Test Manual for F3RP61

Device Support

|  |  |
| --- | --- |
| Revision: | 1.1 |
| Status: | Released |
| Last modification: | January 7, 2014 |
| Created: | December 18, 2013 |

|  |  |  |
| --- | --- | --- |
| **Prepared by** | **Reviewed by** | **Approved by** |
| Gregor Kostevc | Takashi Nakamoto | Takashi Nakamoto |
|  |  |  |

Document History



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Revision | Date | Changed/reviewed | Section(s) | Modification |
| 0.0 | 2013-12-20 | gkostevc | All | Creation |
| 0.1 | 2013-12-27 | gkostevc | All |  |
| 1.0 | 2014-01-06 | tnakamoto | All | Reviewed. Reflected review results. |
| 1.1 | 2014-01-07 | gkostevc  tnakamoto | 5 | Added examples of how to add new test cases. |

Scope

This document describes test procedures for F3RP61 EPICS Device Support. It provides instructions on how to properly set up hardware and software testing environment and run Automated and Manual Tests. Test reports are generated automatically.The audience of this document are the test engineers who will execute the Test Plan.

This document covers the following aspects:

* Description of testing framework.
* Test environment setup.
* Automated test execution.
* Manual test execution.
* Instructions on how to extend the test for new Test Cases.

Table of Contents

[1. Introduction 5](#_Toc376865968)

[1.1. Requirements 5](#_Toc376865969)

[2. Hardware set-up 7](#_Toc376865970)

[3. Test environment set-up 8](#_Toc376865971)

[3.1. f3rp61-linux 8](#_Toc376865972)

[3.2. Sequence CPU 8](#_Toc376865973)

[3.3. Host machine 8](#_Toc376865974)

[3.3.1. Using existing CentOS machine 8](#_Toc376865975)

[3.3.2. New installation of CentOS 9](#_Toc376865976)

[3.3.3. Troubleshooting 9](#_Toc376865977)

[3.4. Test configuration 10](#_Toc376865978)

[4. Execution of tests 11](#_Toc376865979)

[4.1. Automated Test 11](#_Toc376865980)

[4.1.1. Test Report generation 11](#_Toc376865981)

[4.1.2. Test procedure 11](#_Toc376865982)

[4.2. Manual Test 11](#_Toc376865983)

[4.2.1. Test Report generation 12](#_Toc376865984)

[4.2.2. Test procedure 12](#_Toc376865985)

[4.3. Troubleshooting 12](#_Toc376865986)

[4.3.1. Specific PVs 12](#_Toc376865987)

[4.3.2. Run specific command 13](#_Toc376865988)

[5. Extending and modifying tests 14](#_Toc376865989)

[5.1. Changing Report style 14](#_Toc376865990)

[5.2. Adding New PVs 14](#_Toc376865991)

[5.3. Adding Test Cases 14](#_Toc376865992)

[5.3.1. Automated Test 14](#_Toc376865993)

[5.3.2. Manual Test 15](#_Toc376865994)

[5.3.3. Manual Test iocsh Commands 16](#_Toc376865995)

[5.4. Replacing Channel Access library 17](#_Toc376865996)

Figures

[Figure 1: Loopbacks between output and input modules. 7](#_Toc376781760)

Tables

[Table 2‑1: Loopbacks 7](#_Toc376781761)

Glossary of Terms

|  |  |
| --- | --- |
| CLI | Command Line Interface |
| Device | Shared relay/register, Internal relay/register, I/O module registers |
| GUI | Graphical User Interface |
| HMI | Human Machine Interface |
| IOC | Input / Output Controller |
| Test case | Set of test inputs, execution conditions, and expected results developed for a particular objective. |
| Test item | A software or system item that is an object of testing.  Synonym: system under test. |
| Test procedure | Detailed instructions for the setup, execution, and evaluation of results for a given test case. |
| iocsh | EPICS IOC Shell |
| BCD | Binary-coded-decimal |

