

# Updating a Component to Operate with OSSIE 0.6.2

Matt Carrick and Carl Dietrich  
Virginia Tech, Blacksburg, VA  
2 March 2008

## Introduction:

This guide will walk through updating a component from a previous release to the latest release, 0.6.2. This is relatively simple for components that currently run with OSSIE 0.6.1, as the major changes that affect components are to the directory structure. For components that run with other versions of OSSIE, more extensive changes may be required. One approach is to use the OSSIE waveform developer (OWD) to generate 0.6.2 skeleton code for the component and copy the signal processing code from the old component to the new component. The component that will be used in this guide is WFMDemod, a digital implementation of a FM Demodulator written by Philip Balister. The code is written for a version of OSSIE other than 0.6.2 and the procedure will walk through the necessary steps to make it compatible with the latest OSSIE release. The original code for the component is listed at the end of this guide in Appendix A, while the updated code has been listed in Appendix B. This guide also assumes familiarity with OSSIE, OWD and basic Linux commands. For additional help, refer to the OSSIE 0.6.2 User Guide at: <http://ossie.wireless.vt.edu/trac>

## Guide Outline:

1. Have a copy of the component you want to edit.
  - a. If the component runs with OSSIE 0.6.1, change the paths in the SPD file and, if needed, change the configure.ac file and build the component. These changes may be sufficient for the component to run with 0.6.2. Otherwise:
  - b. Find the component's port names and types in the SCD file.
  - c. Find the component's property names and values (if any) in the PRF file.
2. Re-create the component in OWD
  - a. Name the component and enter a description.
  - b. Create the ports with the same port types.
  - c. Add the property values.
  - d. Generate the component.
3. Update Code to conform with 0.6.2 Template
  - a. Copy signal processing code from the old version into the new 0.6.2 version.
  - b. Update old variable names
  - c. Include necessary libraries
4. Compile the code
  - a. Re-edit code if necessary
5. Install the Component
6. Put the component into a waveform to verify its operation.

## Procedure:

Procedures are described for modifying OSSIE 0.6.1 components to work with OSSIE 0.6.2 and an example is given of an approach for modifying components developed for other versions of OSSIE to work with version 0.6.2.

## **Modifying OSSIE 0.6.1 components to work with OSSIE 0.6.2**

OSSIE 0.6.2 uses a new directory structure that places component and waveform files under /sdr instead of /home/sca as in OSSIE 0.6.1.

1. OSSIE 0.6.1 components can be modified to work with OSSIE 0.6.2 by placing their xml files in /sdr/xml/<componentname>/ (instead of /home/sca/xml/<componentname>/) and binary files in /sdr/bin (instead of /home/sca/bin).
2. The paths specified in the <componentname>.spd.xml file will need to be edited, deleting the leading "../.." for each path (scd, spd, and executable).
3. If the above steps do not work or if the component is still under development and will be recompiled, it will be necessary to edit the configure.ac file in the component's source directory and recompile the component. The components supplied with the ossie\_demo waveform (TxDemo, ChannelDemo, and RxDemo) can be used as a guide.

## **An approach for modifying components that run with OSSIE versions other than 0.6.1**

One of the possible approaches is shown here. It involves regenerating skeletal source code for the component using the OSSIE waveform developer (OWD) and copying functional code from the original component. This approach is generally applicable. However, it results in a component with the default port implementations, which may not be desired in all cases. An example is given but some details may vary depending on the version of OSSIE with which the original component works, and on the specific component implementation.

1. The example component to be updated is WFMDemod, and the code is provided in Appendix A. The important sections of the code are highlighted in red so it is easier to find.
  - a. The SCD file contains, among other things, the names of the ports and their types. In the highlighted section of WFMDemod.scd.xml, look for the following two lines:

```
<provides providesname="dataIn"  
repid="IDL:standardInterfaces/complexShort:1.0">  
  
<uses repid="IDL:standardInterfaces/complexShort:1.0"  
usesname="dataOut">
```

The provides port is named `dataIn`, the uses port is named `dataOut`, and types are both `complexShort`. This information will be used to re-create the component in OWD.

- b. The PRF file contains information on property values the component uses. In `WFMDemod.prf.xml`, the only property listed is a dummy property which is not used.

```
<value>1</value>
<description>Dummy</description>
```

Since no real properties are in the file, none will need to be added to the component definition in OWD. If there had been real properties, for example say the gain of an amplifier, then the name of the property would need to be noted as well as the default value and type.

