Wurbelizer
- User Manual -

Harald Krake

October 2007

Contents

List of Figures

1 Introduction

The WURBELIZER is an ultra lightweight code generator. It is easy to integrate, easy to learn and yet flexible and powerful. This document covers the usage and integration.
For a detailed introduction, a tutorial and examples, please visit `http://www.wurbelizer.org`.

2 Code Levels

While typical code generation describes code at two levels (generator level and generated code level), the WURBELIZER generative programming divides the problem into three levels:

1. wurblet level: holds the code for the generator itself
2. output level: the generated code
3. source level: the final source code, i.e. your application's code.

The source code of the wurblet itself contains a mix of the first two levels. The third level is parsed by the wurbler. During the wurbilation phase, the wurbler splits *your* source files into comment-, Java- and generated code. The comments, which are ignored by the Java compiler, are parsed for wurblet anchors, variables or documents. Then the wurblets are applied and the generated code inserted or replaced in the source file. The wurbler repeats these steps until there is no more change and finally writes back the source file to the filesystem (if there was any change at all).

3 Apache Ant Integration

The WURBELIZER integrates into existing development environments via Ant. Most Java-IDEs support ant out-of-the-box or even base their project management on ant, like Netbeans. Note, that it is possible to run wurblets standalone as console applications (see `DefaultWurbler.java`), however this is *not* covered by this document.

3.1 Compiling Wurblets

The wurbiler is defined as an ant task:

```
<taskdef name="wurbile" classname="org.wurbelizer.AntWurbiler">
  <classpath>
    <pathelement location="${ext}/wurbelizer.jar"/>
  </classpath>
</taskdef>
```

`${ext}` defines the directory containing `wurbelizer.jar`.
The wurbile target then looks as follows:

```
<target description="compile the wurblets" name="wurbile"
      depends="whatevertarget">
  <!-- wurbile the wurblets -->
  <mkdir dir="${wbuild}"/>
  <wurbile srcdir="${wrbl}" destdir="${wbuild}"
      includes="**/*.wrbl" />
  <!-- compile wurbiled wurblets -->
  <javac debug="on" source="1.6" deprecation="true"
    destdir="${wbuild}" includes="**" >
      <src path="${wsrc}"/>
      <src path="${wbuild}"/>
      <compilerarg value="-Xlint:unchecked"/>
      <classpath refid="wurb-cp"/>
  </javac>
</target>
```

- `${wrbl}`: directory holding the wurblet sources (filanames with extension `.wrbl`).
- `${wsrc}`: optional directory holding additional wurblet sources in Java (i.e. parent classes, interfaces, etc...)
- `${wbuild}`: directory where the wurbiled sources and compiled classes go.

Note, the `wurb-cp` classpath which defines all archives necessary to compile the wurbiled wurblets. An optional step may create a jar archive of all wurblets from `${wbuild}`.
Additional options for the wurbile task:

- `verbose` sets the verbosity (bool, default false).
- `failonerror` determines whether to stop the ant build on error (bool, default true).

Of course, all options as defined by `...ant.taskdefs.MatchingTask` can be applied as well.

3.2 Running Wurblets

The guardtype-property determines the patterns used to flag a code section as being generated according to the IDE. Currently, only `netbeans` or `other` is supported.

```
  <!-- type of guarded code blocks (depending on IDE) -->
  <property name="guardtype" value="netbeans" />
```

The wurbler is defined as an ant task as follows:

```
<taskdef name="wurbelize" classname="org.wurbelizer.AntWurbler">
  <classpath refid="wurb-cp"/>
</taskdef>
```

The classpath `wurb-cp` defines all archives necessary to run the wurblets. It must at least contain the following setup:

```
<path id="wurb-cp">
  <!-- application specific wurblets -->
  <pathelement location="${wbuild}"/>
  <!-- the wurbelizer itself -->
  <pathelement location="${ext}/wurbelizer.jar"/>
</path>
```

additional jar files can can be added as required by the wurblets.
A very basic example of an ant target to run the wurbler looks like this:

```
<target name="wurbelize" depends="wurbile"
      description="wurbelize java sources">
  <wurbelize srcdir="${src}" includes="**/*.java"/>
</target>
```

which simply applies the wurbler to all java sources.
An advanced example of the wurbler ant target may look like this:

```
<target name="wurbelize" depends="apt"
    description="wurbelize sources">

  <wurbelize printstacktrace="true" printwurblet="false"
    srcdir="${src}" includes="**/*.wurb"
    wurbletpath="org.tentackle.wurblets:de.krake.myapp.wurblets"/>

  <wurbelize printstacktrace="true" printwurblet="false"
    srcdir="${src}/de/krake/myapp/dbms"
    infoDir="${apt}" includes="*.java"
    wurbletpath="org.tentackle.wurblets:de.krake.myapp.wurblets"/>

  <wurbelize printstacktrace="true" printwurblet="false"
    srcdir="${src}/de/krake/myapp/dbms/rmi" includes="*.java"
    wurbletpath="org.tentackle.wurblets:de.krake.myapp.wurblets"/>

</target>
```

