# Generic RF Interface Guide

Version 1.4

***Revision History***

| Revision | Description of Change | Date |
|---|---|---|
| v1.1 | Initial Release | 3/2017 |
| v1.2 | Updated for Release 1.2 | 8/2017 |

# Table of Contents

# 1 References

This document assumes a basic understanding of the Linux command line (or "shell") environment. It requires a working knowledge of OpenCPI and the OCPI ACI. The reference(s) in Table 1 can be used as an overview of OpenCPI and may prove useful.

Table 1: References

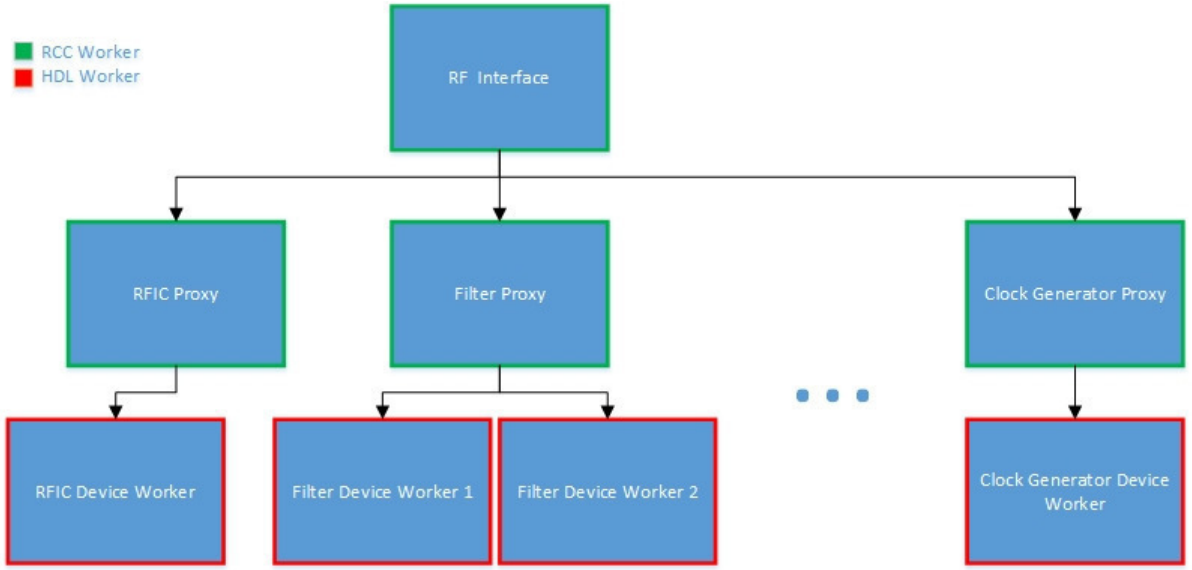
| Title | Published By | Link |
|---|---|---|
| Getting Started | ANGRYVIPER Team | `Getting_Started.pdf` |
| Installation Guide | ANGRYVIPER Team | `RPM_Installation_Guide.pdf` |
| Acronyms and Definitions | ANGRYVIPER Team | `Acronyms_and_Definitions.pdf` |
| Overview | ANGRYVIPER Team | `http://opencpi.github.io/Overview.pdf` |

# 2 Overview

In OpenCPI, it is the intention that an application developer won't need to care or know about the Radio Interface(RF) interfaces. Without the ability for generic RF interfaces, applications can not be seamlessly moved from one OpenCPI-supported platform to another OpenCPI-supported platform. This document describes the generic RF interface components that have been developed as part of the OpenCPI release.

These interface components are not intended to cover all features of all receivers and transmitters. They are intended to have the minimum amount of functionality that all receivers and transmitters will have. Any extra functionality on a receiver or transmitter can be added at the worker implementation level.

