# OpenCPI
# REDHAWK Export Guide

How to Export OpenCPI Applications and Components for Execution in a REDHAWK environment (see redhawksdr.org)

*Revision History*

| Revision | Description of Change | Date |
|---|---|---|
| 1.0 | Initial Version | 2017-03-23 |
| 1.1 | Update for 2017.Q2 branch | 2017-07-28 |

## Table of Contents

# 1 References

This document depends on the OpenCPI Application Development Guide. For information on component development, which is *not* a prerequisite of this document, see the OpenCPI Component Development Guide (CDG).

*Table 1 - Table of Reference Documents*

| Title | Published By | Link |
|---|---|---|
| OpenCPI Application Development Guide | OpenCPI | Public URL: https://github.com/opencpi/opencpi/raw/2017.Q2/doc/pdf/OpenCPI_Application_Development.pdf |
| OpenCPI Overview | OpenCPI | Public URL: https://github.com/opencpi/opencpi/raw/2017.Q2/doc/pdf/OpenCPI_Overview.pdf |
| OpenCPI Component Development Guide | OpenCPI | Public URL: https://github.com/opencpi/opencpi/raw/2017.Q2/doc/pdf/OpenCPI_Component_Development.pdf |

# 2 Overview

The purpose of this document is to describe a tool, **`ocpirh_export`**, that exports OpenCPI components and applications for execution as components in a REDHAWK (RH) environment. This tool is part of OpenCPI. The user of this tool must be familiar with application development in OpenCPI, as it is based on, and analogous to preparing OpenCPI applications for execution in the OpenCPI runtime environment.

REDHAWK is a software-defined radio (SDR) framework, available at redhawksdr.org. It is a software component-based framework based on Eclipse, CORBA and the Software Communication Architecture (SCA) as defined at http://www.public.navy.mil/JTNC/SCA, with some modifications and enhancements.

The capability described in this document is a specific bridge between the two environments that facilitates exactly one of many possible interaction and integration use cases between the OpenCPI and REDHAWK frameworks. Namely:

- An OpenCPI component or application developer needs to provide a package to a RH developer such that the RH developer can install the OpenCPI component or application as a component in the RH environment.
- The OpenCPI developer *does not need to install or learn about* RH to provide this service.
- The RH developer *does not need to learn about* (but must install), the OpenCPI environment.

The preparation for this export scenario is very similar to the preparation to execute any OpenCPI application: component implementations must be built, available as executable artifacts, and discoverable via the **`OCPI_LIBRARY_PATH`** environment variable. Except when exporting a single OpenCPI component, the OpenCPI application must be prepared and defined in an XML file called an OpenCPI Application Specification (OAS).

The result of this export operation is a single package file that is in the normal format used to export a component from one RH environment to another (called an RPM package). It can be sent to a RH user as an email attachment or any other single file transfer method.

The export package is heterogeneous, meaning that the package contains artifacts that allows the OpenCPI components in the application to execute on any type of OpenCPI container available in the OpenCPI installation on the RH system. This enables both applications using a mix of workers targeting different processors as well as multiple, alternative execution scenarios where multiple artifacts are provided for the same component. The RH installation must be one of the RH supported platforms, currently CentOS6 or CentOS7.

The OpenCPI developer prepares the OpenCPI application to execute in a number of deployment scenarios (types of containers and artifacts), and then uses the **`ocpirh_export`** tool to prepare the RH component package (RPM file) to transfer to the RH user/developer.

# 3 Preparing the Application for Export

As mentioned above, components or application may be exported from OpenCPI to RH. Exporting a *component* means exporting an application consisting of that single component, with its properties taking default values, and all its ports made external. Thus exporting a component is just a shortcut for exporting a simple application based on that component, with all its ports declared as external using the **`externals`** attribute in the single **`instance`** element of the OAS.

