootOutput**)

Store output files in *dir*. *dir* may be relative to the working directory.

-f *format*, -output-format *format* (default value: **c**)

Specify the format of output files Soot should produce, if any.

Note that while the abbreviated formats (**jimp**, **shimp**, **b**, and **grimp**) are easier to read than their unabbreviated counterparts (**jimple**, **shimple**, **baf**, and **grimple**), they may contain ambiguities. Method signatures in the abbreviated formats, for instance, are not uniquely determined.

Possible values:

J, jimple	Produce .jimple files, which contain a textual form of Soot's Jimple internal representation.
j, jimp	Produce .jimp files, which contain an abbreviated form of Jimple.
S, shimple	Produce .shimple files, containing a textual form of Soot's SSA Shimple internal representation. Shimple adds Phi nodes to Jimple.
s, shimp	Produce .shimp files, which contain an abbreviated form of Shimple.
B, baf	Produce .baf files, which contain a textual form of Soot's Baf internal representation.
b	Produce .b files, which contain an abbreviated form of Baf.
G, grimple	Produce .grimple files, which contain a textual form of Soot's Grimp internal representation.
g, grimp	Produce .grimp files, which contain an abbreviated form of Grimp.
X, xml	Produce .xml files containing an annotated version of the Soot's Jimple internal representation.
n, none	Produce no output files.
jasmin	Produce .jasmin files, suitable as input to the jasmin bytecode assembler.
c, class	Produce Java .class files, executable by any Java Virtual Machine.
d, dava	Produce .java files generated by the Dava decompiler.

-xml-attributes Save in XML format a variety of tags which Soot has attached to its internal representations of the application classes. The XML file can then be read by the Soot plug-in for the Eclipse IDE,

which can display the annotations together with the program source, to aid program understanding.

3.4 Processing Options

-p *phase opt:val*, **-phase-option** *phase opt:val* Set *phase*'s run-time option named *opt* to *value*.

This is a mechanism for specifying phase-specific options to different parts of Soot. See *Soot phase options* for details about the available phases and options.

-O, **-optimize** Perform intraprocedural optimizations on the application classes.

-W, **-whole-optimize** Perform whole program optimizations on the application classes. This enables the Whole-Jimple Optimization pack as well as whole program mode and intraprocedural optimizations.

-via-grimp Convert Jimple to bytecode via the Grimp intermediate representation instead of via the Baf intermediate representation.

-via-shimple Enable Shimple, Soot's SSA representation. This generates Shimple bodies for the application classes, optionally transforms them with analyses that run on SSA form, then turns them back into Jimple for processing by the rest of Soot. For more information, see the documentation for the *shimp*, *stp*, and *sop* phases.

3.5 Application Mode Options

-i *pkg*, **-include** *pkg* Designate classes in packages whose names begin with *pkg* (e.g. `java.util.`) as application classes which should be analyzed and output. This option allows you to selectively analyze classes in some packages that Soot normally treats as library classes.

You can use the include option multiple times, to designate the classes of multiple packages as application classes.

If you specify both include and exclude options, first the classes from all excluded packages are marked as library classes, then the classes from all included packages are marked as application classes.

-x *pkg*, **-exclude** *pkg* Excludes any classes in packages whose names begin with *pkg* from the set of application classes which are analyzed and output, treating them as library classes instead. This option allows you to selectively exclude classes which would normally be treated as application classes

You can use the exclude option multiple times, to designate the classes of multiple packages as library classes.

If you specify both include and exclude options, first the classes from all excluded packages are marked as library classes, then the classes from all included packages are marked as application classes.

-dynamic-class *class* Mark *class* as a class which the application may load dynamically. Soot will read it as a library class even if it is not referenced from the argument classes. This permits whole program optimizations on programs which load classes dynamically if the set of classes that can be loaded is known at compile time.

You can use the dynamic class option multiple times to specify more than one dynamic class.

-dynamic-dir *dir* Mark all class files in *dir* as classes that may be loaded dynamically. Soot will read them as library classes even if they are not referenced from the argument classes.

You can specify more than one directory of potentially dynamic classes by specifying multiple dynamic directory options.

-dynamic-package *pkg* Marks all class files belonging to the package *pkg* or any of its subpackages as classes which the application may load dynamically. Soot will read all classes in *pkg* as library classes, even if they are not referenced by any of the argument classes.

To specify more than one dynamic package, use the dynamic package option multiple times.

3.6 Input Attribute Options

- `-keep-line-number` Preserve line number tables for class files throughout the transformations.
- `-keep-bytecode-offset`, `-keep-offset` Maintain bytecode offset tables for class files throughout the transformations.

3.7 Annotation Options

- `-annot-nullpointer` Perform a static analysis of which dereferenced pointers may have null values, and annotate class files with attributes encoding the results of the analysis. For details, see the documentation for Null Pointer Annotation and for the Array Bounds and Null Pointer Check Tag Aggregator.
- `-annot-arraybounds` Perform a static analysis of which array bounds checks may safely be eliminated and annotate output class files with attributes encoding the results of the analysis. For details, see the documentation for Array Bounds Annotation and for the Array Bounds and Null Pointer Check Tag Aggregator.
- `-annot-side-effect` Enable the generation of side-effect attributes.
- `-annot-fieldrw` Enable the generation of field read/write attributes.

3.8 Miscellaneous Options

- `-time` Report the time required to perform some of Soot's transformations.
- `-subtract-gc` Attempt to subtract time spent in garbage collection from the reports of times required for transformations.