For Beehive Developers

This page contains information useful to committers and contributors to Beehive.

Contents

Building

Below are the steps for building Beehive:

- download and install J2SE 5.0 JDK from here. Beehive requires J2SE 5.0 or greater.
- download and install Subversion from here
- sync the Beehive tree using these steps

```
mkdir beehive
cd beehive
svn checkout https://svn.apache.org/repos/asf/beehive/trunk
```

- follow the instructions in BUILDING.txt

More about Subversion and the ASF Source Code Repository

Additional references:

- The Apache Software Foundation Source Code Repository
- svn online manual
- intro to svn for cvs users

Example operations:

```
1. adding new files: svn add
2. editing existing files -- just do it, svn doesn't require an "open" or "edit" command.
3. syncing to repository changes: svn update
4. submitting changes: svn commit
5. examining history: svn log
```

Note that svn treats directories as first class objects, so you need to "svn add" them.

No authentication is needed for read-only access to the repositories. However, you must be a committer and authenticate using your Apache userid/svn password when using svn to do any modifying operations. Set your svn password according to the instructions at http://www.apache.org/dev/version-control.html.

Build Conventions

The Beehive build is structured such that the Beehive components are peers and have a location in which to reference shared components. Artifacts of these shared components are defined as properties in the `/beehive-imports.xml` file, which can be included by any downstream Ant build file. Some of these artifacts include references to the XMLBeans JAR, the Servlet and JSP API JARs, and the JUnit JAR. The motivation behind this is to minimize the number of properties used to refer to the same resource, and when possible, these common properties should be used in component projects.

The structure of the Beehive source tree from ${beehive.home}/ is:

```
  ant/                -- Ant build files shared among Beehive components
  beehive-imports.xml -- defines Ant paths, filesets, and macros that are shared among the Beehive components
  build.xml           -- top-level build targets that can be used to clean, build, deploy, and test Beehive
components
  build/              -- a transient top-level directory in which project wide build artifacts are placed
  controls/           -- the Controls component
  docs/               -- website and Beehive documentation
  external/           -- 3rd party libraries / software that is shared among Beehive components
  installed/          -- installed 3rd party software such as Ant and Tomcat
  netui/              -- the NetUI component
  samples/            -- all of the shipping  samples
  system-controls/    -- the Beehive system controls JDBC, JMS, EJB, and web service
  test/               -- common test code and infrastructure
  wsm/                -- the Web Service Metadata component.
```

In order to track dependencies between Beehive components, it's best practice to define resources at the top-level that are shared across components. Otherwise, scoping resources to the components that use them makes them easier to change, and it's always easy to promote resources up the tree but harder to move them down. Generally, one component should never directly reference another for files, properties, etc.

The top-level `ant/` directory contains source and Ant files that are shared among components. This includes Ant which can be used to start / stop Tomcat (the default Servlet container for Beehive) and to deploy / undeploy / build a Beehive web application. The Ant files used to perform these operations are:

- beehive-tools.xml
- beehive-runtime.xml
- tomcat-imports.xml

and are defined as top-level properties in the beehive.properties file so that components need not reference the build files directly.

The component builds should expose a set of common targets that can be invoked from the top-level build. This makes the `/build.xml` file clean and allows developers to move between targets and incrementally build with the same set of targets.

```
clean     -- clean the artifacts generated during a build
build     -- build the component's source files into JARs
deploy    -- deploy the components's runtime bits into a simulated distribution (coming soon)
drt       -- run the Developer Regression Tests (drt)
docs      -- build the documentation for a component
```

In general, Beehive components are structured as:

```
  build.xml   -- the component's top-level build file implementing the targets above
  ant/        -- Ant build files used by a component
  build/      -- transient build directory that contains build and test artifacts
  docs/       -- documentation for a component
  external/   -- external libraries used by a Beehive component.  For example, NetUI has the Struts runtime
here.
  src/        -- root source directory
    <source modules>
  test/       -- test cases, sources, and Ant files
  tools/      -- tools used by a component
```

This project structure can be nested in the `tools/` and `test/` directories as needed.

General guidelines:

- build/ directories at the top-level and in components should not be checked into the tree
- all artifacts generated during a build should be placed in the build/ directory so that a "clean" of Beehive can easily delete generated files.