### *3.1 Defining the External Ports of the Application*

External ports are designated in the OAS XML file using one of these methods:

- the **`externals`** (plural) boolean attribute of an instance specifies that all its unconnected ports should be considered external to the application, using the instance's port names, e.g.:

```
<application>
   <instance component='ocpi.bias' externals='true'/>
</application>
```

- the **`external`** (singular) string attribute of an instance can designate that a single port of the instance be external, using the instance's port name.

```
<application>
   <instance component='c1' external='input' connect='c2'/>
   <instance component='c2' external='output'/>
</application>
```

- a **`connection`** element in the application can designate that a port of an instance should be considered external, using a specified name that may be different than the instance's port name., e.g.

```
<application>
   <instance component='c1' connect='second'/>
   <instance component='c2'/>
   <connection>
      <external name='extIn'/>
      <port name='in' instance='c1'/>
   </connection>
   <connection>
      <external name='extOut'/>
      <port name='out' instance='c2'/>
   </connection>
</application>
```

External ports of the OpenCPI application being exported become the ports of RH component when it is imported. The OpenCPI port protocol is translated into a RH-supported protocol (defined by CORBA IDL) so that the exported application can interoperate with other RH components via ports. This translation has limitations since not all OpenCPI protocols can be translated into a supported RH protocol. This translation capability will be expanded over time, but is currently limited to protocols with a single message (operation) carrying a sequence or array of scalar values. Any of the OpenCPI scalar types is appropriately translated into the related RH “bulkIO” protocols.

### *3.2 Executing the Application or Component Prior to Export*

Before exporting the application, it should be executed locally using **`ocpirun`** to ensure it executes correctly. Once the application is verified to run correctly, it can be exported.

To execute an application with external ports using **`ocpirun`**, they can be redirected to files using the **`-f`** or **`--file`** option to **`ocpirun`**, e.g.:

```
ocpirun –f in=test.input -f out=test.output myapp.xml
```

This redirection simply uses the **`file_read`** and **`file_write`** utility components to produce data for external input ports and consume data from external output ports. The syntax of these redirection options is the same as the HTTP URL query syntax in that the file name can be followed by a **`?`** to indicate further ***`<name>=<value>`*** options, separated by **`&`**. In this case the options are property values supplied to the **`file_read`** and **`file_write`** components. The most commonly used properties for this purpose are the **`messagesInFile`** boolean property (indicating that the files contain explicitly delimited and opcode-tagged messages), and the **`messageSize`** attribute indicating the size of messages in bytes when reading or writing raw message data without any delimiters or opcode tags. An example where an input file has been previously written using the **`messagesInFile`** option, might be:

```
ocpirun –f in=test.input\?messagesInFile=true \
        -f out=test.output myapp.xml
```

Notice that the **`?`** character is escaped for the shell command line.

The file redirection options are used to verify the correct behavior of the application at its external ports. When actually exporting the application, they are not used at all.

When the application is working under these conditions, the other preparatory step before exporting is to decide which OpenCPI deployments are to be enabled in the exported package. A ***deployment*** is a file that is optionally written when an application is executed, using the **`--deploy-out`** option to **`ocpirun`**. The file captures, for each component in the application, which artifact file was used to execute each component. Another (boolean) option, **`--no-execute`**, specifies that **`ocpirun`** should do all the work to figure out how to execute the application (doing lots of error checking), and then stop before the execution actually takes place. By combining these two options, a deployment file is created that captures one way that the application could run.

If no deployment files are supplied to the export process, one is automatically created by using these two options, creating a default deployment. When **`--no-execute`** is specified, external ports do not need to be redirected at all.

So to prepare for export, testing is done with **`ocpirun`** using file redirection (or possibly with a custom ACI application), and if needed, a set of deployment files should be created if there is more than one way the application should be run when exported (i.e. on different platforms or using different artifacts).

When the export function is run (using the ocpirh_export command described below), it will use the application XML file (unless only a component is being exported), optionally accompanied by a set of deployment files.

# 4 Using the `ocpirh_export` Tool

This tool is enabled when OpenCPI is built from source as described below. It's primary input is either an OpenCPI application XML file (OAS) or the name of a component (usually with a package prefix). Its primary output is an installable RH package file (an RPM file). Without any options the RH package file is created in the current directory.

When this tool runs, it does not perform any compilation or linking since it relies on the existence of OpenCPI artifact files that have resulted from building component implementations (workers). *It does not rely on any RH software being installed.*

The syntax is:

**`ocpirh_export [`*`options`*`] <`*`app-file-or-component-name`*`> [<`*`depfiles`*`>...]`**

The output file will be in the current directory, and its name will start with the component or application name, and have a **`.rpm`** suffix. The options are in the table below. Boolean options have no value. String options have a value in the argument following the option designation.