2. Now that the basic definition of the component has been obtained, the new 0.6.2 version of the component needs to be generated. Start OWD, and within OWD open the Component Editor.
  - a. First, the name of the component needs to be entered. The name `WFMDemod_update` will be used as an example. Also enter a description describing the basic operation of the component.
  - b. The ports now need to be created. In the component editor, select Add Port, select the interface, provides or uses, and enter the port name. In this example, enter the port name `dataIn`, select port type `Provides`, and select `complexShort` under `standardInterfaces`. Now for the other port, enter the name `dataOut`, select port type `Uses`, and select `complexShort` under `standardInterfaces`.
  - c. At this point the properties would be added. To do this, select the Add Property button, enter the name, description and default value for the property. However, for this example there are no properties that need to be added so this can be skipped.
  - d. Now generate the code for the component into a new directory.
3. The basic shell for the updated component has been created, now the signal processing code must be inserted.
  - a. In the original copy of `WFMDemod.cpp`, find the signal processing code which needs to be placed into the newly created 0.6.2 version. In the highlighted code, the `ProcessData()` function calls `Demod_Lyons()`, which performs the FM demodulation. Within `Demod_Lyons()`, select the code within the while loop:

```
for (unsigned int i(0); i < len; ++i) {
...
...
// Update delay terms;
I2 = I1;
I1 = I0;
Q2 = Q1;
Q1 = Q0;
```

Copy this code into the portion of the newly created WFMDemod\_update.cpp where it says `/*insert code here to do work*/`. WFMDemod\_update.cpp can be found in Appendix B.

- b. Now the variable names need to be reconciled between the two versions. First find the lines in WFMDemod\_update.cpp:

```
I_out_0.length(); //must define length of output
Q_out_0.length(); //must define length of output
```

Update the lines to define the length of the output:

```
I_out_0.length(I_in_0_length); //must define length of
output
Q_out_0.length(Q_in_0_length); //must define length of
output
```

The next problem is the old variable names. In the version of OSSIE that WFDemod was originally written for, the input variables were `I_in` and `Q_in`, however with 0.6.2 they have been renamed to `I_in_0` and `Q_in_0`. Another example is the output variables are now called `I_out_0` and `Q_out_0` instead of `I_out` and `Q_out`. All instances of the old variables need to be updated.

The variables `I1`, `I2`, `Q1`, `Q2` also need to be declared. Outside of the while loop, enter the line:

```
int I1, I2, Q1, Q2;
```

- c. The demodulation process calls the `sqrt()` function, which is listed in the `math.h` library. Add the library at the top of the file:

```
#include <math.h>
```

Note: Please refer to Appendix B at the end of this guide for a print out of the code with all appropriate changes made.

Note: The number of variables and libraries that need to be changed or added in the new 0.6.2 version of a component depend on the code that is being updated. The number of changes may be more or less than what is listed in this guide.

4. Now compile the code and install the component.

- a. Run the commands:

```
./reconf
./configure
make
```

After the `make` command, check to see if there have been any compile errors.

5. Now install the waveform by running the command: `$ su -c "make install"`
6. The component is now installed. To test the functionality of the component, add it to a waveform in OWD and then run it in ALF. For additional help on creating and running a waveform, refer to Labs 1 – 3 on the OSSIE website (<http://ossie.wireless.vt.edu>). Also refer to Labs 4 – 5 for help with using the USRP.

## Appendix A:

### WFMDemod.scd.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE softwarecomponent SYSTEM "../dtd/softwarecomponent.dtd">
<!--Created with OSSIE WaveDev Beta Version 0.6.2-->
<!--Powered by Python-->
<softwarecomponent>
  <corbaversion>2.2</corbaversion>
  <componentrepid repid="IDL:CF/Resource:1.0"/>
  <componenttype>resource</componenttype>
  <componentfeatures>
    <supportsinterface repid="IDL:CF/Resource:1.0" supportsname="Resource"/>
    <supportsinterface repid="IDL:CF/LifeCycle:1.0" supportsname="LifeCycle"/>
    <supportsinterface repid="IDL:CF/PortSupplier:1.0"
      supportsname="PortSupplier"/>
    <supportsinterface repid="IDL:CF/PropertySet:1.0" supportsname="PropertySet"/>
    <supportsinterface repid="IDL:CF/TestableObject:1.0"
      supportsname="TestableObject"/>
    <ports>
      <provides providesname="dataIn"
        repid="IDL:standardInterfaces/complexShort:1.0">
        <porttype type="data"/>
      </provides>
      <uses repid="IDL:standardInterfaces/complexShort:1.0"
        usesname="dataOut">
        <porttype type="data"/>
      </uses>
    </ports>
  </componentfeatures>
  <interfaces>
    <interface name="Resource" repid="IDL:CF/Resource:1.0">
      <!--[Inherited interface IDL:CF/LifeCycle:1.0]-->
      <inheritsinterface repid="IDL:CF/LifeCycle:1.0"/>
      <!--[Inherited interface IDL:CF/PortSupplier:1.0]-->
      <inheritsinterface repid="IDL:CF/PortSupplier:1.0"/>
      <!--[Inherited interface IDL:CF/PropertySet:1.0]-->
      <inheritsinterface repid="IDL:CF/PropertySet:1.0"/>
      <!--[Inherited interface IDL:CF/TestableObject:1.0]-->
      <inheritsinterface repid="IDL:CF/TestableObject:1.0"/>
    </interface>
    <interface name="complexShort" repid="IDL:standardInterfaces/complexShort:1.0"/>
  </interfaces>
</softwarecomponent>
```

## WFMDemod.prf.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE properties SYSTEM "../dtd/properties.dtd">
<!--Created with OSSIE WaveDev Beta Version 0.6.2-->
<!--Powered by Python-->
<properties>
  <simple id="DCE:35c5eb82-c548-11dc-9f05-000c29a1ba37" mode="readonly" name="Dummy"
    type="short">
    <value>1</value>
    <description>Dummy</description>
    <kind kindtype="configure"/>
  </simple>
</properties>
```