The above example splits the wurbelization into 3 steps.
The first step is an alternative to select java sources for wurbilation: the wurbiler locates files with the extension `.wurb` which are property files optionally defining additional wurblet variables. For each `.wurb`-file the wurbler assumes a corresponding `.java`-file. This is a convenient way in a large project to speed up wurbilation because the wurbiler does not need to scan all java sources, where most of them may not refer to any wurblet. Example for such a property file:

```
# this is MyClass.wurb
scriptpath=$scripts/myclass.scr
```

This declares the wurblet variable `$scriptpath` and set its value to `$scripts/myclass.scr` while `$scripts` is an ant property (or environment variable).
Steps 2 and 3 reveal some other options:

- `wurbletpath` sets the package names that the wurbler will try to load wurblets without an absolute classname. Package names are separated by colons. In the example above, the wurbler will first attempt to prepend `org.tentackle.wurblets.` and if that fails use `de.krake.myapp.wurblets.`.

- `printstacktrace` enables or disables a stacktrace for errors encountered during the execution of wurblets (boolean, default is false). This is not to be confused with exceptions thrown by the wurblets themselves!
- `printwurblet` logs the name of the wurblets processed including their filename to (bool, default false). This is useful for verifying the order in which wurblets are invoked, especially if wurblets rely on data produced by other wurblets. By default, only the names of the files that were modified are printed.
- `infodir` defines a directory containing extra data for wurblet processing, usually to exchange information between wurblets. For example, an JDK "apt" run may create information other wurblets rely on (as this is the case in tentackle).
- `runonce` determines whether wurbelizing a file is allowed more than once within an ant build (bool, default true).
- `verbose` sets the verbosity (bool, default false).
- `failonerror` determines whether to stop the ant build on error (bool, default true).

Again, all options from `org.apache.tools.ant.taskdefs.MatchingTask` apply as well.

3.3 Errors

The WURBELIZER may produce several kinds of of errors:

3.3.1 Wurbile Errors

The most common errors are unbalanced level switching within the wurblet source. In this case the wurbiler prints a error messages like these:

> SEVERE: wurblet-level closed unexpectedly on output level 'out'
> at 13:2 in MyWurblet.wrbl
>
> SEVERE: unclosed wurblet level (2) in XYWurblet.wrbl

Another typical source of errors are Java errors on the wurblet level. Note, that the line numbers printed by the Java compiler are not the line numbers of the wurblet source. Instead you have to visit the java-file generated by the wurbiler! The wurbiler does its best to produce human readable java code with comments helping associate the error line to the location in the wurblet. Example:

```
if (remote) {
  // create includes
  RemoteIncludes genInc = new RemoteIncludes(getContainer(), true);
  PrintStream implOut   = genInc.getImplStream();
  PrintStream remoteOut = genInc.getRemoteStream();
  out.print(source[8]); // 111:2 = "    if ("
  out.print(dbref);
  out.print(source[9]); // 119:10 = ".isRemote())  {      try {"
  if (contextLine != null)  {
    remoteOut.print(source[10]); // 124:2 = "  public int "
    remoteOut.print(methodName);
    remoteOut.print(source[11]); // 125:15 = "(ContextDb contextDb"
```

The comments show the line and column number of the `.wrbl`-file and the leading part of the generated output.

3.3.2 Wurbler Errors

Wurblets should throw an exception if they detect an error. The wurbler catches these exceptions and terminates the ant build process (if failOnError=true, which is the default) and prints the name of the file containing the wurblet. The stacktrace of the exception will be saved inside a comment block in the generated output, i.e. right into the file corresponding to the wurblet anchor.

```
  // @wurblet attribs Attributes .qickstartModel

// Code generated by wurblet. Do not edit!

/*
java.io.FileNotFoundException: no such heapfile '.qickstartModel'
  at org.wurbelizer.WurbUtil.openReader(WurbUtil.java:266)
  at AbstractAttributes.run(AbstractAttributes.java:55)
  at Attributes.run(Attributes.java:8)
...
```

4 Designing Wurblets

As the wurblet's source code *is* Java, you are encouraged to use all the features this elegant language provides. Especially an object oriented design helps a lot for keeping the `.wrbl` files small and readable. Remember that wurblets conserve your know how! Wurblets implement aspects and as such are *implementors*!

