The ForrestBot

by Marc Portier

This document explains how the Forrest project can be used to centrally build the documentation for different projects. It also explains which pieces in the Forrest distribution form this functionality, and how they could be extended in the future.

1. Foreword

This document explains how the Forrest project can be used to centrally build the documentation for different projects. It also explains which pieces in the Forrest distribution form this functionality, and how they could be extended in the future.

See some related documents:

- Our Contract - explains the rules with which your project should comply
- Forrestbot introduction - concise overview of Forrestbot

2. Using the Forrestbot

2.1. Setting up the centralized service

The goal is to publish a static website that can be updated in near-real-time to accomodate newly available content. Forrestbot can run on one machine, get content from another machine, and publish to yet another machine.

Running the bot is as easy as calling the following in your shell. (executed in the root dir of the distribution.)

```
build bot -Dbot.config=myOwn.conf.xml -Dtemplate.echo=[true|false]
```

It is quite typical to perform this from a shell script that gets scheduled for execution on a regular basis.

2.2. Describing your project to the bot

FIXME (mpo):
There is not yet a DTD for this configuration file.

The actions the bot is going to take are defined in the XML file pointed at with the `bot.config` System property. (if not specified with `-Dbot.config` then the default config from the distribution is taken: `./forrestbot.conf.xml`)

This configuration file has one (required, but optionally empty) child-element for setting cross-project defaults, followed by a list of project-elements that hold the configuration for the various projects that the bot is going to work on.

```
<?xml version="1.0"?>
<forrest-config>
  <defaults>
     ...<!-- different fallback values for various workstages -->
  </defaults>

  <project name="project-name-1">
     ...<!-- different subelements configure the various workstages -->
  </project>

  <project name="project-name-2">
     ...<!-- different subelements configure the various workstages -->
  </project>
</forrest-config>
```

During the bot-run, each project entry will start a parallel and independent process consisting of a number of so called 'workstages'.

These workstages are listed in the `./src/resources/forrestbot/stages.conf.xml` file and their actual execution is handled by the various template tasks in the `./resources/forrestbot/ant/templates.build.xml` file.

Currently the various workstages are:

Inside the project element the configuration allows to set some parameters that get consumed in the template targets. This workstage dependant configuration always takes the following layout: an element with the name of one of the tasks, and an optional type-attribute with nested elements that hold parameter names, with required name attribute holding the actual parameter value. Example:

```
<project name="xml-forrest">
    <prepare>
        <skin name="forrest-site"/>
    </prepare>

    <get-src type="cvs">
      <host name="cvs.apache.org"/>
      <root name="/home/cvspublic"/>
      <user name="anoncvs"/>
      <passwd name="anoncvs"/>
      <module name="xml-forrest"/>
      <content-dir name="src/documentation"/>
      <project-dir name=""/>
    </get-src>

    <generate>
      <debuglevel name="ERROR" />
    </generate>

    <deploy type="scp">
      <host name="krysalis.sourceforge.net"/>
      <root name="/home/groups/k/kr/krysalis/htdocs"/>
      <user name="forrestbot"/>
      <dir name="forrest"/>
    </deploy>
  </project>
```

Note:
Only the workstages that need configuration settings have an entry here. At all times there will be a workstage template called for each stage of the stages.conf.xml (even if it is not explicitly mentioned inside the <project> element.)

Explained in detail, this will

- SET names properties (workstage(.@type)?.*) to appropriate values (*/@name)

  prepare.skin
  forrest-site
  get-src.cvs.host
  cvs.apache.org
  get-src.cvs.root
  /home/cvspublic
  get-src.cvs.user
  anoncvs
  get-src.cvs.passwd
  anoncvs
  get-src.cvs.module
  xml-forrest
  get-src.cvs.content-dir
  src/documentation
  get-src.cvs.project-dir
  generate.debuglevel

ERROR
deploy.scp.host
krysalis.sourceforge.net
deploy.scp.root
/home/groups/k/kr/krysalis/htdocs
deploy.scp.user
forrestbot
deploy.scp.dir
forrest

- and CALL the following templates (template.workstage(.@type)?):
  - template.prepare
  - template.get-src.cvs
  - template.generate
  - template.deploy.scp
  - template.cleanup

What each of these templates is going to do with the parameters can be found in the templates.build.xml itself (at this stage) and is put out to the console if the bot is run with the `-Dtemplate.echo=true` flag

2.2.1. Setting bot-wide defaults

The `<defaults>` section in the `<forrest-config>` allows for one-time setting of parameters you want to reuse for different projects. The syntax (and semantic value) of the nested elements is identical to the use inside the `<project>` tags.

Note:

Once you understand all of this you might decide NOT to run your own centralized service but rather ask your project to be taken up in the centralized service of the project team itself: just send a mail to forrest-dev mail list defining the required parameters to be taken up in the process.