## WFMDemod.cpp

```
/******
```

Copyright 2008 Virginia Polytechnic Institute and State University

This file is part of the OSSIE WFMdemod.

OSSIE WFMdemod is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

OSSIE WFMdemod is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with OSSIE WFMdemod; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

```
*****/
```

```
#include <string>
#include <iostream>
```

```
#include <math.h>
#include <assert.h>
#include <byteswap.h>
```

```
#include "WFMdemod.h"
```

```
WFMdemod_i::WFMdemod_i(const char *uuid, omni_condition *condition) :
Resource_impl(uuid), I1(0), I2(0), Q1(0), Q2(0), component_running(condition)
{
    dataIn = new standardInterfaces_i::complexShort_p("dataIn");
    dataOut = new standardInterfaces_i::complexShort_u("dataOut");

    //Create the thread for the writer's processing function
    processing_thread = new omni_thread(run, (void *) this);

    //Start the thread containing the writer's processing function
    processing_thread->start();
}
```

```
WFMdemod_i::~~WFMdemod_i(void)
{
```

```

    delete dataIn;
    delete dataOut;
}

// Static function for omni thread
void WFMDemod_i::run( void * data )
{
    ((WFMDemod_i*)data)->ProcessData();
}

CORBA::Object_ptr WFMDemod_i::getPort( const char* portName ) throw
( CORBA::SystemException, CF::PortSupplier::UnknownPort)
{
    DEBUG(3, WFMDemod, "getPort() invoked with " << portName)

    CORBA::Object_var p;

    p = dataIn->getPort(portName);

    if (!CORBA::is_nil(p))
        return p._retn();

    p = dataOut->getPort(portName);

    if (!CORBA::is_nil(p))
        return p._retn();

    /*exception*/
    throw CF::PortSupplier::UnknownPort();
}

void WFMDemod_i::start() throw (CORBA::SystemException, CF::Resource::StartError)
{
    DEBUG(3, WFMDemod, "start() invoked")
}

void WFMDemod_i::stop() throw (CORBA::SystemException, CF::Resource::StopError)
{
    DEBUG(3, WFMDemod, "stop() invoked")
}

void WFMDemod_i::releaseObject() throw (CORBA::SystemException,
CF::LifeCycle::ReleaseError)
{
    DEBUG(3, WFMDemod, "releaseObject() invoked")

    component_running->signal();
}

void WFMDemod_i::initialize() throw (CF::LifeCycle::InitializeError,
CORBA::SystemException)
{
    DEBUG(3, WFMDemod, "initialize() invoked")
}

void WFMDemod_i::configure(const CF::Properties& props) throw (CORBA::SystemException,
CF::PropertySet::InvalidConfiguration, CF::PropertySet::PartialConfiguration)
{
    DEBUG(3, WFMDemod, "configure() invoked")

    std::cout << "props length : " << props.length() << std::endl;

    for (unsigned int i = 0; i < props.length(); i++)