4.1 Wurbelizer API

All wurblets implement the interface `org.wurbelizer.Wurblet`. The implementation is generated by the wurbiler. The most important method for the wurblet to access its runtime environment is `getContainer()` which returns an instance of the interface `org.wurbelizer.Wurbler`.

4.2 Wurbler Properties

The wurbler provides a set of properties for the wurblet. Properties are categorized into namespaces as follows:

- `Wurbler.PROPSPACE_ENV ("env")`: namespace holding the environment variables
- `Wurbler.PROPSPACE_WURBLET ("wurblet")`: namespace holding properties defined by the wurbler
- `Wurbler.PROPSPACE_EXTRA ("extra")`: extra namespace. The AntWurbler will store all ant-properties here. Note, that `"ant"` is provided by the AntWurbler as an alias for `"extra"`.

The wurblet namespace will be initialized by the wurbler for each wurblet separately. If the wurbler is a SourceWurbler (this is what AntWurbler invokes), the following properties are defined:

- `WURBPROP_FILENAME ("filename")`: the absolute pathname of the source file
- `WURBPROP_CLASSNAME ("classname")`: the java classname of the source file
- `WURBPROP_WURBNAME ("wurbname")`: the pathname of the *.wurb-file (additional properties), null if none
- `WURBPROP_DIRNAME ("dirname")`: the directory of the source file
- `WURBPROP_PACKAGENAME ("packagename")`: the package name of the class

- `WURBPROP_GUARDNAME ("guardname")`: the guardname of the wurblet anchor (wurblet tag)
- `WURBPROP_WURBLETNAME ("wurbletname")`: the name of the wurblet

Example:

```
String packageName = getContainer().
      getProperty("wurblet", "packagename");
if (packageName == null)  {
  throw new Exception("can't determine <packagename>");
}
```

5 Wurblet Language

The WURBELIZER's extensions to the Java syntax are all defined with the help of the character '@'. There are two kinds of such extensions that apply to the wurblet source code:

1. switching between wurblet- and output-level
2. wurbiler directives

Special significance of a character can be turned off by preceding it with a backslash. A line terminated by a backslash is concatenated with the following line (continuation line). The backslash itself is expressed by a double backslash.

5.1 Level Switching

The wurblet source starts at the output level. Thus, if no level switching occurs, the wurblet's source will simply be copied to the generated output. Changing the level can be achieved in two ways:

1. code switching
2. value extraction

With code switching the wurblet changes the code level between the generated output and the control flow of the wurblet's execution, and vice versa. With value extraction it is possible to insert values of the wurblet level into the generated output stream.

5.1.1 Code Switching

Code switching is achieved by an @-sign and a square bracket.

- `@[` switches to wurblet level
- `]@` switches to output level

Example:

```
@[
   // generate debug message if debug == true
   if (debug) {
]@
   System.out.println("debug message ...");
@[
   }
]@
```

The debug message is only generated on the output level if debug on the wurblet level is true.

5.1.2 Value Extraction

To copy values from the wurblet level to the generated output the @-sign is used in conjunction with round braces.

- `@(` enters the wurblet level
- `)@` leaves the wurblet level

For example, the wurblet source:

```
@[
   for (int i=0; i < 3; i++) {
]@
     System.out.println("This is line @(i+1)@");
@[
   }
]@
```

Will generated the following code:

```
     System.out.println("This is line 1");
     System.out.println("This is line 2");
     System.out.println("This is line 3");
```

Any variable in Java, regardless if it is an atomic integer, an object, the return value of a method, or an expression, can be inserted into the generated output this way. Rule: if you can print it with `System.out.print()`, you can use it for value extraction.

5.2 Wurbiler Directives

The wurbiler accepts directives at compile time. Directives are enveloped by a leading `@{` and a closing `}@` and are allowed on the wurblet and the output level. They take the form:

```
@{<directive> <arg1> <arg2> ... <argN>}@
```

The following directives are supported:

5.2.1 include

`@{include <filename>}@` includes the source file given by `<filename>`. Includes may be nested to any depth. Loops are detected by the wurbiler. The filename allows $-variables that will be replaced at compile time.
Example:

```
@{include $wurblets/wrbl/processargs.inc}@
```

5.2.2 args

`@{args <arg1> ... <argN>}@` presets the wurblet arguments. The wurblet retrieves its arguments by `getContainer().getArgs()` at the wurblet level. By default, the arguments are provided by the wurbler which usually gets them from the wurblet-anchor within the source. `@{args}@` without any arguments switches back to the default, i.e. wurbler args. This is especially useful after including wurblet sources to make sure that arguments are processed.

