ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WECPR04

AUTOMATED TESTING AND VALIDATION OF CONTROL PARAMETERS*

P. Kankiya†, J. P. Jamilkowski, A. Sukhanov
Brookhaven National Laboratory, Upton, USA

Abstract

The BNL CA-D controls environment has recently been adopting modern programming languages such as Python. A new framework has been created to instantiate setting and measurement parameters in Python as an alternative to C++ and Java process-variable-like objects. With the help of automated testing tools such as pyTest and Coverage, a test suite is generated and executed before the release of Python-based accelerator device objects (ADO) to assure quality as well as compatibility. This suite allows developers to add custom tests, repeat failed tests, create random inputs, and log failures.

INTRODUCTION

A particle accelerator machine is a collection of large number of hardware equipment that form the basic layer of control system software. The control access to these instruments at collider accelerator department of Brookhaven National Laboratory is provided by a logical device object known as ADO (Accelerator Device Object) [1]. ADO is a C++ container type class which provides a software view of a collection of collider control points known as parameters. These parameters are the basis of supervisory control , data acquisition and monitoring. Parameters are software entities derived from a base class with several properties that constitute the metadata of each instantiated control and measurement object. These objects when added to the container class represent the controls framework. By assuring that creation of these parameters is of a certain standard of quality- will help to prevent crashes in operational time and hence down time of beam.

Large part of testing of this framework is exercised manually at the time of creation of ADOs. The steps to test the properties associated with each parameter are repeatable and therefore should be automated. Automation of testing software provides benefits such as repeatability, reliability, and report generation. Several open source tools are rich in features that can make the unit testing of ADOs seamless. One such tool in consideration is pyTest [2].

ROADMAP OF UNIT TESTING OF ADOS

The testing phase is broken down into five steps that should be carried out to before release of ADO software as a best practice. These steps are depicted in Fig. 1.

Phase 1 requires preparation of test data in general unit testing terms this phase is referred to as set up phase. Once the test data is available and expected format it should run through all the respective test suites, which is phase 2. This phase concludes the quality assurance phase testing. This step will assure ADOs are compatible with most of the control's framework in place.

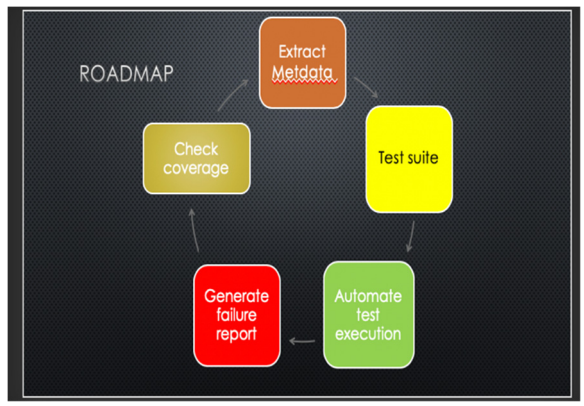

Figure 1. A general guideline to maximise coverage of tests to follow for creating unit tests for ADO parameters.

parameter name		parameter:desc
testC	0	This is a configuration parameter
menuS	second	menu setting
charS		char setting
ucharS	0	uchar setting
shortS	1	short int setting
ushortS	0	unsigned short int setting
intS	321	int setting
uintS	0	unsigned int setting
longS	133	long int setting
ulongS	7	unsigned long int setting
floatS	3.5e+02	single float setting
doubleS	344.643554688	double float setting
slowS	0	like longS, except set takes 12 s
shortWatchM	1	this parameter is used for monito
shortWatchM:something	0	this parameter is used for monito
ushortWatchM	0	this parameter is used for monito
stringInS	IP6v	string setting
stringMonitorM	IP6v	this parameter is used for monito
stringS	user 8	string setting
charArrayS	[1 2 3 4 5 -100 -101 -102	char array setting
ucharArrayS	[0 0 0 0 0 0 0 0 0 0]	unsigned char array setting
shortArrayS	[0 0 0 0 0 0 0 0 0 0]	short array setting
ushortArrayS	[3 4 5 6 7 8 20 5 7 26]	unsigned short array setting
longArrayS	[0 0 0 0 0 0 0 0 0 0]	long array setting
ulongArrayS	[1 0 0 0 0 0 0 0 0 0]	unsigned long array setting
floatArrayS	[0.1 0.2 0.3 0.4 0.5...]	float array setting
doubleArrayS	[-0.1 -0.2 -0.3 -0.4 -0.5	double array setting

Figure 2: A generic ADO called simple containing a large number of parameters.

System Under Test

The snapshot in Fig. 2 shows the variety of parameters under test ADO. This ado called simple ADO is used to perform testing upon. Parameters here vary from being configuration data , numeric normative types, strings to arrays of such basic data types.

