EchoPoint Component Library
Developer Guide
Version 0.80

Rakesh Vidyadharan

November 11, 2008

Contents

List of Tables

1 Introduction

This document serves as a guide for the developers working on re-implementing EchoPointNG for Echo3. The guide covers source code organisation, unit testing guidelines etc.

The primary focus of the re-implementation effort is to make available the most popular components from EchoPointNG. Some components from EchoPointNG are made obsolete by complimentary components in the Echo3 Extras library. The team will try to avoid duplicating components, unless the features provided by EchoPointNG were far in excess of those made available by Echo3 Extras. In the event a decision is made to implement a component that already exists in Echo3 Extras, it will be treated as a lower priority. The list of components that are planned to be re-implemented are presented in [1].

Javadoc API Specifications (see [2]) for the component library will be updated and maintained periodically. Javadocs will be generated with linked source code, enabling users to view the implementation files without needing to check out the source code.

1.1 Maintaining Component List

Developers who contribute to the re-implementation effort must update the component list (see [1]) as appropriate. Filling in the implementor name against the component will assist with bug-fixing efforts as well. This is also the best means of taking responsibility of a component to avoid duplication of effort by other developers involved in the effort. Hence updating this list and committing to the repository at frequent intervals is the best way to ensure that no duplication of effort occurs.

1.2 Package and Namespace

In keeping with NextApp using `echo` instead of `echo3` for all packages and namespaces we shall be using `echopoint` as the package/namespace for the library. Accordingly the server side component interfaces are under the `echopoint` package, while the client side components are under the `echopoint` namespace.

1.3 Java Version

At present all the server side sources require Java 5 as the minimum JDK version. This may be changed in future to support Java 1.4 if so required.

1.4 Source Code Repository

The source code for the project is maintained in the subversion server provided by java.net. The location of the repository is:

https://echopoint.dev.java.net/svn/echopoint/

Note that the working copy of the code is available under the `trunk` area. An older attempt to port EchoPointNG by just changing package names is available under the `branches/obsolete` directory.

1.4.1 Checking Out and Building Library

The following steps illustrate the process of checking out and building the library.

```
svn co https://echopoint.dev.java.net/svn/echopoint/trunk \
  echopoint --username <username>
cd echopoint
cp ant.properties.sample ant.properties
vi ant.properties # Edit as necessary to suit environment
ant
```

1.4.2 Gaining Access to Repository

Anyone[1] may check out the source code repository and use the library. Developers who wish to contribute to the project may request write access to the repository by contacting `rakesh` through the java.net project page or the Echo Developer Forum.

1.5 Licence

EchoPoint will be distributed under the Mozilla Public Licence (see [3]). The `LICENCE.txt` file under the `trunk` directory in `subversion` (see section 1.4) may be used to copy the licence clauses to each source file in the project.

[1] java.net allows only members to access subversion.

2 Creating releases

New releases of the library may be created by the developers working on the project when new components are added, or when modifications or bug fixes have been applied to existing components. Sections 2.x describe the steps to be followed to create a release of the library.

2.1 Version Number

Update the version number for the library. The version number is maintained in the ant build properties files. Remember to update the version number in the master `ant.properties.sample` file. All version numbers start with `3` since this is the third generation of the library. The general form of the version numbers is shown below.

```
3.<major version>.<minor version>
```

Alpha/beta release numbers may be appended to the version number to indicate the maturity of the release. The major and minor version numbers are incremented only after the library reaches stable level (passes beyond alpha/beta stage).

2.1.1 Major Version

Major version numbers are incremented when new components are added or significant enhancements are applied to the code base. Major changes that affect multiple components lead to the major number being incremented. In general refactoring, bug fixes and performance enhancements that affect multiple components will lead to incrementing major version number.

2.1.2 Minor Version

Minor version numbers are incremented when minor modifications are applied to the code base. Minor modifications typically affect only a single or a small number of components.

2.2 Library

A versioned `jar` file should be uploaded to the Documents & files>releases area on java.net. Note that the `releases` directory has been further divided into `alpha`, `beta` and `stable` directories. The project home page should also be updated with the link to the latest release file.

2.3 Announcement

Create a new announcement under the Announcements page on java.net. In the announcement, list the additions, modifications ... that were applied in the new release.