Note, philosophically speaking, this build structure isn't meant to be a set of hard-and-fast rules; rather, it's a set of guidelines defined with the intention of making understanding Beehive and its components easier and in keeping the build and test infrastructure manageable and loosely coupled. There are exceptions to every rule. <g>

Testing in Beehive

In general, tests that must be run before every checkin are "drts" (developer regression tests), and longer tests that are run regularly (but not before every checkin) are "bvts" (build verification tests). The "drt" ant target exists at the root of the tree, and in each subproject. DRTs are run in each subproject using the command:

```
cd <sub-project>
ant drt
```

BVTs are run in each project with a similar command:

```
cd <sub-project>
ant bvt
```

In both cases, the tests passed when Ant displays the usual BUILD SUCCESSFUL message. When the tests fail, the BUILD FAILED message is shown and detailsa about the failures can be found in each test suite's log files.

Specific information on testing in each subproject can be found here:

- NetUI
- Controls
- System Controls
- WSM

To run a full build and test suite against *both* the SVN tree and distribution, run:

```
cd ant
ant -f nightly.xml run
```

This target can take over an hour to run depending on the computer hardware. It will run all of the DRTs and BVTs for each sub-project in Beehive.

Coding Conventions

CodingConventions

Source Headers

source header files

Documentation Conventions

DocConventions

Setting up Beehive in an IDE

Each Beehive sub-component has a set of source paths and library dependencies that are needed to develop Beehive from an IDE. Below are the list of sub-components and their source paths and required libraries. In addition, there are (will be) specific instructions for how to setup a Beehive development environment in various IDEs. The paths below are referenced from `$BEEHIVE_HOME` unless otherwise specified.

General

- Controls
  - Source Paths
    - `controls/src/api`
    - `controls/src/runtime`
    - `controls/src/spi`
  - Library Dependencies
    - `$ANT_HOME/lib/ant.jar`
    - `$JAVA_HOME/lib/tools.jar`
    - `external/commons/commons-discovery-0.2.jar`
    - `external/servlet/servlet-api-2.4.jar`
    - `external/velocity/velocity-1.4.jar`
    - `external/velocity/velocity-dep-1.4.jar`

- NetUI
  - Source Paths
    - netui/src/bootstrap
    - netui/src/compiler-core
    - netui/src/compiler-xdoclet
    - netui/src/compiler
    - netui/src/core
    - netui/src/javascript
    - netui/src/pageflow
    - netui/src/scoping
    - netui/src/tags-databinding
    - netui/src/tags-html
    - netui/src/tags-template
    - netui/src/util
    - netui/src/webapp-template
  - Library Dependencies
    - $ANT_HOME/lib/ant.jar
    - $JAVA_HOME/lib/tools.jar
    - external/commons-codec/commons-codec-1.3.jar
    - external/xmlbeans/apache-xbean.jar
    - external/servlet/servlet-api-2.4.jar
    - external/servlet/jsp-api-2.0.jar
    - installed/jsr173/jsr173_1.0_api.jar
    - netui/external/commons-el/commons-el.jar
    - netui/external/jsf/myfaces-1.0.9/lib/myfaces.jar
    - netui/external/jstl/jstl.jar
    - netui/external/jstl/standard.jar
    - netui/external/struts/commons-beanutils.jar
    - netui/external/struts/commons-collections.jar
    - netui/external/struts/commons-digestor.jar
    - netui/external/struts/commons-fileupload.jar
    - netui/external/struts/commons-logging.jar
    - netui/external/struts/commons-validator.jar
    - netui/external/struts/jakarta-oro.jar
    - netui/external/struts/struts.jar
    - netui/external/xdoclet/xdoclet-1.2b4.jar
    - netui/external/xdoclet/xdoclet-web-module-1.2b4.jar
    - netui/external/xdoclet/xjavadoc-1.1-v3.jar
  - Optional Library Dependencies (to include JSF support)
    - $JSF_HOME/lib/jsf-api.jar
    - $JSF_HOME/lib/jsf-impl.jar
- WSM
  - Source Paths
    - wsm/src/api
    - wsm/src/core
    - wsm/src/axis
  - Library Dependencies
    - $JAVA_HOME/lib/tools.jar
    - $ANT_HOME/lib/ant.jar
    - controls/build/jars/beehive-controls.jar
    - external/commons/commons-logging-1.0.4.jar
    - external/servlet/servlet-api-2.4.jar
    - external/velocity/velocity-dep-1.4.jar
    - external/xmlbeans/apache-xbean.jar
    - installed/jsr173/jsr173_1.0_api.jar
    - wsm/build/jars/beehive-wsdltypes.jar
    - wsm/external/axis.jar
    - wsm/external/axis-ant.jar
    - wsm/external/jaxrpc.jar
    - wsm/external/saaj.jar
    - wsm/external/wsdl4j.jar
- System Controls
  - Source Paths
    *system-controls/src/ejb
    *system-controls/src/jdbc
    *system-controls/src/jms
    *system-controls/src/webservice
  - Library Dependencies
    - system-controls/external/commons/commons-collections.jar
    - external/commons/commons-logging-1.0.4.jar
    - system-controls/external/geronimo-spec-j2ee-1.4-rc2.jar
    - system-controls/external/geronimo-spec-jms-1.0-M1.jar
    - wsm/external/axis.jar
    - wsm/external/axis-ant.jar
    - wsm/external/jaxrpc.jar
    - wsm/external/saaj.jar
    - wsm/external/wsdl4j.jar
    - wsm/lib/wsdltypes.jar
    - external/xmlbeans/apache-xbean.jar

