



AGETOR®

Tool Guide

Table of contents

1	Introduction.....	1
2	Tools.....	1
2.1	Overview.....	1
2.2	jc – Compile Java files.....	1
2.2.1	Usage.....	1
2.2.2	Examples.....	2
2.3	iml2java – Compile IML servlets and components.....	2
2.3.1	Usage.....	2
2.4	idl2java – Generate and compile IDL definition files.....	3
2.4.1	Usage.....	3
2.5	version.....	4
2.5.1	Usage.....	4
2.6	props – show the configured system and AGETOR properties.....	4
2.6.1	About properties.....	4
2.6.2	Usage.....	5
2.6.3	Examples.....	5
2.7	conf – show the AGETOR configuration.....	5
2.7.1	Usage.....	5
2.7.2	Examples.....	5
2.8	proxygen – generate static proxies for entities.....	6
2.8.1	Usage.....	6
2.8.2	Examples.....	6

1 Introduction

This guide explains how to use the various tools that are included in the ADK package. The purpose and usage of each tool is explained, including the properties and command line arguments that they understand.

2 Tools

2.1 Overview

This table shows the purpose and syntax of the tools.

Name	Purpose	Syntax
jc	Compiles java files	jc <rel. file name>
iml2java	Compiles IML servlets and components	iml2java [-d<root>] [-s<sourcepath> - -sourcepath:<sourcepath>] <ihtml-file> -l<list_file> -all [ihtml-file extensions]
idl2java	Generate and compile Java class files based on an IDL definition file	idl2java <idl-file> [-d<destination>] [--nocompile]
version	Shows the current AGETOR version	version
props	Shows the configured system and AGETOR properties	props [<prefix>]
conf	Show the AGETOR configuration	conf [components datacomponents idl properties servlets storages] [<prefix>]

2.2 jc – Compile Java files

The `jc` script is used for compiling Java files in AGETOR® projects. The script will make sure that Java files are compiled using the classpath defined in the `agetor` script.

2.2.1 Usage

The syntax for `jc` is:

```
jc <rel. file name>
```

where `<rel. file name>` is the relative name of the Java source file.

By default `jc` will search for the Java source file in `AGETOR_HOME/application/java`, but this behavior can be overridden by setting the following property:

```
agetor.javacompiler.outputpath
```

`jc` will now search for the Java source file in the directory specified by this property, and it will also place the generated classes in the given directory.

2.2.2 Examples


This setting:


```
agetor.javacompiler.outputpath=C:/cvs/java
```

and this command:

```
jc com/acme/MyClass.java
```

will compile the source file `C:\cvs\java\com\acme\MyClass.java`

 Paths in property files on Windows platforms can be defined by using slashes, '/', as file separators. Alternatively, you may use two back slashes, '\\'.

 `jc` accepts both slashes and back slashes in file names.

2.3 iml2java – Compile IML servlets and components

`iml2java` compiles IML servlets and components to Java servlets and classes.

2.3.1 Usage

The syntax for `iml2java` is:

```
iml2java [-d<root>] [-s<sourcepath>|--sourcepath:<sourcepath>]
<ihtml-file>|-l<list_file>|-all [ihtml-file extensions]
```

The options and arguments are:

- `-d<root>`

`iml2java` will place the generated Java files in the given root directory.

This root directory can also be specified the following property:

```
agetor.iml2java.java.outputpath
```

The settings have these priorities:

1. Command line option (`-d`)
2. Property (`agetor.iml2java.java.outputpath`)

- `-s<sourcepath>` or `--sourcepath:<sourcepath>`

`iml2java` will look for IML source files in the given directory.

The root directory for the source files can also be specified by a property:

```
agetor.iml2java.sourcepath
```

The settings have these priorities:

1. Long option (`--sourcepath`)
 2. Short option (`-s`)
 3. Property (`agetor.iml2java.sourcepath`)
 4. Default (`AGETOR_HOME/app/ihtml`)
- `<ihtml-file>|-l<list_file>|-all`
 - `<ihtml-file>` is the name of the IML file that should be compiled
 - `<list_file>` is a file containing a list of IML files to compile
 - `-all` will make `iml2java` compile all IML files that have been configured. It will compile the files in the right order, but if you have version – show the current AGETOR version circular dependencies in the form of link tags (`servletA` links to `servletB` using the link tag, and vice versa) `iml2java` cannot handle this.
 - [`ihtml-file extensions`] Use this together with `-all` if your IML files uses some special extension.

2.4 idl2java – Generate and compile IDL definition files

`idl2java` generates and compiles Java class files based on IDL definition files.

2.4.1 Usage

The syntax for `idl2java` is:

```
idl2java <idl-file> [-d<destination>] [--nocompile] [--htmldoc] [--orbridge]
```

or

```
idl2java --all [--source:<path>] [-d<destination>] [--nocompile] [--htmldoc]
```

The options and arguments are:

- `<idl-file>`

This is the name of the IDL definition file you wish to use as input.

- `--all`

All the files in the source path will be compiled. If the source path is not defined then `${AGETOR_HOME}/app/idl` is used.

- `--source:<path>`

The source path where to search for idl files.