5.2.3 to

A wurblet may generate code into more than one stream. The default stream `out` is already provided by the container and thus does not need to be explicitly opened or closed by the wurblet. Other streams, however, must be managed by the wurblet. `@{to <outputstream>}@` sets the output stream the generated code will be written to. `@{to out}` resets to the default stream, which is equivalent to `@{to}@`.
Example (from Tentackle's AppDbSelectList-wurblet):

```
@{to remoteOut}@
  public RemoteDbCursor @(methodName)@(ContextDb ....
@{to implOut}@

  public RemoteDbCursor @(methodName)@(ContextDb ...
    try {
      setContextDb(contextDb);
      return new RemoteDbCursorImpl(this, ...
    }
    catch (Exception e) {
      throw new RemoteException("remote ...
    }
  }
@{to}@
        return new @(varListType)@(getContextDb(), ...
```

5.2.4 package

`@{package <packagename>}@` sets the package name of the wurblet. By default wurblets are not created into a particular package. It is good practice, however, to assign a package to wurblets. Notice that package names must be defined in the wurbler ant task via the `wurbletpath`-option.

5.2.5 import

`@{import <importpath>}@` adds import statements to the wurblet source. Note, that `org.wurbelizer.*` is always imported.
Example:

```
@{import java.util.*}@
```

5.2.6 extends

`@{extends <parentclass>}@` sets the parent class the wurblet extends. The default is `org.wurbelizer.AbstractWurblet`. Extending an interim parent class is a common design pattern for wurblets. The interim parent class usually implements all helper code, i.e. to access the model, while the wurblet contains only wurblet code.

5.2.7 implements

`@{implements <interface1> ... <interfaceN>}@` adds interfaces the wurblet will implement. Two or more `@{interface}@`-directives will be concatenated. Example:

```
@{implements MyInterface1 MyInterface2}@
@{implements MyInterface3}@
```

will result in

```
... implements MyInterface1, MyInterface2, MyInterface3 {
```

5.2.8 indent

By default the wurbiler does its best to produce readable and debuggable code. However, the indentation can be changed manually to improve readability.

```
@{indent <columns>}@
```

will set the indentation fixed to `<columns>`. To switch back to the automatic indentation (which is the default), use `@{indent auto}@`.

6 Source Language

Java comments, whether single line or block comments, are parsed by the wurbler for wurblet anchors, variables and documents. This applies to generated comments as well! Consequently, generated code may include new anchors, variables or documents that in turn are evaluated by the wurbler and so on. This is an important feature because it allows bootstrapping from a rudimentary template similar to a software developer starting with a sketch (*you won't understand until you know how it became*).
Special significance of a character can be turned off by preceding it with a backslash. A line terminated by a backslash is concatenated with the following line (continuation line). The backslash itself is expressed by a double backslash.

6.1 Wurblet Anchor

All generated code, whether a single line or a large block, must be bound to a wurblet and optional variables or resources such as model information. The binding is done by a so-called **wurblet anchor**. There may be as many anchors in a comment block as desired. The wurblets are applied by the wurbler in the order of their anchors in the source file. The generated output is inserted into the source file right after the comment. Each output block is embraced by special comment lines that uniquely identify the block and allow the wurbler to associate it to the anchor in subsequent wurbelization passes. Once the code is generated, the anchor may be moved in the source file anywhere without losing the binding. Thus, the source file may be edited and anchors or the model information changed between wurbelization passes without getting out of sync.
An anchor is of the form:

```
@wurblet[(wurbler-directive)] <tag> <wurblet> [<arg1> [... <argN>]]
```

where:

- `@wurblet` starts an anchor
- `(wurbler-directive1 ... N)` optional directives to the wurbler
- `<tag>` is a tag uniquely identifying the anchor within the source file
- `<wurblet>` is either the classname of the wurblet. This is either an absolute classname or a basename to which the wurbler will prepend the package names provided by the `wurbletpath=...` option of the corresponding ant target.
- `<arg1>...` optional wurblet arguments. Usually, the wurblet arguments carry model information or refer to such.