Each of the RF interfaces will be setting the properties of several other workers referred to as slaves. In some cases these slaves will also have their own slaves as well. Any application that uses one of the RF Interfaces needs to also include all of the required slaves and slaves of any of the slaves' slaves. A block diagram of this relationship is as follows:



# 3 Control Interface

Each setting has a max, min, and step value associated with it. Where the max is the highest possible value, min is the lowest possible value, and step is the minimum granularity for changes of the associated setting. These associated properties are available to be used by application developers for reading back information about the functionality of the interface during runtime. Both receive and transmit use different spec files but have the same property sets. The component specification file locations are as follows:

| | |
|---|---|
| Receive | core/specs/rx_spec.xml |
| Transmit | core/specs/tx_spec.xml |

The properties that are described in these spec files can be observed in the following table. For more details on these properties check the matchstiq_rx or matchstiq_tx data sheets.

| Name | Usage |
|---|---|
| `rf_gain_dB` | The runtime-configurable value in dB of the RF gain stage of the front-end receiver. |
| `rf_gain_max_dB` | Maximum valid value for RF gain setting in dB. This value represents the datasheet-specified hardware limitations of the front-end's receiver, and therefore is buildtime-configurable only (i.e. it is a parameter). This property is intended to prevent the worker from re-configuring at runtime the RF amplifiers to have an invalid gain value. |
| `rf_gain_min_dB` | Minimum valid value for RF gain setting in dB. This value represents the datasheet-specified hardware limitations of the front-end's receiver, and therefore is buildtime-configurable only (i.e. it is a parameter). This property is intended to prevent the worker from re-configuring at runtime the RF amplifiers to have an invalid gain value. |
| `rf_gain_step_dB` | Minimum granularity for the RF gain setting in dB. This value represents the datasheet-specified hardware limitations of the front-end's receiver, and therefore is buildtime-configurable only (i.e. it is a parameter). This property serves two purposes: 1) to provide the end user with the knowledge of how the value applied to rf_gain_dB will be rounded, and 2) if necessary, to provide the worker implementation that is performing the rounding with the information necessary to do so. |
| `bb_gain_dB` | The runtime-configurable value in dB of the baseband gain stage of the front-end receiver. |
| `bb_gain_max_dB` | Maximum valid value for baseband gain setting in dB. This value represents the datasheet-specified hardware limitations of the front-end's receiver, and therefore is buildtime-configurable only (i.e. it is a parameter). This property is intended to prevent the worker from re-configuring at runtime the baseband amplifiers to have an invalid gain value. |
| `bb_gain_min_dB` | Minimum valid value for baseband gain setting in dB. This value represents the datasheet-specified hardware limitations of the front-end's receiver, and therefore is buildtime-configurable only (i.e. it is a parameter). This property is intended to prevent the worker from re-configuring at runtime the baseband amplifiers to have an invalid gain value. |
| `bb_gain_step_dB` | Minimum granularity for the baseband gain setting in dB. This value represents the datasheet-specified hardware limitations of the front-end's receiver, and therefore is buildtime-configurable only (i.e. it is a parameter). This property serves two purposes: 1) to provide the end user with the knowledge of how the value applied to bb_gain_dB will be rounded, and 2) if necessary, to provide the worker implementation that is performing the rounding with the information necessary to do so. |
| `frequency_MHz` | The runtime-configurable value in MHz for the tuned center frequency of the front-end receiver. |