2.3. Checking logs and getting them by mail

For each distinct project the Forrest bot is maintaining a separate working log of the different workstages being executed. These logs can be opened and viewed in the ./build/bot directory

In addition to the @name attribute the <project> tag also can contain a @sendlogto attribute that must be holding a (comma separated list off) email address(es) where the project specific log needs to be sent to after all workstages have been completed (or as soon as building failed)

3. Current supported workstages and types.

3.1. <prepare>

No specific type versions. (don't use type-attribute)

Template arguments:

- `<skin>` @name holds the name of the skin to use.

3.2. <get-src ...>

3.2.1. <get-src type="cvs">

Template arguments:

- `<host>` @name holds the ipaddress or dns name of the cvs serving host
- `<root>` @name holds the cvs-root directory on that host
- `<user>` @name holds the username to use on the cvs repository
- `<passwd>` @name holds the password to use on the cvs repository
- `<module>` @name holds the module name that holds the
- `<dir>` @name holds the relative path to the {docroot} directory. (This is the dir that is holding the ./content/xdocs as specified in t.

3.2.2. <get-src type="local-copy">

Template arguments:

- `<content-dir/>` @name holds the path pointing to the {docroot} directory. (See the contract to understand what should be there.)
- `<project-dir/>` @name holds the path pointing to the {projecthome} directory where the xml project descriptors reside.

3.3. <generate>

No specific type versions. (don't use type-attribute)

Template arguments:

- `<debuglevel>` @name holds the threshold-level for showing debug statements in the cocoon generation process.

3.4. <deploy ...>

3.4.1. <deploy type="scp">

Template arguments:

- `<host>` @name holds the remote host where the file will be copied to.
- `<user>` @name holds the username to be used when logging onto the remote host.
- `<root>` @name holds the prefix part of where the content needs to be published.
- `<dir>` @name holds the suffix part of where the content needs to be published.

Note:
Using this approach requires that the public key of the user executing the bot target is present in the authorized key file (`~/.ssh/authorized_keys2`) of the remote user (on the remote host).

Note:
Might be interesting to understand that the full process actually is bundling all files to copy in a *.tar.gz that is un-tar.gzipped on the remote host using ssh.

3.4.2. <deploy type="local-copy">

Template arguments:

- `<destination>` @name holds the path where the generated content needs to be copied to.

3.4.3. <deploy type="ftp">

Template arguments:

- `<host>` @name holds the hostname of the ftp server to publish to.
- `<user>` @name holds the username to use for loging onto the ftp server.
- `<passwd>` @name holds the password to use.
- `<destination>` @name holds the path to the directory on the remote host where the generated content needs to be published. The process is using the `ftp cd` command to get there. This means this directory has to exist.

Note:
Currently the ant distribution that is included in forrest cvs is not offering the required NetComponents.jar to actually support this type of deployment. If you want to use it anyway you should get the required jar from http://www.savarese.org/oro/downloads and drop that into your `{forrest-sandbox}/tools/antipede/lib` directory.

3.5. <cleanup>

No specific type versions. (don't use type-attribute)

Template arguments: None.

4. ForrestBot design

Most of us will just like things to work, and will be happy enough just using it, maybe even letting the forrest-dev mail list know if we could improve this or the other. Some however might be drawn to the how stuff works of things. For them is this section as a start in the rest of their own pursuit.

4.1. The Ant play

All actions and targets related to the actual execution of the bot's work is separated over 3 distinct build files:

The complete process runs as follows:

1. 'bot' target inside main build.xml depends on a number of targets that
   1. 'bot.init': initialize the environment
   2. 'bot.conf2build': use XSLT stylesheets to convert the forrestbot.conf.xml into the work.build.xml (and a properties file for the default-section)
   3. 'bot.run': delegates to the 'work' target of the generated work.build.xml
2. 'work' target inside generated work.build.xml will then start in parallel (so failure of the one would not influence the other) the different project specific work-processes.
3. each of these project runs (generated 'work.[projectname]' targets) depends on the sequential execution of the different workstage actions (generated '[workstage].[projectname(.[type])?]' targets)
4. In those specific targets the pattern is as follows:
   1. set the project specific parameters for this project and workstage
   2. load the default parameter values for the ones that aren't set yet
   3. call the actual template.[workstage(.[type]?)]

4.2. Adding features to the bot - The meta template

Given the mechanism, the actual things the bot **can** do boil down to the template targets it can call. So extending the bot means: adding templates-targets.

If you want to add your own tempales to `template.build.xml`, take in account the following guidelines:

- in the future there will be a DTD for forrestbot.conf.xml that should be updated probably after you added stuff to the `templates.build.xml`. Backwards-incompatible changes need a formal vote.
- build an echo target for each template target you make:

```
<target name="echo.workstage-name.optional-type" if="template.echo">
    ... here you put echo ant-tasks that somewhat tell what you will
    ... do with all the params you expect
</target>

<target name="template.workstage-name.optional-type"
            depends="shared.set-props, echo.workstage-name.optional-type">
    ... here you put whatever ant-tasks that actually do it
</target>
```