Robot Framework
Generic Test-Automation

Test Automation & Concepts

Introduction to Robot Framework

Ell-i Robot Framework & Targets

Test Automation & Concepts

- Automated tests, handy tool for regression
- Dedicated software to automate repetitive tasks
- Used in agile software development methodology
  - *Acceptance Testing*
    - *Progress in right direction*
  - *Accepted Test Driven Development*
    - *Encompasses acceptance testing*
    - *Coding based on acceptance tests*

Introduction to Robot Framework

- A Test Framework
- Easy-to-use tabular test data syntax
- Test libraries extension capabilities
- Supporting tools & libraries

A Test Framework

- Integrated automation system
- Sets rules of automation process
- Includes IDE, function libraries, test data sources and other utilities
- Simplifies test development and automation process
- Responsible for providing:
  - *Test data syntax or format*
  - *Test setup and closeup for Device under Test (DuT)*
  - *Executing or running the tests*
  - *Documentation & reporting the results*

Easy-to-use tabular test data syntax

- Test data file formats
  - *HTML format*
  - *TSV format*
  - *Plain Text format*
  - *ReStructured Text format*
- Keyword-driven testing approach
  - *New keywords from existing keywords*
- Test data styles
  - *Keyword-Driven style*
  - *Data-Driven style*
  - *Behavior-Driven Style*

Tabular Test Data (reStructured text format)

DigitalRead.rest

```
================ ==============================
    Setting          Value
================ ==============================
Library          BuiltIn
Library          ${TESTPATH}/DigitalRead.py
Test Setup       \
Test Teardown    \
================ ==============================

============== =============
   Variable        Value
============== =============
${PIN}         0
${HIGH}        1
${LOW}         0
============== =============

==================== ============== ================ =================
    Test Case           Action        Argument        Argument
==================== ============== ================ =================
Read high from pin   ${HIGH}=       Read High        ${PIN}
Check high from pin  Check High     ${PIN}           ${HIGH}
Read low from pin    ${LOW}=        Read Low         ${PIN}
Check low from pin   Check Low      ${PIN}           ${LOW}
==================== ============== ================ =================

============= ================= ======== =========
   Keyword    \                 \        \
============= ================= ======== =========
Check High    [Arguments]       ${pin}   ${high}
\             Should Be Equal   ${pin}   ${PIN}
\             Should Be Equal   ${high}  ${HIGH}
Check Low     [Arguments]       ${pin}   ${low}
\             Should Be Equal   ${pin}   ${PIN}
\             Should Be Equal   ${low}   ${LOW}
============= ================= ======== =========
```

Line 1, Column 1 Tab Size: 4 reStructuredText

Keyword-driven testing approach (1)

- Also known as table-driven or action word based testing
  - *Suitable for both manual and automated testing*
  - *Separates tests design, test execution & programming tasks*
  - *Keywords symbolizes functionality to be tested*

Pros:

- Maintenance is low:
  - *Test cases are concise*
  - *Test cases are readable*
  - *Test cases easy to modify*
  - *New keywords from existing ones*

Keyword-driven testing approach (2)

- Keyword re-use across multiple test cases
- Independent of tools or programming languages
- Devision of Labor
  - *Test case construction – lesser tool/programming skills & strong test domain expertise*
  - *Keyword implementation – strong tool/programming skills & lower test domain expertise*
- Abstraction of layers

Cons:

- Longer time to market
- Moderately high learning curve initially

Test Data Style Examples

- Keyword-Driven Style
  - *Work-flow tests*
  - *Test case constructed form several keywords with arguments*
  - *Test data uses tabular syntax e.g. rows & columns*

Test Case	Action	Argument	Argument
Valid Login	Open Login Page		
	Input Name	demo	
	Input Password	mode	
	Submit Credentials		
	Welcome Page Should Be Open		
Setting Variables	Do Something	first argument	second argument
	${value} =	Get Some Value	
	Should Be Equal	${value}	Expected value

- Data-Driven Style

  - *Tests work-flow similar to keyword-driven*

  - *Varying input data*

Data-driven testing example

Setting	Value	Value	Value
Test Template	Login with invalid credentials should fail		

Test Case	User Name	Password	
Invalid User Name	invalid	${VALID PASSWORD}	
Invalid Password	${VALID USER}	invalid	
Invalid User Name And Password	invalid	whatever	
Empty User Name	${EMPTY}	${VALID PASSWORD}	
Empty Password	${VALID USER}	${EMPTY}	
Empty User Name And Password	${EMPTY}	${EMPTY}	