References

1. CaChannel  
   <https://pypi.python.org/pypi/CaChannel>
2. PyUnit  
   <http://pyunit.sourceforge.net/>
3. Yokogawa Electric Cooperation,  
   e-RT3: "User’s Manual: RTOS-CPU module (F3RP61-□□) Linux BSP Reference Manual",  
   IM 34M06M51-44 2nd Edition
4. Yokogawa Electric Cooperation,  
   e-RT3: "User’s Manual: RTOS-CPU module (F3RP61-□□) Linux BSP Start-up Manual"  
   IM 34M06M51-43 2nd Edition
5. J. Odagiri,  
   Device and Driver Support for F3RP61 (Ver. 1.2.0)  
   <http://www-linac.kek.jp/cont/epics/f3rp61/DevSup_F3RP61-1.2.0.pdf>

# Introduction

All test scripts are written in Python. Python library used for ChannelAccess communication is CaChannel[1]. Test framework used for testing is PyUnit[2]. Test framework can be obtained from F3RP61 Device Support git repository on GitHub (<https://github.com/EPICS-F3RP61/epics-f3rp61>).

Also EPICS database files are provided. They are needed to run IOC and WideField project with a ladder program that is to be run on sequence CPU during the test.

A directory tree of F3RP61 Device Support test is provided below with description of each file or directory:

test/

├── automatedTest.py # defines the runner used to run automated test

├── RunAutomatedTest.py # used to execute automated test

├── runIOC.py # app build, ssh and IOC start functions

├── manualTest.py # Test Cases for manual test

├── manualTestCom.py # Test Cases for manual test of iocsh commands

├── RunManualTest.py # used to execute manual test

├── testIO.py # Test Cases for IO driver (F3RP61)

├── testSeq.py # Test Cases for seq driver (F3RP61Seq)

├── testSysCtl.py # Test Cases for SysCtl driver (F3RP61SysCtl)

├── wrapper.py # wrapper functions for caget caput

├── iocTest # IOC source files

│   ├── configure

│   ├── iocBoot

│   ├── Makefile

│   └── TestApp

├── modules # Additional software

│   ├── HTMLTestRunner.py # Used to generate HTML test report

├── LICENSE # License file

├── OPI # Provided OPI screens

│   ├── TEST-f3rp61.opi # OPI screen with all used PVs

├── README

├── reports

├── seqTest # Sequence CPU program/project for WideField

│   ├──

All Test Cases are provided with description in the form of 'print' statements at the

beginning of every Test Case. First line of the description is always a list of **keywords**

that are relevant to the Test Case, with detailed description in the following lines.

Description of each Test Case is provided also in the test report.

## Requirements

Requirements to run the automated and manual test for F3RP61 Device Support are the following:

* Linux Host machine: CentOS 6.4 32-bit with the following installed:
  + f3rp61 Device Support, version that is to be tested (see [4] on how to install device support)
  + Python (preferably 2.7) with pyUnit, CaChannel and pexpect modules
  + **Optional**: Control System Studio (CSS) for manual PV manipulation through provided OPI screen
* Windows machine with WideField software installed
* PLC with the following modules: SP71-4S, RP61-2L, YC08-0N, XD08-6F
* f3rp61-linux running on RP61-2L (see [4] on how to prepare f3rp61-linux)

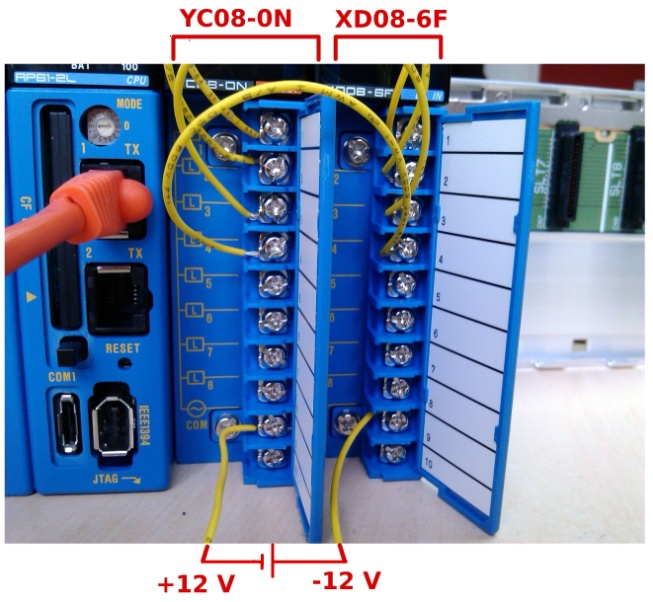
# Hardware set-up

This section provides instructions on how to set-up hardware before running any tests.

