Stubs and RI "match" verification

This page documents the process(es) that implement a mechanism to validate that the RI and stubs are in agreement. It turns out that this is trickier than one might guess.

The RI and stubs need to agree in two areas:

- Their public API (Application Programming Interface).
- Javadoc comments (because in some cases the Javadoc describes the behavior of the system).

Comparing these two facets directly is somewhat problematic since the two code bases are formatted differently. Fortunately there is a tool (Javadoc) that is customizable for this situation. The Javadoc framework does the heavy lifting (parsing the code) and then calls a custom "doclet" - in this case APIRipper (https://community.cablelabs.com/svn/OCAPRI/trunk/common/tools/stublet/com/cablelabs/tools/stublet/APIRipper.java). This doclet constructs normalized files of the designated code base. Javadoc is invoked on the two code bases (the RI and the stubs) and the output saved on disk. These files are then "diff'd", resulting in output that indicates differences between the two code bases. The objective is to have no differences...

Of course that is "easier said than done". In addition we need a mechanism in order to ignore:

- trivial differences (such as format),
- non-trivial differences that have been identified and documented (in the form of a JIRA issue in the OCORI, OCSPEC, or OCVET systems). It can take a while for these differences to be investigated and resolved.

For trivial differences APIRipper has a mechanism using properties files for excluding various things (see the code for documentation).
This same mechanism is used for ignoring non-trivial differences that will likely be resolved by changing code in the RI. Each ignored component is identified in the file with a comment that points back to the particular OCORI issue. As the OCORI issues are resolved the ignored component should be included in the analysis.
For differences that may result in changing the stubs (and likely specifications), patches are checked into https://community.cablelabs.com/svn/OCAPRI/trunk/common/tools/stublet/known_diffs/. Theses patches are applied before analysis (and then reversed). Each patch identifies the particular OCSPEC or OCVET issue. As the OCSPEC/OCVET issue is resolved, the patch should be removed from the analysis.

There are ant targets that encapsulate the complexities:
In build.xml,

- verify.ri.stubs - a convenience target that applies patches, performs diff.rips, reverses the patches, and examines the stub_x_analysis.txt files (counts lines).
- diff.x.rip, where x is base, ds, dvr, fp, hn - these targets invoke Javadoc/APIRipper on a particular subsystem. The result is in stub_x_analysis.txt.
- diff.rips - Rips and compares all subsystems.
- ant -f rip_api.xml apply.patches - applies the patches to the stubs.
- ant -f rip_api.xml reverse.patches - reverses the patches to the stubs.

The workflow that I figured out when working on stub patches is

- Based on a non-patched RI
- ant -f rip_api.xml apply.patches
- ant diff.x.rip (analyze the appropriate subsystem).
- make changes to working copy of the stubs. Rerun the analysis as necessary.
- in the stubs directory, create the patch (usually svn -x -wb diff > new.patch)
- move/copy the patch into known_patches.
- add the patch to the apply.patches target.
- ant -f rip_api.xml reverse.patches

This way the patches are isolated - sometimes this doesn't work and you have to figure out a "late patch", but generally the described workflow works.