*Work supported by Brookhaven Science Associates, LLC under Contract No. DE-SC0012704 with the U.S. Department of Energy.
†pkankiya@bnl.gov

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WECPR04


```python
params_num = []

@pytest.fixture()
def ado():
    from cad import pyado
    name  = 'simple.test'
    iface = pyado.useDirect()
    rtn   = iface.getMeta(name, all = True)

@pytest.fixture()
def params_list(ado):
    param_props = ado.iface.getMeta(ado, all = True)
    for param in param_props.items():
        if(param[1][2] in ['UIntType','DoubleType','IntType','UShortType','LongType','FloatType']):
            params_num.append(param)

def test_englow(ado):
    param_englow = []
    for param in params_num:
        param_englow = iface.get(ado,str(f[1][0]+'.engHigh'))
        if(param.items()[0][1].items()[1][0]!= 'error'):
            param_englow.append(param.items())
    assert len(param_englow) != len(params_num)
```

Figure 3: Code snippet describing displaying a fixture written for testing a parameter property associated with numeric data type control parameters.

GENERATING TESTS

With help of the concept of fixtures (see Fig. 3) which allow for variables to persist through out a test session, pyTest [2] provides for setup and teardown of test environment. The test cases are categorized into following:

1. Test of metadata of parameters
2. Test of set and get methods of each parameters
3. Tests for benchmarking expected outputs
4. Test customized for functionality

Tests for Metadata

At the very basic each parameter is expected to be possessing of a valid value, description and readability mode. Based on the normative type the valid values for these can be predicted. The pyTest suite would fail if these properties are not set up with one of the expected settings. The test suite for metadata testing is contained in a sperate file. The file can be executed by pyTest with ado name provided as a command line input. At the start of the test execution, a fixture generates and maintains a list of parameters and its properties to be tested. These collections are stored in a python list. An individual unit test is written for each property. For example , when testing for engineering limits of numeric data types - simple checking if the return value is numeric will report the test as passed. When a parameter does not have this value set, the test is failed.

Test for Set and Get Methods

All control parameters when updated with a value invoke set or get methods based on their category being a setting or a measurement. In case of ADOs these methods are referred to as setcode and getcode. The return value of these member methods is defined by set of error codes. A test suite is added to the file for testing setcodes which provides for randomly generated test inputs based on properties set on the parameter which can be extracted from the metadata. Again, a fixture is used to extract parameters that are settable and to maintain this list throughout the test session. pyTest also allows for catching warning and exceptions instead of errors are reported appropriately. A similar strategy is adopted to create unit test for measurement parameters and their returned values are compared with expected to validate the member function.

Test for Benchmarking

ADO as a class is instantiated at multiple machine interfaces. To generalise the tests of each instance without manual execution of test suite for individual ADO a feature of pyTest called parameterisation is utilised. Parameterisation is the process of providing collection of input values for a given test stub and the expected output value. This can significantly reduce developer efforts to deploy ADOs at a large scale and catch errors that human eyes can miss. For example, when deploying an ADO for controlling magnet power supplies which can exist at the order of hundreds of ADOs a list of operation limits of the hardware must be enforced. To provide a parameterised list of input a simple decorator "@pytest.mark.parameterise" is added before the test function and a list of inputs is attached.

Custom Tests

A separate test file containing fixtures of generating ado-metadata is created where test stubs for testing business logic of the ados can be added. These tests can include testing of sub functions that ados invoke while interacting with the hardware. Pre-written test fixtures allow for easy integration of custom tests.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-WECPR04

Advantages of pyTest

By utilizing various command line switches of pyTest, the failed tests are extremely well documented with an exact reason of mismatch in return value and expected value as well as tracebacks. pyTest avails plug-in installations such as coverage.py and hypotheses. Coverage is a tool to analyse the amount of code that has been tested. It is good for generating HTML reports.

Challenges of Automation in Controls

Several bottlenecks of automation had to be overcome because controls system consists of the various hardware and software components and the availability of these components depend upon the operational status. Specifically, hardware components are not always available for testing and simulating the hardware components require extra effort and can require software development . Also, control system of measurement and data and control system is non-deterministic, writing tests based on expected values can requires a lot of tolerance. Finally, in case of CAD the ADO infrastructure is written in a domain specific language which makes it harder to interface with industry tools.

CONCLUSION

A sample ado comprising of all parameter types was made to run through the test routine. Several parameters missing properties were uncovered pyTest provided a seamless test framework creation software that can be used used for quality assurance, benchmarking, as well as regression testing. The ability to customise every test session is valuable.

REFERENCES

[1] L.T. Hoff and J.F.Skelly, "Accelerator Devices at Persistent Software Objects," *Nucl. Instr. and Meth. in Phys. Res. A,* vol. 352, pp. 185-188, Dec. 1994. doi:10.1016/0168-9002(94)91494-X

[2] Parametrizing fixtures and test functions, https://docs.pytest.org/en/latest/parametrize.html