- `-d<destination>`

If wish to place the generated and compiled files somewhere else, you can specify the path with this option. E.g. `-d/app/java/mydir`

The destination path can also be specified by a property:

```
ageton.idl2java.java.outputpath
```

The settings have these priorities:

1. Command line option (`-d`)
2. Property (`ageton.idl2java.java.outputpath`)

- `--nocompile`

If you only wish to generate the files, not compile them, then use this option.

- `--htmldoc`

If you wish to generate the html documentation to a custom path use this option.

- `--orbridge`

Generate OpenROAD bridge for use with AGETOR® OpenROAD integration

2.5 version

version shows the AGETOR version used in the current project.

2.5.1 Usage

The syntax for *version* is:

```
version
```

2.6 props – show the configured system and AGETOR properties

props shows the system and AGETOR properties configured in the current project.

2.6.1 About properties

You can add new AGETOR properties by adding them in of the configured properties. You can also add new properties files in the properties directory.



Remember that if you have two properties with the same name then the last one will override the first one. The order is given by the order in the properties file and the order in the `conf.cfg` file.

See the ARE guide for more information about configuration.

2.6.2 Usage

The syntax for *props* is:

```
props [<prefix>]
```

The tool will return all the properties, but if a prefix is passed then only the properties with the name starting with prefix are returned.

2.6.3 Examples

```
props
```

Prints all the properties.

```
props java.class
```

Prints all the properties whose names start with java.class.

```
props | find "java" on Windows, or
```

```
props | grep java on Linux/Unix
```

Prints all the properties containing the word java.

2.7 conf – show the AGETOR configuration

conf shows the AGETOR configuration.

2.7.1 Usage

The syntax for *conf* is:

```
conf [components | datacomponents | idl | properties | servlets | storages  
[<prefix>]]
```

When the tool is invoked without arguments then it returns the files listed in the conf.cfg file. When a configuration area is passed then the tool returns the configured entries for that area.

The returned entries can be filtered by using a prefix in the same way as done for the *props* command (3rd syntax form).

2.7.2 Examples

```
conf
```

Prints all the files listed in the conf.cfg file.

```
conf properties
```

Prints all the configured properties (same as *props*).

```
conf properties java.class
```

Prints all the properties whose names start with java.class (same as *props java.class*).

2.8 proxygen – generate static proxies for entities

proxygen generates static proxies for entity components. Read more about static proxies and the entity model in the Application Developer's Guide.

The generated proxies are placed in a sub-package of the given entity interface's package, e.g., for the entity interface `shop.common.Customer`, proxygen will generate a static proxy called `shop.common.proxy.CustomerProxy`.

2.8.1 Usage

The syntax is:

```
proxygen [-k|--keep] [-r|--recursive] [-s<source path>|--sourcepath:<source path>] [-d<output path>|--outputpath:<output path>] <interface>|<package>
```

`-k|--keep`

Keep the generated Java source files (normally not necessary).

`-r|--recursive`

For directories only: Generate proxies for all entity interfaces in the given directory and all sub-directories.

`-s<source path>|--sourcepath:<source path>`

The root path of the entity interfaces.

`-d<output path>|--outputpath:<output path>`

Currently not supported.

`<interface>|<package>`

The fully qualified name of the entity interface or the package containing entity interfaces.

2.8.2 Examples

To generate a proxy for a single interface, you can use any of the forms below:


```
proxygen shop.common.Customer
proxygen shop/common/Customer
proxygen shop/common/Customer.java
proxygen shop/common/Customer.class
```

All these commands will generate the proxy for the entity interface `shop.common.Customer`. We recommend that you use one of the first two forms as the last two forms are provided as a convenience for tools using `proxygen`.

To generate proxies for several entity interfaces in a package, you can use these commands:

```
proxygen shop/common
proxygen shop.common
```

In both cases, `proxygen` will generate proxies for all the entity interfaces it finds in the package `shop.common` while all other classes and interfaces are ignored.

 Entity interfaces and packages must be designated by relative names (i.e., package name + interface name). If you wish to use absolute path names, use the source path option (see below) for that part of the name, which is not part of the Java package. E.g., to generate a proxy for the interface `shop.common.Customer` placed in `C:/project/src/shop/common/Customer.java`, use this command:

```
proxygen -sC:/project/src shop/common/Customer
```

`proxygen` can also perform a recursive search for entity interfaces:

```
proxygen -r shop/common
or:
proxygen --recursive shop/common
```

In this case, `proxygen` will search the package `shop.common` and all sub-packages for entity interfaces and create proxies for these interfaces.

By default, `proxygen` will search for interfaces in `AGETOR_HOME/app/java`. To override the default path, you can use the source path option:


```
proxygen -sC:/project/src shop/common/Customer
```

Here, `proxygen` will look for the interface `shop.common.Customer` in the directory `C:/project/src`. It is also possible to use a property to set the source path:

```
agetor.entity.proxy.sourcepath=C:/project/src
```

The command line option takes precedence over the property.

Currently, proxygen needs to be able to find the interfaces and packages in its source path, and it will generate the proxies in the source path, as the output path option (and the corresponding property) is unsupported.

 If both forms of the source path or the output path options are used in a single command, the argument of the long form will be used. In the following example, proxygen will use the source path `C:/temp`:

```
proxygen --sourcepath:C:/temp -sC:/project/src shop/common/Customer
```

As with most other AGETOR® tools, the order of the options is insignificant.