Anchors that span more than one line must be concatenated by a trailing backslash at the end of the lines to be continued. Arguments are separated by whitespaces. If an argument contains whitespaces, it must be enclosed in double quotes. Such arguments may span more than one line as well. Arguments may contain $-signs denoting a variable. Variables may either be declared in a variable section, can be an ant property, a property from a wurb-file or an environment variable. The variable substitution process is recursive, i.e. variables may contain strings referring to other variables and so on.
For example:

```
/**
  * Principals are cached preloaded with a second index 'code'.
  *
  * @wurblet cache AppDbCache $mapping --preload code
  */

/**
  * Deletes all role objects belonging to either
  * object A or object B.
  *
  * @wurblet deleteById AppDbDeleteBy $mapping $remote \
      idA[id] idB:=:id \
      --sql="FIELD_IDA + \"=? OR \" + FIELD_IDB + \"=?\""
  */
```

Notice: in continuations lines make sure the IDE or editor does not insert extra asterisks or double slashes as they will become part of the arguments passed to the wurblet.

6.1.1 Unnamed Anchor

If the `<tag>` in the wurblet anchor does not start with a letter, it is considered as an *unnamed wurblet anchor*. Unnamed anchors are useful for so-called *inline wurblets*, which are commonly used to inject a small piece of code into the code block directly preceding or following the anchor. The code is injected between two consecutive *empty block comments* (`/**/`). The first character of the tag determines whether the preceding or following code block is targeted by the wurblet.

- The characters `-` or `<` refer to the preceding block
- all others, including `+` or `>` refer to the following block

Here is an example of an inline wurblet used for dependency injection at build time:

```
private Subscriber subscriber =
  new /**/MockSubscriber/**/(); // @wurblet < Inject $subscriber
```

In this example the `MockSubscriber` has been inserted by the `Inject`-wurblet. Please keep in mind that the code generated by inline wurblets is *not* protected against manual modifications within IDEs like Netbeans.

6.1.2 Conditional Anchor

Wurblets may be invoked conditionally. This is achieved by the `test:`-directive.
Example:

```
@wurblet(test:$remote==yes) myRemoteMethod GenRemote $mapping
```

Will invoke the wurblet if the variable `$remote` equals the string `yes`. Otherwise the comment `// condition not met` will be generated. Currently, the operators `==` and `!=` are supported.

6.1.3 Extra Indentation

The generated code may be indended by extra spaces with the indent wurbler directive. A negative value is treated as 0.
Example:

```
/**
  * Sets the SQL-fields in a prepared statement.
  *
  * @wurblet(indent=4) setFields DbSetFields $mapping
  */
```

6.1.4 Editor Fold

For the selected IDE an optional editor fold type for the generated code may be specified. Currently editor folds are only supported for the Netbeans-IDE. By default, no editor fold will be generated. The following fold types are supported:

- `collapsed`: the generated code will be collapsed
- `expanded`: the generated code will be expanded
- `none`: the generated code will get no fold-handle (default)

Example:

```
/**
  * Sets the SQL-fields in a prepared statement.
  *
  * @wurblet(fold=collapsed) setFields DbSetFields $mapping
  */
```

6.2 Documents

Documents are an elegant way to provide model information to the wurblets or to create files in general. Whenever the wurbler detects a document section within a comment block, it creates a file either on heap or in the filesystem depending on the file's name. Document sections start with `@>` and terminate with `@<`.
Example:

```
// @> $relations
// # relations for Principal
//
// PrincipalAddress:
//  relation=tracked composite list,
//  select=lazy, delete=static;
// @<
```

Notice that the filename (`$relations` in this example) may be a variable again. If the filename starts with a dot, it will not be saved to the filesystem but instead kept in memory only. Furthermore, such *heap files* must be unique within the wurbler instance. If the wurbler is invoked from within ant (which usually is the case), the file is available during the whole wurbelization process, i.e. ant build process, to all wurblets in all files. This feature allows running wurblets in distinct phases, each phase creating some information for the next phase.

6.3 Variable Declaration

Ant properties and environment variables are automatically declared as variables by the wurbler. Additional variables may be declared either in extra property-files (extension `.wurb`) or directly at the source level within any comment block. Such a declaration section starts with `@{` and ends with `@}`.
Example:

```
// @{
// tablename = principal
// mapping   = $scripts/$tablename.map
// relations = $scripts/$tablename.rel
// @}
```

Notice that the curly braces follow the `@`-sign. We are at the source level, not at the wurblet level!