PLC modules must be installed in the following order:

* Power Supply: F3PU30-0S
* **slot1** Sequence CPU: F3SP71-4S
* **slot2** Linux CPU: F3RP61-2R
* **slot3** Output Module: F3YC08-0N
* **slot4** Input Module: F3XD08-6F

Loopbacks must be made between input and output modules (F3YC08-0N and F3XD08-6F). This is mandatory for inputs/outputs from 1 to 4 as shown in the picture. The rest of inputs/outputs can stay unconnected.

Table 2‑1: Loopbacks

|  |
| --- |
| Out1🡪In1 |
| Out2 🡪In2 |
| Out3 🡪In3 |
| Out4 🡪In4 |

Figure 1: Loopbacks between output and input modules.

# Test environment set-up

Before any tests can be run it is necessary to prepare the testing environment. This section describes how to prepare testing environment according to requirements described in section 1.1. Those are:

* f3rp61-linux must be running and allow ssh connection – this way IOC application with the required database files can be copied to it and run.
* On Sequence CPU the provided ladder program must be running.
* Host CentOS machine must have required software installed to be able to execute tests.

Additionally, Test must be configured according to the environment.

To properly set up test environment and configure the test follow the steps described in the following sections.

## f3rp61-linux

If f3rp61-linux is already running on F3RP61 module, with the correct version of the Device Support and the IP of f3rp61-linux is known and it accepts ssh connections, this section can be skipped. Proceed with 3.2.

Otherwise, follow instructions in [4] on how to prepare f3rp61-linux on CF card with correct Device Support version and run it on F3RP61 module.

## Sequence CPU

Open project from directory *seqTest/* in WideField, download it to Sequence CPU and set it to Run mode.

## Host machine

### Using existing CentOS machine

If followed section 3.1 the CentOS Host Machine with EPICS Base and correct version of F3RP61 Device Support is already installed. In this case, only the following Python modules need to be installed:

* CaChannel: <https://pypi.python.org/pypi/CaChannel>
* pyUnit: <http://pyunit.sourceforge.net/>
* pexpect: <http://pexpect.readthedocs.org/en/latest/>

All of them can be installed using easy\_install (<https://pypi.python.org/pypi/setuptools>):

$ easy\_install cachannel

$ easy\_install pyunit

$ easy\_install pexpect

### New installation of CentOS

This section can be skipped if section 3.3.1 was followed successfully.

To prepare new CentOS installation download CentOS 6.4 32-bit and install it. During installation choose ‘Installation for Developing’. This will install most of development tools needed. After installation development tools can be installed using the command

# yum groupinstall "Development tools"

# yum groupinstall "Additional Development"

After operating system installation has finished EPICS Base and the correct version of F3RP61 Device Support need to be installed. To build EPICS Base (<http://www.aps.anl.gov/epics/download/base/>) a package readline-devel is required and environment variable EPICS\_HOST\_ARCH needs to be set to linux-x86. To install F3RP61 Device Support follow instructions in [5].

Set environment variable for Channel Access:

$ export EPICS\_CA\_ADDR\_LIST=*IP*

Where *IP* is the IP address of f3rp61-linux set in the section 3.1.

To finish preparing test development environment some Python modules need to be installed. To do this follow the steps in 3.3.1.

### Troubleshooting

Default Python installation on CentOS 6.4 is version 2.6. Required modules should be available for this version and tests should run without errors. However, it was only tested on Python version 2.7 so in case of any problems with version 2.6 please install Python 2.7 and repeat installation of required modules under new 2.7 installation.

In case of python2.7 installation all commands described in the following sections containing

$ python

must be changed to

$ python2.7

## Test configuration

To configure the test according to the environment that was just set up, set the following parameters:

* f3rp61-linux IP address (set when following instructions in section 3.1)
* f3rp61-linux root password (set when following instructions in section 3.1)

That are found at the top of the file test/runIOC.py .

Example:

IP="10.5.2.190"

PASSWORD="root12"

# Execution of tests

Execution of tests requires test environment set up as described in section 3 and hardware set up as described in section 2.

## Automated Test

Automated test is created in a way that, after proper test environment was set up, it only needs to be run by the tester. All testing is automated as well as test report generation.