```

```

    {
        std::cout << "Property id : " << props[i].id << std::endl;

        if (strcmp(props[i].id, "DCE:35c5eb82-c548-11dc-9f05-000c29a1ba37") == 0)
        {
            CORBA::Short simple_temp;
            props[i].value >>= simple_temp;
            simple_0_value = simple_temp;
        }
    }
}

void WFMDemod_i::query (CF::Properties & configProperties) throw (CORBA::SystemException,
CF::UnknownProperties)
{
}

void WFMDemod_i::runTest (CORBA::ULong _number, CF::Properties & _props) throw
(CORBA::SystemException, CF::TestableObject::UnknownTest, CF::UnknownProperties)
{
}

void WFMDemod_i::ProcessData()
{
    DEBUG(3, WFMDemod, "ProcessData() invoked");

    Demod_Lyons();
}

void WFMDemod_i::Demod_Lyons()
{
    PortTypes::ShortSequence I_out, Q_out;

    PortTypes::ShortSequence *I_in(NULL), *Q_in(NULL);

    while(1)
    {
        dataIn->getData(I_in, Q_in);

        unsigned int len = (*I_in).length();
        I_out.length(len);
        Q_out.length(len);

        for (unsigned int i(0); i < len; ++i) {
            int IO((*I_in)[i]), QO((*Q_in)[i]);

            DEBUG(10, WFMDemod, "Normalized: I_in = " << IO << ", Q_in = " << QO << ", mag
            = " << sqrt(IO*IO + QO*QO));

#ifdef 1
            // Calculate output from Lyon's Fig 13-61(b)
            I_out[i] = ((I1 * (QO - Q2)) >> 10) - ((Q1 * (IO - I2)) >> 10);
#else
            // Brute force FM demod
            I_out[i] = atan2((IO*I1 + QO*Q1), (I1*QO - IO*Q1)) * 5000;
#endif

            Q_out[i] = I_out[i];

            // Update delay terms;

```

```

        I2 = I1;
        I1 = I0;
        Q2 = Q1;
        Q1 = Q0;
    }

    dataIn->bufferEmptied();
    dataOut->pushPacket(I_out, Q_out);
}
}

```

## WFMDemod.h

/\*\*\*\*\*

Copyright 2008 Virginia Polytechnic Institute and State University

This file is part of the OSSIE WFMdemod.

OSSIE WFMdemod is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

OSSIE WFMdemod is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with OSSIE WFMdemod; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

\*\*\*\*\*/

```

#ifndef WFMDEMODO_IMPL_H
#define WFMDEMODO_IMPL_H

```

```

#include <stdlib.h>

```

```

#include "ossie/cf.h"
#include "ossie/PortTypes.h"
#include "ossie/Resource_impl.h"
#include "ossie/debug.h"

```

```

#include "standardinterfaces/complexShort_u.h"
#include "standardinterfaces/complexShort_p.h"

```

```

/** \brief
 *
 *
 */

```

```

class WFMDemod_i : public virtual Resource_impl
{
public:
    /// Initializing constructor
    WFMDemod_i(const char *uuid, omni_condition *sem);

```

```

/// Destructor
~WFMDemod_i(void);

/// Static function for omni thread
static void run( void * data );

///
void start() throw (CF::Resource::StartError, CORBA::SystemException);

///
void stop() throw (CF::Resource::StopError, CORBA::SystemException);

///
CORBA::Object_ptr getPort( const char* portName )
throw (CF::PortSupplier::UnknownPort, CORBA::SystemException);

///
void releaseObject() throw (CF::LifeCycle::ReleaseError,
CORBA::SystemException);

///
void initialize() throw (CF::LifeCycle::InitializeError,
CORBA::SystemException);

/// Configures properties read from .prf.xml
void configure(const CF::Properties&) throw (CORBA::SystemException,
CF::PropertySet::InvalidConfiguration, CF::PropertySet::PartialConfiguration);

void query (CF::Properties & configProperties) throw (CORBA::SystemException,
CF::UnknownProperties);

void runTest (CORBA::ULong _number, CF::Properties & _props) throw
(CORBA::SystemException, CF::TestableObject::UnknownTest,
CF::UnknownProperties);

```

```
private:
```

```

/// Disallow default constructor
WFMDemod_i();

/// Disallow copy constructor
WFMDemod_i(WFMDemod_i&);

/// Main signal processing method
void ProcessData();

void Demod_Lyons();
int I1, I2, Q1, Q2;

omni_condition *component_running; ///< for component shutdown
omni_thread *processing_thread;    ///< for component writer function

CORBA::Short simple_0_value;

// list components provides and uses ports
standardInterfaces_i::complexShort_p *dataIn;
standardInterfaces_i::complexShort_u *dataOut;

```

```
};
#endif
```

## Appendix B:

### WFMDemod\_update.cpp

```
/******
```

Copyright 2007 Virginia Polytechnic Institute and State University

This file is part of the OSSIE WFMDemod\_update.