2.4 Tagging

Create a tagged version of the trunk under the `tags` area in subversion for each release. The following steps show the process of tagging a release.

```
svn copy https://echopoint.dev.java.net/svn/echopoint/trunk \
  https://echopoint.dev.java.net/svn/echopoint/tags/<version> \
  -m "Version 3.x.x of library with new components - Xxx, Xxx \
  and bug fixes for issues x, y, z."
```

For convenience of users, an additional `current` directory is maintained under the `tags` area that contains the latest released revision. This will need to be deleted and recreated with each release.

```
svn delete https://echopoint.dev.java.net/svn/echopoint/tags/current \
  -m "Deleting old version to move to version: 3.x.x"
svn copy https://echopoint.dev.java.net/svn/echopoint/trunk \
  https://echopoint.dev.java.net/svn/echopoint/tags/current \
  -m "Version 3.x.x of library with new components - Xxx, Xxx \
  and bug fixes for issues x, y, z."
```

Notes:

- A simple shell script `bin/release.sh` has been provided that will perform the above steps. You may invoke the script with the release number to create the tagged version of the sources.
- Another simple shell script `bin/rerelease.sh` has been provided that will re-release a tagged version of the sources. Re-releasing involves deleting the previous tagged release and recreating it.

2.5 Test Applications

Updated test applications may be uploaded to the Documents & files>releases area of the project. This will allow users the ability to test and view the components provided by the library.

echopointclienttest.war - The packaged client-side test application.

echopointtest.war - The packaged server-side test application.

3 Project Directories

The project contents are organised under the following primary directories:

src - The root directory under which the source code for the library are stored. Note that both client and server side sources are stored under this root.

lib - All third party libraries used to implement and test the library are stored under this directory.

docs - All documentation pertaining to the project are stored under this directory. Note that Javadoc API specifications are generated under a sub-directory of this directory by the build script.

www - The directory used to control the content on the project java.net page. Please do not use the directory for any other purpose. You may update the content displayed on the project home page by editing the `index.html` page.

Note that the project directory structure tries to remain similar to the structure followed by the Echo3 and Echo3 Extras projects. Table 1 shows the key directories and their purpose.

Table 1: Project directories

Directory	**Purpose**
`src/client`	The root directory under which client side implementation of the library exists.
`src/client/js`	The root directory under which the client-side components and their rendering peers are stored.
`src/client/test`	The root directory under which the test client application is stored.
`src/server-java`	The root directory under which the server side implementation of the library exists.
`src/server-java/app`	The root directory under which the sources for the components exist.
`src/server-java/webcontainer`	The root directory under which the sources for the component sync peers exist.
`src/server-java/test`	The root directory under which the sources for the server side test web application exist.

3.1 Client Side Sources

All client side sources are stored under the `client` sub-directory. This directory is further split into directories for the library implementation sources and the client test application.

3.1.1 Client Side Library

The component definition and sync peer sources are stored under the `js` sub-directory. Table 2 shows the primary source files available in this directory. Note that the naming convention follows the same convention as followed by Echo3 and Echo3 Extras.

Table 2: JavaScript Sources

File	Purpose
`Echopoint.js`	The primary source file that bootstraps the client environment. See table 3 for a description of the primary namespaces maintained by the library. See section 3.2.2.2 for a description of how the client-side environment is bootstrapped by the server.
`Application.<Component>.js`	The source file that defines the client-side interface of the component. Note that inheritance hierarchies may be stored in a single file to reduce client-server interaction.
`Sync.<Component>.js`	The client-side sync peer for the component.

Table 3: JavaScript Namespaces

Namespace	Purpose
`echopoint`	The primary namespace for the library. Similar to the server-side library, all components are made available under this root namespace. Add any global routines, objects that may be shared among components to the primary `Echopoint.js` file.
`echopoint.constants`	A namespace used to define constants used by the client-side library. In particular constants are defined to hold the component names, since the same name needs to be used in both the component (`Application.xxx.js`) and sync peer (`Sync.xxx.js`) files.
`echopoint.model`	A namespace used to define model objects used as the backing data for client-side components.
`echopoint.internal`	A namespace used to hold abstract base classes and other similar classes that are not meant to be publicly used.

3.1.2 Client Test Application

A test application (similar to the server-side test application[2]) is used test the client-side library. All the primary classes for the test application are maintained under the `echopoint.test` namespace. A unit test framework like Selenium may be used to create a test suite around the test application. The test application may be accessed at [4].

[2] See section 3.2.3 for details.

The test application sources are stored under the `app` sub-directory. Sections 3.1.2.x describe the source files that need to be edited and created to add tests to the application. Table 4 shows the directory structure for the test application.