- *High-level keyword hides actual test work-flow*

- *Useful for testing same scenario with different input/output data*

Data-driven test with multiple data variations

Test Case	User Name	Password
Invalid Password	[Template]	Login with invalid credentials should fail
	invalid	${VALID PASSWORD}
	${VALID USER}	invalid
	invalid	whatever
	${EMPTY}	${VALID PASSWORD}
	${VALID USER}	${EMPTY}
	${EMPTY}	${EMPTY}

- Behavior-Driven Style
  - *Test cases for non-technical project stakeholders*
  - *Popular Given-When-Then style*

Example test cases using behavior-driven style

Test Case	Step
Valid Login	Given login page is open
	When valid username and password are inserted
	and credentials are submitted
	Then welcome page should be open

- Behavior-Driven Style
  - *Test cases for non-technical project stakeholders*
  - *Popular Given-When-Then style*

Example test cases using behavior-driven style

Test Case	Step
Valid Login	Given login page is open
	When valid username and password are inserted
	and credentials are submitted
	Then welcome page should be open

Test libraries extension capabilities

- Many existing test libraries
- Often a need to extend or create new test library
- Simple and straight forward API
- Robot framework has three different test library APIs:
  - ***Static API***

    Module/class with methods which maps directly to keyword names
  - ***Dynamic API***

    Classes that implement a method to get names of keywords and a method to execute these keywords. The names of keywords and how they execute are determined dynamically at runtime.
  - ***Hybrid API***

    A hybrid between the static & dynamic API. The method telling what keywords to implement dynamically, but the keywords must be available directly.

Python scripts (test libraries) for keywords

DigitalRead.py

```python
"""
TEST CASES START
"""

import digitalRead

#print digitalRead.readHigh()

def read_high(pin):
    """Read 'HIGH' from pin"""
    #print 'Here it uses test library to interact with Ell-i runtime C code
    #"""Also returns value for pin"""
    return digitalRead.readHigh(pin)

def check_high(pin, high):
    """Check 'HIGH' from pin"""
    #print 'Here it uses test library to interact with Ell-i runtime C code

"""
*****************************************************************************
"""

def read_low(pin):
    """Read 'LOW' from pin"""
    #print 'Here it uses test library to interact with Ell-i runtime C code
    #"""Also returns value for pin"""
    return digitalRead.readLow(pin)

def check_low(pin, low):
    """Check 'LOW' from pin"""
    #print 'Here it uses test library to interact with Ell-i runtime C code

"""
TEST CASES END
"""

```

Line 1, Column 1 Spaces: 4 Python

Supporting tools & libraries

- **Standard**
  - *Builtin*
  - *OperatingSystem*
  - *Screenshot*
  - *Telnet*
  - *String*
  - *Collections*
  - *Remote*
  - *Dialogs*
  - *XML*
  - *Process*

- **External**
  - Android & iOS libraries
  - Archive Library
  - AutoItLibrary
  - Database Library
  - Eclipse Library
  - FTP & HTTP & SSH Libraries
  - Selenium 1&2 Libraries
  - Swing Library
- **Other**
  - APIs for creating test libraries