- `installed/jsr173/jsr173_1.0_api.jar`
- `wsm/build/jars/beehive-jsr181.jar`

Eclipse

Tested with eclipse 3.1.1

You must first setup Eclipse 3.1 to use Java 5:

- Start Eclipse with a Java 5 JVM.
- Set Eclipse to use Java 5 compiler settings. You can do this by going to Windows->Preferences->Java->Compiler->Compiler compliance level and setting it to *5.0*. You may also want to create a *beehive* workspace specifically for working with Beehive.

You will need to create three projects in your workspace:

- controls
- netui
- wsm
- system-controls

Each of these projects will have certain source paths, project dependencies, and library dependencies that you need to setup.

Handling XMLBeans

Because Eclipse 3.1 does not have any native or plugin support for Apache XMLBeans, you will need to generate a couple jars from Beehive XML Schemas.

To create the schemas jars, run the following:

```
$ ant -f ${beehive.home}/netui/ant/build-schema.xml
$ ant -f ${beehive.home}/wsm/build-schema.xml
```

This will generate two jar files, `netui-schema.jar` and `wsm-schema.jar`
for use with Eclipse. Any time that Beehive's XML schemas are updated, you will need to re-generate these jar files.

Project Setup

Create the projects, controls, netui, and wsm. For each of the projects, setup its source, project dependencies, and libraries path:

Controls Project

Controls Project rooted at `${beehive.home}/controls`

Set the source and library dependencies as specified previously.

This project has no dependencies.

Netui Project

Netui Project rooted at `${beehive.home}/netui`

Set the source and library dependencies as specified previously. Additionally, make sure that you add the `netui-schema.jar` which you generated earlier to the library path.

This project has a dependency on the controls project.

WSM Project

WSM Project rooted at `${beehive.home}/wsm`

Set the source and library dependencies as specified previously. Additionally, make sure that you add the `wsm-schema.jar` which you generated earlier to the library path.

This project has a dependency on the controls project.

System Controls Project

System Controls Project rooted at `${beehive.home}/system-controls`

Set the source and library dependencies as specified previously.

This project has a dependency on the controls and WSM projects.

Ant Setup

Please add junit.jar in ANT's classpath. (NOT eclipse's classpath) (Window -> Preferences -> Runtime -> Ant -> add {`${beehive.home}/external/junit/junit.jar`} in Global Entries in the Classpath tab)

IDEA

Tested with IDEA 4.5.x and Irida builds.

No specific settings are required. Please follow the General setup steps above.

NetBeans

Tested with NetBeans 4.0 beta 2

Please use New Project > Standard > Java Project with Existing Ant Script for each subproject. You need to run beehiveCmd.env or beehiveCmd.sh first and start up NetBeans on the same command window or shell.