*Table 2: Options to* ***`ocpirh_export`***

| Letter | Datatype | Description |
|---|---|---|
| **`d`** | String | The name of a directory where the resulting package file will be created. The default is the current directory where the command was run. |
| **`t`** | String | The staging directory used during the export process. The default will be a temporary directory created by the command. The directory and its contents will be removed when the comment completes unless the **`-k`** option is specified. |
| **`k`** | Bool | Keep the contents of the staging directory after the command completes in cases where failures need to be diagnosed. |
| `c` | Bool | Indicates that the argument after *[options]* will be treated as a component name (with optional package prefix as in OAS **`instance`** elements). If **`-c`** is not specified, the argument will be processed as an OAS XML file. |
| **`i`** | Bool | Install the resulting RH package RPM file on the local system, assuming there is a RH installation present. The file will still exist after installation. |
| **`p`** | String | The RH package prefix to use for the resulting package. This is not necessarily an OpenCPI package name. It will set the grouping name in the RH IDE, the directory where the package will be installed under **`/var/redhawk/sdr/dom/components`**, and the prefix on the name of the resulting package file. |
| **`f`** | String | Specifies a file that should be included in the package that can be used by the exported application. This option can occur multiple times. |

As an example, the following command would create a RH package for the **ocpi.bias** component, put the resulting RPM file in the **myrh** directory, assign this component in the RH group **mycomps**, and add the file **mydata** to the package so that it is available for the component to access when it runs. Before the command completes it will immediately install the resulting RM package file in the local RH installation.

```
ocpirh_export -c -d myrh -p mycomps -f mydata -i ocpi.bias
```

The **mydata** file will be accessible from the current working directory of the component when it executes.

Another example below, would export the **myapp** OpenCPI application, and ensure that it will be able to run deployments as indicated in the **dep1** and **dep2** XML files. It would use the **mytmp** directory as a staging area, and keep its contents intact for analysis of the export process.

```
ocpirh_export -t mytmp -k myapp dep1 dep2
```

# 5 Receiving and Using the Exported Package in REDHAWK

The RH package file (an RPM file) can be installed into a RH environment in several ways. The primary method is to use the **yum** command, e.g.:

```
yum install mycomps.bias.1.0.0.rpm
```

RPM files can also be installed while in the IDE using the Open->File menu item, which will ask whether the RPM should be installed.

In both cases, if the RH IDE is running when the installation takes place it must be restarted to recognize the existence of the newly installed components. This can be done by existing and starting the IDE, or by using the File->Restart menu option in the IDE.

The newly imported components will appear in the IDE under the “components” pallet, under the category with the package name indicated during export. The components that are built in to the RH system are found under the **rh** category. So if a number of components or applications are being exported, they can all be part of such a grouping when imported.

The following screen image shows the components hierarchy in two different places in the RH IDE. In both, the **rh** group shows the built-in components that come with RH, and the other three result from installing exported components and applications in the **ocpi**, **ocpipkg**, and **pkgocpi** groups. These groups are determined at the time of export.

# 6 Building the `ocpirh_export` tool in an OpenCPI Source Installation

Using this **`ocpirh_export`** tool does not require a RH installation to be present at all. However, to create this tool in a source distribution of OpenCPI, at least a partial RH installation is required. If a complete RH installation has already been performed on the system running OpenCPI, then the standard build of OpenCPI from source will build this tool and the internal assets used by it.

If there is no RH installation on the system and no need for one, then a minimal installation can be performed by the **`scripts/install-redhawk.sh`** script in the OpenCPI source tree. The resulting installation is sufficient to build the **`ocpirh_export`** tool (and its internal assets used at the time of export), but is not otherwise sufficient for *using* RH. Once the build process is complete, even this minimal RH installation is unnecessary.

This minimal subset RH installation is still installed globally on the system. This is different from other prerequisite package installations that are normally installed in an OpenCPI-specific sandbox area that does not affect or provide a global installation of those packages. If a complete RH installation is performed later it will use the preinstalled subset assuming the versions match.

When at the root of the OpenCPI source tree, the command:

```
./scripts/install-redhawk.sh
```

will install the subset after copying various files from their network download location.

Then the normal source build commands as described in the OpenCPI installation document will also build this tool. If the RH installation (full or partial) is done later, then rerunning the OpenCPI build commands will then build this tool even if it was not built prior to the RH installation.