Installation Guide

Jonathan 2.0

April 12, 2002

To work through the examples supplied with Jonathan, read sections 1 and 2 of this file, and refer to section 5 if you encounter problems.

To link the Jonathan javadoc documentation with your copy of the JDK API javadoc documentation, read sections 1, 2 and 3, and refer to section 5 if you encounter problems.

To compile Jonathan source code (not normally necessary since Jonathan is delivered with its classes), read sections 1, 2 and 4, and refer to section 5 if you encounter problems.

1 Prerequisites: JDK, JNDI and a Make utility

- You will need a Java compiler (javac) and a Java runtime (java). While core Jonathan and the CORBA personality ”David” can be used with JDK 1.1 and higher, the Java RMI personality “Jeremie” can only be used with JDK 1.2 and higher.
- The Jeremie registry can be accessed using JNDI. If you are going to use this possibility, you will need the JNDI `javax.naming` package version 1.2. It is available as a jar file from:

  `http://java.sun.com/products/jndi/index.htm`
- If you are on a Unix platform, you will need GNU Make. If not already installed on your machine, it is available from:

  `http://www.gnu.org/software/make/make.html`.
- If you are on a Windows 95/98 platform, you will need a port of GNU Make to Windows. The latter is available as part of the GNU Cygwin toolset at

  `http://sourceware.cygnus.com/cygwin/`.

  Enhydra proposes a pre-installed version of Cygwin that may be used to compile Jonathan too. See `http://www.enhydra.org/DownloadCygwin.html`.
- If you are on a Windows NT platform, you will need either NMAKE, which is Microsoft’s proprietary make utility, or else the port of GNU Make to Windows.

2 Instructions for configuring Jonathan

This phase is obligatory if you want to run the examples using the provided Makefiles, or recompile Jonathan. It is done in the `config` directory.

1. You first need to compile the file `Configure.java` by typing e.g., `javac tools\Configure.java` from a DOS shell or `javac tools/Configure.java` from a Unix or Cygwin shell.

2. Do `"./configure"` from a unix shell, `"./cygwin-configure"` from a Cygwin shell, or `"configure"` from a DOS shell. This script requires that a java interpreter named ”java” is available in your path. If not, you should either add such an interpreter to your path, or modify the script you're using.

   Important: the java interpreter used must be the java interpreter you intend to use to run the examples and your applications.

   The configure utility will ask you a few questions to create a file named `"jonathan.config"`, that contains informations about the various directories, paths and parameters needed for (re)building Jonathan or running the examples.

   Amongst other things, it specifies the locations of any other classes and jar files (such as `jndi.jar`) that you want added to the classpath when doing the examples, when building Jonathan, and when regenerating the javadoc documentation. It also sets a number of other parameters depending on what you are going to do with Jonathan: for example, the version of JDK that you are using is needed for some examples.

   You may edit the `"jonathan.config"` file to change some settings. It is normally self-documenting. Note that all directories, jar files, etc., must be specified with absolute names.

   If you have problems, first look at section 5.1.

3. Configure Jonathan makefiles. This step takes into account your make utility (GNU Make or NMAKE).

   In the config directory:

   - if you are on a Unix platform, do `make` from a shell.
   - if you are on a Windows platform with GNU Make, do `make` from a cygwin shell.
   - if you are on a Windows NT platform with NMAKE, do `nmake` from a DOS shell.

   This step generates a file called `Make.rules` in the `src` and `examples` directories and recursively copies it through their subdirectories. It also generates a file called `LASTCONFIG` in the config directory, indicating the current configuration of Jonathan.

3 Instructions for (re)generating the Jonathan Javadoc documentation

This phase is necessary if you wish to link the Jonathan javadoc documentation with your copy of the JDK API javadoc documentation[1], or if you modify Jonathan source code and wish to regenerate the documentation. The following instructions except for step 1 are carried out in the `doc/src` directory.

Note that you re-generating the documentation requires javadoc 1.2.