Creating and Applying Patches

Creating a Patch

1. cd to the root directory of the Beehive tree.
1. Make sure your *entire tree* is updated to a single revision (preferably the head revision, so you take on the burden of resolving merge conflicts). `svn update` here will do the trick.
1. Create the patch.

```
svn diff > patch.txt
```

Applying a Patch

(This section contains at least one nonobvious step.)

1. cd to the root directory of the Beehive tree.
1. Open the patch file (say, patch.txt) and get two pieces of information:
a. The root directory for the patch. You'll need to infer this from the first "Index:" entry. It should be the root of the Beehive distribution, but it helps to verify this.
a. The base revision of the patch. Every file entry should look something like this:

```
            --- netui/src/pageflow/org/apache/beehive/netui/pageflow/PageFlowActionServlet.java   (revision
149222)
            +++ netui/src/pageflow/org/apache/beehive/netui/pageflow/PageFlowActionServlet.java   (working
copy)
```

In this case, the base revision would be 149222.

1. **Sync your tree to the base revision of the patch.** (this step is non obvious) `svn update -r 149222` in the example above. If you don't do this, you risk applying the patch to a different base, with unpredictable results.
1. Apply the patch. `patch -p0 < patch.txt` If you use Windows, see below for information on the `patch` command.
1. Update the tree to the head revision. `svn update`. If the patch was based on an early revision, you may end up resolving merge conflicts. But if you do, you can thank your lucky stars that you updated to the base revision of the patch before applying it. 🙂
1. Build, run the Beehive checkin tests with `ant drt`, commit.

Information for Windows Users

If you're running Windows, you can get the `patch` command as part of Cygwin (http://www.cygwin.com/). The MKS 6.1 `patch` command does *not* work for this – if you use MKS, download Cygwin, make sure it's on your path, and disable the MKS patch.exe.

Creating a Beehive Distribution

Creating a Beehive distribution is reasonably easy and consists of a couple of basic steps:

Building and testing the distribution archives

1. Update the Beehive documentation to replace these tokens "Beehive SVN" with "Beehive <version-number>"
1. `cd trunk/`
1. Run this command to build and test the distribution: `ant -f nightly.xml run -Dbeehive.version=<version-number>`
1. Checksum and sign the release
1. Create these sub-directories: `/www/www.apache.org/dist/beehive/<version-number>/(binaries|source)`
1. FTP the distribution's `.zip`, `.tar.gz`, `.asc`, and `.md5` files to an ASF server
1. Copy the appropriate resources to the `<version-number>/(binaries|source)` sub-directories

Before FTP-ing to the ASF servers, be sure that **all** of the distribution's tests pass by manually verifying a 100% test pass rate in the generated test reports.

Linking the distribution the website

1. Link the documentation for the release to `beehive/site/src/documentation/content/xdocs/documentation.xml`
1. Copy an existing download .cgi script and rename it to `release-<version-number>.cgi`
1. Copy an existing download .html file and rename it to `release-<version-number>.html`
1. Link the download for the release to `beehive/site/src/documentation/content/xdocs/downloads.xml`. Be sure to link to the `.cgi` script
1. Replace the version number in `release-<version-number>.html` with the new <version-number> token
1. Add `releases/release-<version-number>.html` to `site/src/documentation/conf/cli.xconf`
1. Run `cd beehive/site && ant clean build site.stage`
1. Copy `$FORREST_HOME/main/site/releases/release-<version-number>.html` to `beehive/site/www/releases` to work around a very unfortunate Forrest bug
1. Replace the download HTML <form> in `www/releases/release-<version-number>.html` with the <form> templated for the ASF CGI download script from `src/documentation/content/xdocs/releases/release-<version-number>.html`. This is needed because Forrest mangles the tokens that are replaced when the .html file is used as a template from the download's .cgi script
1. Commit the site changes to SVN
1. SSH to an ASF server and run `cd /www/beehive.apache.org && svn update`

While this seems like lots of steps, updating the website doesn't take that long. The worst part is working around the Forrest mismatch with the .cgi scripts!