

UIMA C++ SDK build and packaging

The following are proposed changes to the UIMA C++ SDK build and packaging on Linux.

Build process

The following modifications are proposed to improve the UIMA C++ SDK build process and to have it conform to the standard practices used in GNU Autotools based build systems:

1. Build from source using commands: configure (with appropriate arguments), make and make install.
2. Run the build from the root of SVN extract to enable support for additional make targets such as make dist, make doc, etc.
3. Find and use system install of dependencies - APR, ICU, XERCES, ACTIVEMQ and check / verify minimum version requirements.
4. Require ActiveMQ CPP to be available and require the build of the ActiveMQ CPP based service wrapper (was optional).
5. Enable make dist target to create the source tarball.
6. Enable make check to run the fvt tests.
7. Add support for --with-library-suffix to enable user to build their version of the libraries under a different name.

Version Numbering

This discussion applies to the preparation of UIMA C++ binary package on Linux. Currently we do not use version numbers on Windows binaries.

Currently, with each release of UIMA C++ SDK, the library name is changed to reflect the release number. For example, in version 2.2.2 of UIMA C++, the library is named libuima-2.2.so and in version 2.3.0 it is named libuima-2.3.so. One reason for this library naming approach was to link the UIMA C++ release to the corresponding compatible releases of UIMA Java SDK and UIMA-AS.

Changing the library name with each release means that annotators and applications will work only with the one specific build and requires a rebuild of user code with each new release though there may not be any change to the library interface that breaks binary compatibility. To facilitate migration to newer releases and to allow annotators to work with multiple release, its proposed to move to package and library naming conventions used in GNU build systems. The compatibility and requirements with respect to UIMA Java SDK and UIMA-AS will be specified in the release notes.

There are two distinct features of UIMA C++ to consider when dealing with release compatibility:

- The framework dynamically loads annotators which are user code. The annotators make calls to UIMA C++ APIs and are built with some version of the SDK. A possible scenario is for an application to run annotators that were built with different releases of UIMA C++ SDK.
- The SDK depends on ICU, XERCES, APR and ACTIVEMQ-CPP and a release is built with a particular version of these. Binary compatibility therefore depends on the compatibility of these underlying libraries. In particular, ICU and XERCES encode the major and minor release numbers in the APIs which restricts binary compatibility across releases of these libraries. An application running UIMA C++ is restricted to running one version of the ICU library in a process and all annotators and underlying libraries must use the same ICU version.

The following is a proposal for UIMA C++ version numbers

Package Version Number

This is associated with a release and is in the form MAJOR.MINOR.REVISION with a change having the following meaning:

1. Advance REVISION number for bug fixes maintaining binary compatibility. e.g. fix serialization bugs
2. Advance MINOR number if adding new functionality while maintaining binary compatibility. e.g. add delta CAS and XML support
3. Advance MAJOR number if not binary compatible or source compatible. e.g. moving to new ICU ?

Library Version Number

Propose moving to the **libtool** versioning mechanism for versioning libuima.so. See here for more info: <http://www.gnu.org/software/libtool/manual/libtool.html#Versioning>

The libtool version numbers for libraries are in the form **current:revision:age** where
current is the number that identifies the interface the library implement
revision is implementation number
age is used represent the range of interfaces this library implements

NOTE: The libtool version number is not the same as the package number.

The libtool versioning system is enabled by invoking libtool with the --version-info option during the build.

For example, --version-info 0:0:0 produces library, soname and linkname as shown below:

```
lrwxrwxrwx 1 bsiyer users 20 Feb 2 18:22 libuima.so.0 -> libuima.so.0.0.0
-rwxr-xr-x 1 bsiyer users 15725180 Feb 2 18:22 libuima.so.0.0.0
lrwxrwxrwx 1 bsiyer users 20 Feb 2 18:22 libuima.so -> libuima.so.0.0.0
```

The soname, libuima.so.0, is determined by the value of current.

When preparing a public release, update the version info according to these rules:

Increment revision if there are no interface changes.

Increment current if the interface have been added, removed or modified and set revision to 0;

If the interfaces have been removed or modified, set age to 0 otherwise increment age.

RPM Packaging

Support for packaging as an RPM has been requested by developers of UIMA C++ annotators.