[1]available from `http://java.sun.com/docs`

1. You must have already configured Jonathan as described in section 1.

2. Generate the documentation.

   In the doc/src directory:

   - if you are on a Unix platform, do `make` from a shell.
   - if you are on a Windows platform with GNU Make, do `make` from a cygwin shell.
   - if you are on a Windows NT platform with NMAKE, do `nmake` from a DOS shell.

   This regenerates the Jonathan javadoc documentation in the `doc/apis` directory, with links to your copy of the JDK API documentation if you specified it during the Jonathan confirguration phase.

   Due to limitations imposed by javadoc, this make is not incremental and will regenerate the documentation from scratch every time you run the make.

4 Instructions for (re)building Jonathan

This phase is necessary only if you wish to modify Jonathan source code. It is not needed if you only wish to work through the examples since Jonathan is normally delivered with its classes. Note also that re-compiling all the distribution may take quite a long time. The following instructions except for step 1 are carried out in the src directory.

1. You must have already configured Jonathan as described in section 1.

2. Recompile the source code. To recompile all the sources, do the following:

   In the src directory:

   - if you are on a Unix platform, do `make` from a shell.
   - if you are on a Windows platform with GNU Make, do `make` from a cygwin shell.
   - if you are on a Windows NT platform with NMAKE, do `nmake` from a DOS shell.

   If you are rebuilding Jonathan from scratch, the build will take a while. It is also fairly verbose; you may wish to redirect output to a file, e.g., `make > log` with GNU Make or `nmake > log` with Nmake.

To independently recompile a particular package, proceed as follows: in the `src` subdirectory corresponding to the package (or any parent `src` subdirectory), simply run the make. This will recompile only the modified source files of that package. If several packages are to be recompiled, do the make in a parent directory of those packages.

To remove generated classes, run the make with target ”clean”. This will remove classes (and most derived source files) of the current package and its subpackages.

Note that make without a target is equivalent to making the target `recall`.

5 Hints and common problems

5.1 Running the configuration tool

- Under Unix, if you can't execute "`./configure`", check that the `configure` has the appropriate execution rights.
- Under cygwin, you may need to type "`. ./cygwin-configure`" instead of "`./cygwin-configure`" to run the script.
- Under cygwin, if the script exits with a `ClassDefNotFound` exception, check that you really did "`./cygwin-configure`" and *not* "`./configure`".
- If you modify manually the `config/jonathan.config` file, the file names must be specified with absolute names, and not just names relative to the current directory. This is so that individual packages can be modified and recompiled independently of one another.

5.2 Running the Makefiles

- The source code Makefiles only take into account code generation dependencies between files where a file is generated from another file, e.g., a `.class` file is generated by compiling a `.java` file. The Makefiles do *not* take into account package dependencies, where one package imports/uses another. In this case, if you fundamentally modify a package, you may have to clean and recompile packages that import/use the modified package, depending on the kind of changes you have made.
- Depending on the version of Java that you use, the compilation of some files may produce warnings that certain APIs are deprecated. These warnings can be ignored.
- If you get messages such as `make: *** <directory name>: No such file or directory` or `the specified directory can not be found`, then the cause is likely to be a Makefile in a directory where the value of the macro `SUBDIRS` refers to a subdirectory that does not exist. You can either ignore the message or find and correct the Makefile.
- Messages such as `No rule to make <target>` or `Missing separator` indicate a problem. The cause may simply be that GNU Make is being used on an Nmake makefile, or vice-versa. Check whether you are using the right make utility for the current configuration of Jonathan (check the contents of the `LASTCONFIG` file in the config directory). If not, try reconfiguring Jonathan.

  If you are having these problems on Unix, another cause may be a makefile that was edited on a Windows platform and which may therefore contain non-printing carriage return characters which GNU Make on Unix does not like. Try editing the suspect makefile on Unix to remove such characters.
- By default, GNU Make is unnecessarily verbose in non-error cases. The following messages from GNU Make can safely be ignored:

```
make: Nothing to be done for <target>
```

```
make: Entering directory <directory name>
make: Leaving directory <directory name>
```

You can prevent them by running GNU Make with the `-s` (silent) option or by redirecting the standard output to a file.

- The Windows command `del` for removing files is also unnecesarily verbose: if a file does not exist, it will output a message to that effect. Just ignore such messages.