Note: The contents of the JavaScript files noted in the succeeding sections may be combined within a single file. They are maintained in separate files to keep the application code structurally similar to the server-side test application (see section 3.2.3).

Table 4: Test Application Directory Structure

Directory	Purpose
`app`	The root directory under which the sources for the test application are stored.
`lib`	The root directory under which external libraries required for the application are stored.
`lib/corejs`	The root directory under which the Echo3 Core client-side library files are stored.
`lib/echo`	The root directory under which the Echo3 App client-side library files are stored.
`lib/extras`	The root directory under which the Echo3 Extras client-side library files are stored.

3.1.2.1 index.html

The XHTML 1.0 page that is used to load the client-side test application and all the requisite libraries. This file is maitained under the root `test` directory. Update `head` section with the new JavaScript files that were added. Note that the client-side test application can be tested directly by loading the index page off the filesystem without needing to be deployed to a web/application server.

Notes

1. The client-side library is accessed using relative file references. This may be changed at a later date to copy the library files to a sub-directory to allow loose coupling with the rest of the library.
2. A utility class may be developed that will automatically scan the `app` directory update the index file to avoid having to manually maintain this file.
3. The YUI compressor may be used to reduce load time of the test application. The `clienttest` ant task can be used to generate the `war` file that contains a compressed deployable version of the client-side test application. Note that the files added to `index.html`

should also be added under the `filelist` elements in `build.xml` for a proper test package to be generated.

3.1.2.2 ComponentList.js

The class names of the components to be tested are maintained by the `COMPONENTS` array. Add the class name of the client-side component that is being tested to the array.

3.1.2.3 XxxTest.js

A test class with name `<Component>Test` needs to be added under the `echopoint.test` namespace in the `app` directory. These test classes will take the container component that they will populate as a constructor parameter. If possible user interaction may be enabled to perform manual testing of the components. See the existing test files for samples on usage and set up.

Notes

1. No real styling is done in the test application. The style class as used in the Echo demo application has been copied over, but is not being used properly. Eventually the test application must use proper styles to make the application look presentable.
2. Please assign a unique `renderId` for each component being tested and any associated control components using the `Echo.Component.renderId` property. This makes it easy for automated test tools to trap and monitor the DOM for test results. All `renderId` values are assigned a prefix of `echopointUnitTest` to easily distinguish them (see example below).

```
var button = new Echo.Button(
{
  renderId: "echopointUnitTestDirectHtmlDisplay",
  style: "Default",
  text: "Save"
} );
```

3. Selenium (or similar) web application testing framework may be used to create a interface unit test suite around the test application.

3.2 Server Side Sources

Sources for the server side components and sync peers are available under the `server-java` sub-directory. Staying similar to the Echo3 source tree, the component sources are stored under the `app` directory, while the sync peer sources are stored under the `webcontainer` directory. The unit test application for the server side code is stored under the `test` directory.

3.2.1 Server Side Components

All the components made available by the library are stored under the `echopoint` package under the `app` directory. Sub-packages may be created as necessary to hide implementation details of the components.

3.2.2 Server Side Sync Peers

All the sync peers for the components made available by the library are stored under the `echopoint` package under the `webcontainer` directory. Sub-packages may be created as necessary to hide implementation details of the components. Note that the library distribution file includes both the components and sync peers in one package unlike the Echo3 distribution that separates them into two separate distribution files.

3.2.2.1 Resource Location

Resource files[3] are stored under the `resources` root directory of the distribution library file. Table 5 shows the standard directory structure for referencing resources files from sync peer code.

Table 5: Resource Files

File	Purpose
`resource`	The root directory under which all resource files are stored. Note that we do not use the package of the class file as a base directory of the resources. There is no difference between the two approaches, this just makes the files easier to find.
`resource/js`	The root directory under which all JavaScript implementation files (client side component and sync peers) are stored.
`resource/images`	The root directory under which image resources for the various components are stored.

3.2.2.2 Bootstrapping Environment

The client environment for the library is bootstrapped through the following classes. Each server-side sync peer must invoke the bootstrapping code as appropriate. See existing code samples for examples on how to invoke the bootstrapping code.

[3] JavaScript libraries, images etc.