| | |
|---|---|
| `frequency_max_MHz` | Maximum valid value for frequency setting in MHz. This value represents the datasheet-specified hardware limitations of the front-end's receiver, and therefore is buildtime-configurable only (i.e. it is a parameter). This property is intended to prevent the worker from re-configuring at runtime the front-end LO to have an invalid frequency value. |
| `frequency_min_MHz` | Minimum valid value for frequency setting in MHz. This value represents the datasheet-specified hardware limitations of the front-end's receiver, and therefore is buildtime-configurable only (i.e. it is a parameter). This property is intended to prevent the worker from re-configuring at runtime the front-end LO to have an invalid frequency value. |
| `frequency_step_MHz` | Minimum granularity for the frequency setting in MHz. This value represents the datasheet-specified hardware limitations of the front-end's receiver, and therefore is buildtime-configurable only (i.e. it is a parameter). This property serves two purposes: 1) to provide the end user with the knowledge of how the value applied to frequency_MHz will be rounded, and 2) if necessary, to provide the worker implementation that is performing the rounding with the information necessary to do so. |
| `sample_rate_MHz` | The runtime-configurable sample rate of the front-end's ADC in MHz (MSps). |
| `sample_rate_max_MHz` | Maximum valid value for front-end ADC's sample rate setting in MHz (MSps). This value represents the datasheet-specified hardware limitations of the front-end's ADC, and therefore is buildtime-configurable only (i.e. it is a parameter). This property is intended to prevent the worker from re-configuring at runtime the ADC to have an invalid sample rate. |
| `sample_rate_min_MHz` | Minimum valid value for front-end ADC's sample rate setting in MHz (MSps). This value represents the datasheet-specified hardware limitations of the front-end's ADC, and therefore is buildtime-configurable only (i.e. it is a parameter). This property is intended to prevent the worker from re-configuring at runtime the ADC to have an invalid sample rate. |
| `sample_rate_step_MHz` | Minimum granularity for the ADC sample rate setting in MHz (MSps). This value represents the datasheet-specified hardware limitations of the front-end's ADC, and therefore is buildtime-configurable only (i.e. it is a parameter). This property serves two purposes: 1) to provide the end user with the knowledge of how the value applied to sample_rate_MHz will be rounded, and 2) to provide the worker implementation that is performing the rounding with the information necessary to do so. |
| `rf_cutoff_frequency_MHz` | The cutoff frequency in MHz for any filtering that is done in the RF stage of the front-end receiver. |
| `rf_cutoff_frequency_max_MHz` | Maximum valid value for RF cutoff frequency setting in MHz. This value represents the datasheet-specified hardware limitations of the front-end's receiver, and therefore is buildtime-configurable only (i.e. it is a parameter). This property is intended to prevent the worker from re-configuring at runtime the RF stage's filter(s) to have an invalid cutoff frequency. |
| `rf_cutoff_frequency_min_MHz` | Minimum valid value for RF cutoff frequency setting in MHz. This value represents the datasheet-specified hardware limitations of the front-end's receiver, and therefore is buildtime-configurable only (i.e. it is a parameter). This property is intended to prevent the worker from re-configuring at runtime the RF stage's filter(s) to have an invalid cutoff frequency. |
| `rf_cutoff_frequency_step_MHz` | Minimum granularity for the RF cutoff frequency setting in MHz. This value represents the datasheet-specified hardware limitations of the front-end's receiver, and therefore is buildtime-configurable only (i.e. it is a parameter). This property serves two purposes: 1) to provide the end user with the knowledge of how the value applied to rf_cutoff_frequency_min_MHz will be rounded, and 2) to provide the worker implementation that is performing the rounding with the information necessary to do so. |

| | |
|---|---|
| `bb_cutoff_frequency_MHz` | The cutoff frequency for any filtering that is done in the baseband stage of the front-end receiver. |
| `bb_cutoff_frequency_max_MHz` | Maximum valid value for baseband cutoff frequency in MHz. This value represents the datasheet-specified hardware limitations of the front-end's receiver, and therefore is buildtime-configurable only (i.e. it is a parameter). This property is intended to prevent the worker from re-configuring at runtime the baseband RF stage's filter(s) to have an invalid cutoff frequency. |
| `bb_cutoff_frequency_min_MHz` | Minimum valid value for baseband cutoff frequency in MHz. This value represents the datasheet-specified hardware limitations of the front-end's receiver, and therefore is buildtime-configurable only (i.e. it is a parameter). This property is intended to prevent the worker from re-configuring at runtime the baseband stage's filter(s) to have an invalid cutoff frequency. |
| `bb_cutoff_frequency_step_MHz` | Minimum granularity for the baseband cutoff frequency setting in MHz. This value represents the datasheet-specified hardware limitations of the front-end's receiver, and therefore is buildtime-configurable only (i.e. it is a parameter). This property serves two purposes: 1) to provide the end user with the knowledge of how the value applied to bb_cutoff_frequency_min_MHz will be rounded, and 2) to provide the worker implementation that is performing the rounding with the information necessary to do so. |

# 4 Creating an Application using RF Interfaces

As stated earlier, an application that uses an RF Interface Worker needs to also have any of the slave workers and slave of slave workers also declared in the application XML. This may cause a different application XML per platform to declare the different dependencies per platform. This is a limitation of the framework and will likely be fixed in a future release. An example of how this will work is in both of the ANGRYVIPER Team's Reference applications:

| | |
|---|---|
| FSK app | assets/applications/FSK |
| RX app | assets/applications/rx_app |