To run the automated test open terminal, change to test’s top directory test/ and run the command:

$ python RunAutomatedTest.py

Wait until the test finishes – the following line is written:

$ \*\*\*\*\*\*\*\*\* FINISHED \*\*\*\*\*\*\*\*\*

### Test Report generation

Test report is automatically generated in the file test/generated-reports/automatedTest\_Report.html. Output during the testing that is not caught by testing framework is also logged to a file test/generated-reports/automatedTest\_Output.txt.

### Test procedure

Detailed test procedure is provided just for your information and in case of problems for easier debugging.

├── RunAutomatedTest.py

├── build IOC application in test/iocTest

├── scp IOC app to f3rp61-linux

├── ssh and run IOC on f3rp61-linux

├── Run test cases

├── Generate report

## Manual Test

Manual test is designed in a similar way as Automated test but it is provided with interactive interface and requires input from the tester.

To run Manual test open the Terminal, change directory to test’s top directory test/ and run the command:

$ python RunManualTest.py

New terminal window will open where some commands will be run to start IOC on f3rp61-linux.

After that, some input will be required on the second terminal. When that has finished, some input on the native terminal will be required. Messages on the screen will guide you through the test. When tests in both terminals have finished, second terminal needs to be closed by typing:

$ exit

### Test Report generation

Test report is automatically generated in the directory test/generated-reports as files *manualTest\_Report.txt* and *manualTestCom\_Report.txt*.

### Test procedure

Detailed test procedure is provided just for your information and in case of problems for easier debugging.

├── RunManualTest.py

├── Terminal\_2

├── build IOC application in Test/iocTest

├── scp IOC app to f3rp61-linux

├── ssh and run IOC on f3rp61-linux

├── Run test cases for iocsh commands and acquire some input

├── Run the rest of test cases and acquire some input

├── Generate report

## Troubleshooting

In case of errors (when either Automated or Manual tests cannot be run properly), some commands can be run manually to allow faster discovery of the step that produces the error. Additionally, if tests fail for a specific PV, the behavior of each PV can be tested manually using the provided OPI screen.

### Specific PVs

OPI screen can be found in the directory *test/OPI/* and for opening it requires CSS on the Host Machine. Using that screen each tested PV can be set, read and examined.

In case OPI screen is not shown correctly, Color file IOcolors.def is also provided - this can be imported to CSS through: Edit *->* Preferences *->* Display *->* BOY *->* Color File *.*

### Run specific command

How to run specific steps manually is described with the commands below:

1. Build IOC application:

$ cd iocTest

$ make

$ cd ..

**Expected result:** make command completes without errors.

1. Copy IOC application to f3rp61-linux:

$ cd ..

$ scp –r iocTest root@IP:/opt/epics/

where IP address and password for root user were set during environment set-up (see section 3.1 for details).

**Expected result:** A list of successfully copied files.

1. SSH Connect.

$ ssh root@*IP*

where IP address and password for root user were set during environment set-up (see section 3.1 for instructions).

**Expected result:** f3rp61-linux root shell prompt.

1. Run IOC.

# cd /opt/epics/iocTest/iocBoot/iocTest/

# ../../bin/linux-f3rp61/Test st.cmd

**Expected result:**f3rp61 EPICS Shell prompt (iocsh) with the following database files loaded: *test.db, test\_li.db, test\_lo.db, test\_mbbi.db, test\_mbbo.db, testSeq.db, testSysCtl.db* and no errors or warnings listed.

# Extending and modifying tests

## Changing Report style

Changing report style from HTML to Text can be done by modifying the file *automatedTest.py.* Comment lines that correspond to HTMLTestRunner and uncomment lines for TextTestRunner.

## Adding New PVs

When new PV needs to be added for the new Test Case, it is recommended to add it in the same fashion as it is done for the PV already present - the examples can be seen at the beginning of *unittest.TestCase* classes defined in the files *testIO.py, testSeq.py*, *testSysCtl.py* and *manualTest.py*. PV’s name needs to be added to the *\*PVs* list (below PV initializations) otherwise it will not be called during *setUp* and *tearDown* methods.

## Adding Test Cases

New Test Cases can be added to either of the following five files:

* *testIO.py* (for DTYP F3RP61),
* *testSeq.py* (for DTYP F3RP61Seq),
* *testSysCtl.py* (for DTYP F3RP61SysCtl),
* *manualTest.py* (for manual test Test Cases),
* *manualTestCom.py* (for manual test Test Cases testing *iocsh commands*).