3.2.2.2.1 CommonResources

A static class used to bootstrap the `echopoint` namespace. Invoke `CommonResources.install` within a `static` block of code in each sync peer.[4]

```
static
{
  CommonResources.install();
  WebContainerServlet.getServiceRegistry().add( COMPONENT_SERVICE );
}
```

3.2.2.2.2 CommonService

A static class used to load the code in the `echopoint` core client library. This class must be used in the `init`[5] method of the sync peer.

```
public void init( final Context context, final  Component component )
{
  super.init( context, component );
  ServerMessage serverMessage =
    (ServerMessage) context.get( ServerMessage.class );
  serverMessage.addLibrary( CommonService.ECHOPOINT_SERVICE.getId() );
  serverMessage.addLibrary( COMPONENT_NAME );
}
```

3.2.2.2.3 AbstractPeer

A sync peer class that may be used as the base class for all sync peers. This class loads both `CommonResources` (see section 3.2.2.2.1), and `CommonService` (see section 3.2.2.2.2). If using this class as the base of the sync peer, you need not worry about bootstrapping the `echopoint` namespaces.

3.2.3 Server Side Test Application

A web application has been developed to unit test the server side implementation. Sections 3.2.3.x describe the source files that need to be edited and created to add tests to the application.

3.2.3.1 ComponentList

The class names of the components to be tested are maintained by the `COMPONENTS` array. Add the class name of the component for which a test is being added to the array.

[4] This is not necessary if a super class performs this bootstrapping.
[5] This is not necessary if a super-class performs this loading.

3.2.3.2 XxxTest

A test case class with name `<Component>Test` needs to be added that runs the various unit tests necessary. In addition these tests updates the display area (right side) of the test application window with the component being tested. If possible some user interaction may be enabled to perform manual testing of the components.

Notes

1. All tests are written using JUnit 4 style and not 3.x style. See existing test case sources for samples.
2. No styling is done in the test application at present. This must eventually be modified to make the test application look presentable. Ideally this can be accomplished by creating default styles for the various components used by the application.[6]
3. Please assign a unique `renderId` for each component being tested and any associated control components using `Component.setRenderId`. This makes it easy for automated test tools to trap and monitor the DOM for test results. All `renderId` values are assigned a prefix of `echopointUnitTest` to easily distinguish them (see example below).

```
DirectHtml dhtml = new DirectHtml();
dhtml.setRenderId( "echopointUnitTestDirectHtml" );
```

4. Selenium (or similar) web application testing framework may be used to create a interface unit test suite around the test application.

4 Coding Guidelines

Sections 4.x layout out general coding guidelines for use by the developers involved in the project.

4.1 Build Properties

All build properties are stored in a master `ant.properties.sample` file. The `build.xml` file attempts to read in a local copy `ant.properties`. This dis-connect enables local build configuration without affecting the distribution. When updating the build rules, please remember to update the master property file. Do not add the local `ant.properties` to the subversion repository.

[6] At the time of writing default styles are not supported by Echo3.

4.2 Javadoc Comments

Try to ensure that all public elements (fields, methods, ...) are documented to make their purpose clear to end-users. Try and present sample code in the class level documentation, since that helps users get going with the library quicker. Adding links to screen captures to components stored under the images directory on java.net would be extremely useful.[7]

4.3 Static Constants in Client Components

Try and use static constants in client components similar to how they are done in their server side representations. This helps avoid any runtime errors caused by mistyping property names. At present the client components do not use the `PROPERTY_` prefix for properties.

4.4 Data Object Serialisation

Custom data model objects used to back server-side components may be easily serialised to the client engine in JSON. See the TagCloud component and its associated TagCloudPeer to see how the collection of Tag model objects are serialised to the client. The custom serialisation is affected by over-riding the getOutputProperty method and returning the JSON data for the property of interest.

This requires a little hack on the component sync peer to handle, but reduces the necessity of creating custom serialisers for the model objects. The XStream library is used to serialise the model objects in JSON.

Listing below illustrates how the collection of `Tag` model objects are serialised in JSON.

```
final XStream xstream = new XStream( new JsonHierarchicalStreamDriver() );
xstream.processAnnotations( Tag.class );
setTagsJson( xstream.toXML( tags ) );
```

Listing below shows how the JSON data is handled in the client-side sync peer.

```
var json = this.component.get( echopoint.TagCloud.TAGS_JSON );
if ( json )
{
  var data = eval( "(" + json + ")" );
}
```

[7] We will eventually create a component gallery on java.net project that will showcase the visual components provided by the library.

References

[1] EchoPoint Component List

[2] Javadoc API Specifications

[3] Mozilla Public Licence Version 1.1

[4] EchoPoint Client-Side Test Application