## 4.1 Application XML Example

There are two ways that application developers can control the RF interface workers. The first of which is to use the application XML with the ocpirun utility to set the properties as needed. This is the easier of the two methods and works as long as the application doesn't need runtime-dynamic property control or user interaction to set initial properties.

```
<Application>
  <Instance component='ocpi.core.devices.rx'>
    <Property Name="bb_gain_dB"              Value="-5"/>
    <Property Name="rf_gain_dB"              Value="10"/>
    <Property Name="frequency_MHz"           Value="2400"/>
    <Property Name="sample_rate_MHz"         Value="5"/>
    <Property Name="bb_cutoff_frequency_MHz" Value="2.5"/>
    <Property Name="rf_cutoff_frequency_MHz" Value="2.5"/>
  </Instance>
  <Instance component='ocpi.core.devices.tx'>
    <Property Name="bb_gain_dB"              Value="-5"/>
    <Property Name="rf_gain_dB"              Value="10"/>
    <Property Name="frequency_MHz"           Value="2400"/>
    <Property Name="sample_rate_MHz"         Value="5"/>
    <Property Name="bb_cutoff_frequency_MHz" Value="2.5"/>
    <Property Name="rf_cutoff_frequency_MHz" Value="2.5"/>
  </Instance>

  ...

  The slaves of the RF interfaces and the slaves of the slaves

  ...

  The rest of the application and device workers required for the application

  ...

<Application>
```

## 4.2 Application Control Interface Example

The second way that an application developers can control the RF interface workers is via a C++ program that uses the Application Control Interface library which is provided with the OpenCPI framework. This is required when the application user needs to interact with the application to change properties or if the properties of the workers need to change during a run of the application.

```
...
setup application object
...

double freq = 1000;
app.getProperty("rx","frequency_min_MHz", value);
double rx_frequency_min_MHz = atof(value.c_str());
app.getProperty("rx","frequency_max_MHz", value);
double rx_frequency_max_MHz = atof(value.c_str());
if (bb_bw < rx_frequency_min_MHz || freq > rx_frequency_max_MHz)
{
   printf("Error: invalid freq. setting to a default value. value: %f min: %f max: %f\n",
           freq, rx_frequency_min_MHz, rx_frequency_max_MHz);
   app.setProperty("rx","frequency_MHz", "2400");
}

...
start application or make other decisions based on settings passed in
...
```

# 5 Implementation of a RF Interface Worker

BSP Developers will need to add implementations of these RF interfaces when you want to add a new radio or RFIC card. This section gives a brief walkthrough of how this is done and how it is different then normal RCC worker development. These RF interfaces are just RCC workers with a few caveats, the interface workers generally have the ability to set the properties of several other workers and don't have a run function.

The framework has the ability to have a one to one slave master relationship between workers but there is no defined way to have multiple slaves. The way workers can access multiple slaves is by using the ACI to access application properties. a worker has the ability to access the application object that it is a part of. This object is then used to access any properties within the application, so there is a lot of freedom.

```
OA::Application &app  = getApplication();
app.setProperty("rf_rx_proxy", "lpf_bw_hz", "750000")
```

In this code snippet the application object is provided using the getApplication call and it is used to set the lpf_bw_hz property of a rf_rxproxy component to 750000 Hz. If there is not a rf_rx_proxy component in the application this call will throw an exception and the application will crash.

The framework also has the ability to call functions in the worker code before or after its properties are written by the container. This is done in the following way within the worker:

```
OWD(can't be in OWS):
   <specproperty name='frequency_MHz' writesync='1' default='500'/>

C++:
RCCResult frequency_MHz_written()
{
   ...
   do work based on the property change.
   ...

   return RCC_OK;
}
```

The following workers are good examples of how to create RF Interface workers:

| | |
|---|---|
| matchstiq_rx | assets/hdl/platforms/matchstiq/devices/matchstiq_rx.rcc/ |
| matchstiq_tx | assets/hdl/platforms/matchstiq/devices/matchstiq_tx.rcc/ |