REDHAWK RELEASE NOTES

VERSION 1.9.0

SEPTEMBER 26, 2013

REDHAWK Release Notes, Version 1.9

The effort in REDHAWK 1.9 focused on simplifying maintenance, expanding component descriptions, and reducing the role of the assembly controller.

Changes in this Release

Maintenance is simplified by moving the BulkIO implementation from generated code to base classes. This move has two primary objectives: bug fixes in BulkIO ports no longer require the component code to be re-generated, and the scope of the code generators is greatly reduced, which results in less generated code per component.

Component descriptions are expanded by supporting complex types for properties. Complex values are critical for most signal processing components, and now those value types are fully supported in XML as well as in the component or device internal code. Complex properties are mapped to the complex Python type, the complex C++ standard template library, and the complex CORBA type in Java.

The role of the assembly controller is also de-emphasized by creating three extensions to the software assembly descriptor (SAD) file. First, external ports can now be re-named. The renaming of these ports allows ports from two components, which may have the same name, to both be exposed at the application level. Second, properties from any application component can be promoted to the application level (and re-named); application properties are no longer limited to the assembly controller properties. Finally, the uses device relationship can be established at the application level. This allows components to remain detached from the platform hardware, and thus more generic, by keeping the association to the application level.

Finally, 153 reported bugs in the core framework, code generators, BulkIO, and Python supporting infrastructure were resolved. Where applicable, these bug fixes were also

applied to the 1.8 series.

Upgrading from a Previous Version

The following sections contain guidance for upgrading from a previous version of REDHAWK.

Upgrading from REDHAWK 1.8.X

If you are upgrading from a previous 1.8.x version of REDHAWK, some software from the 1.8 series dependencies must be uninstalled before installing the REDHAWK 1.9 series. The following procedure explains how to uninstall the software.

1. Enter the following command:

```
$ sudo yum erase \
  libomniORB4.1 omniORB-debuginfo omniORB-doc \
  libomniORBpy3 omniORBpy-debuginfo omniORBpy-doc \
  omniEvents-doc omniEvents-debuginfo \
  apache-log4cxx apache-log4cxx-debuginfo
```

2. Additionally, if you are using a Redhat/CentOS 5 system, enter the following command:

```
$ sudo yum downgrade boost
```

   This restores the system-provided boost (version 1.33).

Regenerating 1.8 Components

If you generate an existing 1.8 Component in the 1.9 IDE, the IDE prompts you to upgrade the Component to use the new 1.9 Code Generator. If you select **Upgrade**, hidden project files are modified to indicate the new 1.9 Code Generator has been used. Once these project files are modified, it is difficult to revert back to use the 1.8-style Code Generators for this Component. If you select **Cancel**, the source code for the Component is not modified.

Upgrading to the new 1.9 Code Generator offers the following benefits:

- New component pattern shifts BulkIO implementation from code generator to base classes to ease incorporating future BulkIO-related bug fixes and enhancements.
- New code generators support use of complex properties.

- New code generators are now easily accessed from command line.
- New Java component pattern mirrors the existing Python and C++ component model with a generated base class and a serviceFunction in the main class.

Note: If the main Java component class is not regenerated, the class does not take advantage of the newly generated base class and needs to have a method added to it in order for the base class to compile when building the project. If the main Java component class is regenerated, all custom code added to it is not preserved. For more information, in the REDHAWK Manual, refer to Files Generated for Java Components.

Launching Components from the Sandbox

Prior to REDHAWK 1.9, Components were launched by directly creating an instance of the `Component` class. This interface is still supported for backwards compatibility, but is deprecated and may be removed in a future release.

BulkIO Code Generation

In prior releases of REDHAWK, component code generation provided the interface classes and algorithms for ingress and egress of data via the pushPacket/pushSRI interfaces. This code generation capability has matured to a stable state and was extracted to form the basis of the `bulkio` base class library. This new shared library provides the following benefits:

- reduces generated code for every component
- reduces memory footprint for a component using the shared library
- easily distributes changes to `bulkio` classes through a common library
- continues to allow for customization/specialization via class inheritance

In version 1.8, the data flow implementation for a Component's BulkIO Port interface is provided by the component code generation process. The result of this process is code that resides in the component directory and is compiled and installed for each component you deploy.

In version 1.9, the data flow implementation for a Component's BulkIO Port interface is provided by a shared `bulkio` base class library. The resulting component code instantiates a `bulkio` base class object and makes use of the shared library during deployment and execution.

BulkIO Time Stamps

The following code segment provides an example of how to construct a `BULKIO::PrecisionUTCTime` time stamp to be sent in the SRI:

```
/**
 *  version 1.8 example to create a time stamp from the current time of day
 */
//To create a PrecisionUTCTime object, use the following
//code:
struct timeval tmp_time;
struct timezone tmp_tz;
gettimeofday(&tmp_time, &tmp_tz);
double wsec = tmp_time.tv_sec;
double fsec = tmp_time.tv_usec / 1e6;
BULKIO::PrecisionUTCTime tstamp =
   BULKIO::PrecisionUTCTime();
tstamp.tcmode = BULKIO::TCM_CPU;
tstamp.tcstatus = (short)1;
tstamp.toff = 0.0;
tstamp.twsec = wsec;
tstamp.tfsec = fsec;


/**
  * version 1.9 example to create a time stamp from the current time of day
  */

 BULKIO::PrecisionUTCTime tstamp = bulkio::time::utils::now();
```