They must be appended to the end of the file, following the established naming convention and accompanied with descriptive comments.

### Automated Test

New Test Case can be added to one of the three files *testIO.py, testSeq.py testSysCtl.py*, depending on DTYP of the record being tested. In order for the new Test Case to be run under Automated Test, it needs to be added to the Test Suite in the *automatedTest.py* file.

Example below shows how to add a test case for a new PV named *TEST:mbbo\_example1* with DTYP set to *F3RP61*. For more options also see pyUnit manual [2].

Modifying *testIO.py* file to add PV:

...

...

class IOTest(unittest.TestCase):

pv\_bo\_Y = pv('TEST:bo\_X')

pv\_mbbo\_example1 = pv('TEST:mbbo\_example1')

IOPVs=[..., ...

pv\_bo\_X, pv\_mbbo\_example1 \

]

Modifying *testIO.py* file to add Test Case:

...

...

class IOTest(unittest.TestCase):

...

...

def test\_TC\_17(self):

...

...

def test\_TC\_18(self):

print ('TC-17: ...keywords...')

print ('TC-17 ...description...')

self.pv\_mbbo\_example1.write(100, self.SCAN\_TIME)

self.assertEqual(self.pv\_mbbo\_example1.read(), 100, 'Fail message.')

Modifying *automatedTest.py* file:

...

...

def IOsuite():

suite = unittest.TestSuite()

...

...

suite.addTest(IOTest("test\_TC\_15"))

suite.addTest(IOTest("test\_TC\_16"))

suite.addTest(IOTest("test\_TC\_17"))

return suite

### Manual Test

For Manual Test the only file to modify is manualTest.py. New Test Case needs to be added to *Test\_Cases* list in the *main* function at the bottom of *manualTest.py* file.

Method *log\_( )* is used to print string to stdout and at the same time write it to report file.

Example below shows how to add a test case for a new PV named *TEST:mbbo\_example2* :

Modifying *manualTest.py* file to add PV:

...

...

class manualTest():

pv\_biSysCtl\_REG = pv('TEST:biSysCtl\_REG')

pv\_mbbo\_example2 = pv('TEST:mbbo\_example2')

SysCtlPVs=[..., ...

pv\_biSysCtl\_REG, pv\_mbbo\_example2 \

]

Modifying *manualTest.py* file to add Test Case:

...

...

class manualTest():

...

...

def test\_TC\_05(self, log\_):

...

...

def test\_TC\_06(self, log\_):

log\_ ('TC-06: ...keywords...')

log\_ ('TC-06 ...description...')

ans = self.pv\_mbbo\_example2.read()

ans = raw\_input("Enter expected value of the example2 record: ")

log\_.assert\_ (ans, pos)

Modifying *manualTest.py* file to add Test Case to the list:

...

...

if \_\_name\_\_ == '\_\_main\_\_':

...

...

# Add new test cases to this list in order for them to be run

Test\_Cases = [manualTest.test\_TC\_01,... ,..., manualTest.test\_TC\_06]

...

...

### Manual Test iocsh Commands

For Manual Test of iocsh commands the only file to modify is manualTestCom.py.

Method *log\_(str)* is used to print string to standard output and at the same time write it to report file.

Method *log\_.assert\_(a,b)* is used to check if *a* is equal to *b* and write either *PASS* or *FAIL* to both standard output and report file.

The following method is used to call an iocsh command:

sshh\_.sendline(‘command’)

Example below shows how to add a test case for a new iocsh command called *example3* that takes one integer parameter:

def commandTestCases(sshh\_, log\_):

...

...

# TC\_Com5

sshh\_.sendline('example3 1')

log\_ ('TC\_Com5 Setting example3 to 1.')

ans = raw\_input("Is example3 set to 1? (y/n): ")

sshh\_.sendline('example3 0')

log\_.assert\_(ans, "y")

## Replacing Channel Access library

A wrapper is used to wrap all Channel Access specific functions into custom methods so current Channel Access library *CaChannel* can be easily replaced (for example, by *pyEpics*) without the need to modify code of Test Cases in *testIO.py, testSeq.py*, *testSysCtl.py, manualTest.py and manualTestCom.py*. Only *wrapper.py* file needs to be properly modified.