OSSIE WFMDemod\_update is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

OSSIE WFMDemod\_update is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with OSSIE WFMDemod\_update; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

```
*****/
```

```
#include <string>
#include <iostream>
#include "WFMDemod_update.h"
#include <math.h>
```

```
WFMDemod_update_i::WFMDemod_update_i(const char *uuid, omni_condition *condition) :
    Resource_impl(uuid), component_running(condition)
{
    dataIn_0 = new standardInterfaces_i::complexShort_p("dataIn");
    dataOut_0 = new standardInterfaces_i::complexShort_u("dataOut");

    //Create the thread for the writer's processing function
    processing_thread = new omni_thread(Run, (void *) this);

    //Start the thread containing the writer's processing function
    processing_thread->start();
}

```

```
WFMDemod_update_i::~WFMDemod_update_i(void)
{
    delete dataIn_0;
    delete dataOut_0;
}

```

```
// Static function for omni thread
void WFMDemod_update_i::Run( void * data )
{
    ((WFMDemod_update_i*)data)->ProcessData();
}

```

```
CORBA::Object_ptr WFMDemod_update_i::getPort( const char* portName ) throw (
    CORBA::SystemException, CF::PortSupplier::UnknownPort)
{
    DEBUG(3, WFMDemod_update, "getPort() invoked with " << portName)
}

```

```

CORBA::Object_var p;

p = dataIn_0->getPort(portName);

if (!CORBA::is_nil(p))
    return p._retn();

p = dataOut_0->getPort(portName);

if (!CORBA::is_nil(p))
    return p._retn();

/*exception*/
throw CF::PortSupplier::UnknownPort();
}

void WFMDemod_update_i::start() throw (CORBA::SystemException,
CF::Resource::StartError)
{
    DEBUG(3, WFMDemod_update, "start() invoked")
}

void WFMDemod_update_i::stop() throw (CORBA::SystemException, CF::Resource::StopError)
{
    DEBUG(3, WFMDemod_update, "stop() invoked")
}

void WFMDemod_update_i::releaseObject() throw (CORBA::SystemException,
CF::LifeCycle::ReleaseError)
{
    DEBUG(3, WFMDemod_update, "releaseObject() invoked")

    component_running->signal();
}

void WFMDemod_update_i::initialize() throw (CF::LifeCycle::InitializeError,
CORBA::SystemException)
{
    DEBUG(3, WFMDemod_update, "initialize() invoked")
}

void WFMDemod_update_i::configure(const CF::Properties& props)
throw (CORBA::SystemException,
CF::PropertySet::InvalidConfiguration,
CF::PropertySet::PartialConfiguration)
{
    DEBUG(3, WFMDemod_update, "configure() invoked")

    std::cout << "props length : " << props.length() << std::endl;

    for (unsigned int i = 0; i <props.length(); i++)
    {
        std::cout << "Property id : " << props[i].id << std::endl;
    }
}

void WFMDemod_update_i::ProcessData()
{
    DEBUG(3, WFMDemod_update, "ProcessData() invoked")

    PortTypes::ShortSequence I_out_0, Q_out_0;
}

```

```

PortTypes::ShortSequence *I_in_0(NULL), *Q_in_0(NULL);
CORBA::UShort I_in_0_length, Q_in_0_length;

int I1, I2, Q1, Q2;

while(1)
{
    dataIn_0->getData(I_in_0, Q_in_0);

    I_in_0_length = I_in_0->length();
    Q_in_0_length = Q_in_0->length();

    I_out_0.length(I_in_0_length); //must define length of output
    Q_out_0.length(Q_in_0_length); //must define length of output

    /*insert code here to do work*/

    for (unsigned int i(0); i < I_in_0_length; ++i) {
        int IO((*I_in_0)[i]), QO((*Q_in_0)[i]);

        DEBUG(10, WFMDemod, "Normalized: I_in = " << IO << ", Q_in = " << QO << ", mag = "
        << sqrt(IO*IO + QO*QO));

#ifdef 1
        // Calculate output from Lyon's Fig 13-61(b)
        I_out_0[i] = ((I1 * (QO - Q2)) >> 10) - ((Q1 * (IO - I2)) >> 10);
#else
        // Brute force FM demod
        I_out_0[i] = atan2((IO*I1 + QO*Q1), (I1*QO - IO*Q1)) * 5000;
#endif

        Q_out_0[i] = I_out_0[i];

        // Update delay terms;
        I2 = I1;
        I1 = IO;
        Q2 = Q1;
        Q1 = QO;

    }

    dataIn_0->bufferEmptied();
    dataOut_0->pushPacket(I_out_0, Q_out_0);
}
}

```