DfR Development Resources

All DfR developers should be familiar with the following resources.

Contents:

Source Code Repository

The DfR source code repository is hosted at https://github.com/duraspace/dfr. If you need access but don't have it yet, contact an existing developer.

New to git? Have a look at How I Use Git for some tips.

Once you have cloned the repository, see the README at the root of the source tree for instructions on how to build and test the code yourself.

Continuous Integration and Maven Site

Unit and integration tests are automatically run when changes are pushed to the `master` branch of the source code repository. The results of these builds (pass/fail) are available here on the DuraSpace Bamboo site.

The test build also automatically publishes a Maven Site here, which contains additional developer-oriented information about the codebase, including up to date javadocs, unit test coverage, findbugs reports, and more.

Integration tests currently use a single, shared DuraCloud instance at https://dfrtest.duracloud.org/. For instructions on logging in to this instance and/or running integration tests yourself, see the README at the root of the source tree. The instance currently runs a stock version of DuraCloud with S3 configured as a storage provider. It also runs an instance of Fedora 3.5 at http://dfrtest.duracloud.org:18081/fedora/objects If Fedora needs to be re-installed on this instance (e.g. after a reboot), you can upload this script and run it on the host as indicated.

Developer Notification Mailing List

Automatic notification of commits to the source code repository goes to the dfr-dev mailing list hosted at Google Groups. The list also receives automatic notification of continuous integration build failures. All DfR developers should be signed up to this list. If you're not yet, contact an existing project member.

Note: The `dfr-dev` mailing list is not intended to be used for discussion – we prefer to use Skype and the wiki for that.

Coding Conventions

Consider this a proposal for now. -Chris

If you use IntelliJ Idea 11, you can import this file to get the IDE to help you follow most of these conventions.

- Unix-style newlines. If your editor doesn't do them, they will be auto-converted because of the setting in the `.gitattributes` file at the root of our source tree.
- Indent with spaces..NO TABS.
- Four-space indents (8-space hanging indents) for Java
- Two-space indents (4-space hanging indents) for XML
- 80 character max per line (within reason – long URLs don't need to be unnaturally split, etc.)
- No wildcard imports.
- When wrapping arguments in a method definition or method call, use hanging indents rather than trying to line everything up in the same column.
- Code blocks:
  - Don't open a brace on an otherwise empty line
  - Always close braces on an otherwise empty line. Exception: if-else. See below for example.
- Javadocs:
  - For all classes and public methods.
  - Unnecessary when overriding (or implementing) a method defined elsewhere (use the `@Override` annotation in such cases).

Example code:

```java
package org.example.myapp;

import com.somecompany.somepackage.SomeException;
import com.somecompany.somepackage.SomeInterface;

/**
 * An example demonstrating the coding conventions.
 */
public class Example implements SomeInterface {
    private final String foo;

    /**
     * Creates an instance.
     *
     * @param foo the foo value.
     */
    public Example(String foo) {
        this.foo = foo;
    }

    @Override
    public void doSomething(String exampleArg1,
            String exampleArg2, String exampleArg3,
            String exampleArg4) throws SomeException {
        if (exampleArg1 != null) {
            // TODO: implement
        } else if (exampleArg2 != null) {
            // ...etc.
        }
    }
}
```