- *Example, setting table for test data include libraries*

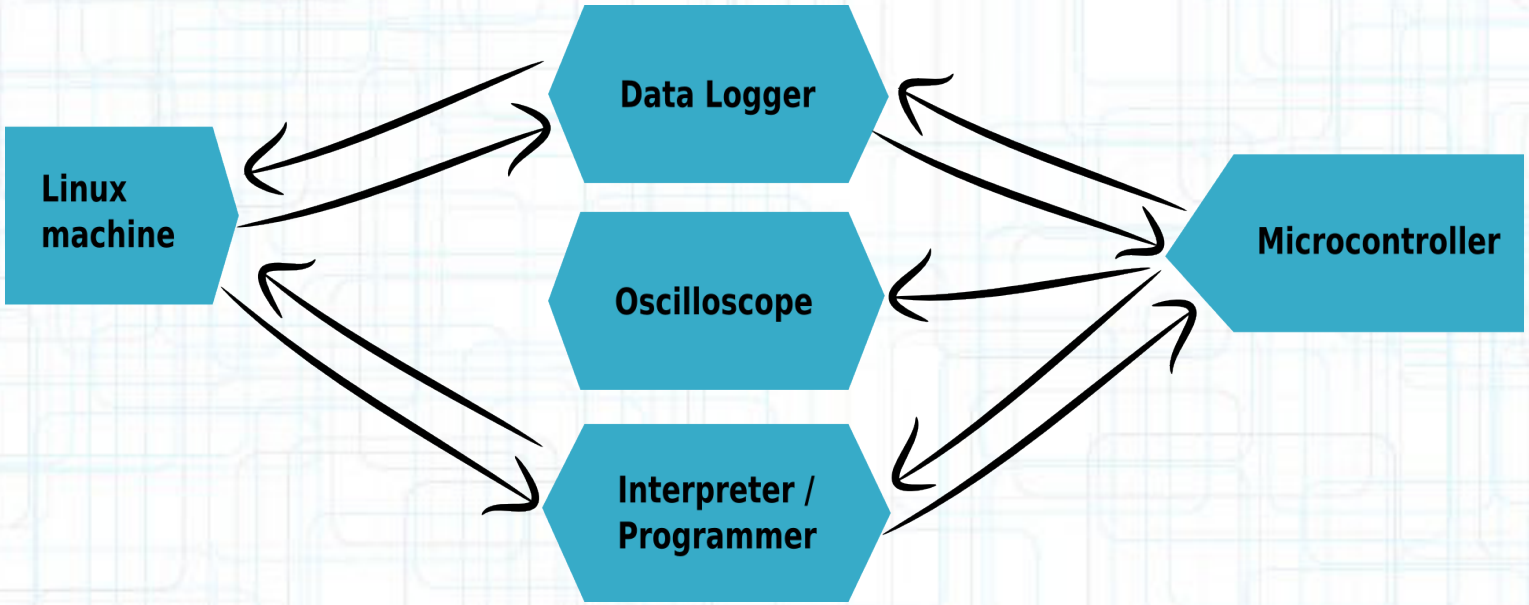
Setting	Value
Library	BuiltIn
Library	*OperatingSystem*
Library	*String*
Library	Android
Library	${TESTPATH}/DigitalRead.py

Ell-i Robot Framework & Targets

- Targets
- Tabular Test Data (reStructured text format)
- Python scripts (test libraries) for keywords
- Python-C extension (C code integration)
- Example python script for running tests

Targets

- Test suites for Arduino APIs
- Testing on Ell-i emulator against test suites
- Testing Ell-i hardware against test suites
- Later at some point, using Jenkins, server for testing daily builds


Linux machine
Data Logger
Oscilloscope
Interpreter / Programmer
Microcontroller

Tabular Test Data (reStructured text format)

DigitalRead.rest

```
================ ==============================
    Setting          Value
================ ==============================
Library          BuiltIn
Library          ${TESTPATH}/DigitalRead.py
Test Setup       \
Test Teardown    \
================ ==============================

============== =============
   Variable        Value
============== =============
${PIN}         0
${HIGH}        1
${LOW}         0
============== =============

==================== ============== ================ =================
    Test Case           Action        Argument        Argument
==================== ============== ================ =================
Read high from pin   ${HIGH}=       Read High        ${PIN}
Check high from pin  Check High     ${PIN}           ${HIGH}
Read low from pin    ${LOW}=        Read Low         ${PIN}
Check low from pin   Check Low      ${PIN}           ${LOW}
==================== ============== ================ =================

============= ================= ======== =========
   Keyword    \                 \        \
============= ================= ======== =========
Check High    [Arguments]       ${pin}   ${high}
\             Should Be Equal   ${pin}   ${PIN}
\             Should Be Equal   ${high}  ${HIGH}
Check Low     [Arguments]       ${pin}   ${low}
\             Should Be Equal   ${pin}   ${PIN}
\             Should Be Equal   ${low}   ${LOW}
============= ================= ======== =========
```

Line 1, Column 1 Tab Size: 4 reStructuredText

Python scripts (test libraries) for keywords

DigitalRead.py

```python
"""
TEST CASES START
"""

import digitalRead

#print digitalRead.readHigh()

def read_high(pin):
    """Read 'HIGH' from pin"""
    #print 'Here it uses test library to interact with Ell-i runtime C code
    #"""Also returns value for pin"""
    return digitalRead.readHigh(pin)

def check_high(pin, high):
    """Check 'HIGH' from pin"""
    #print 'Here it uses test library to interact with Ell-i runtime C code

"""
*****************************************************************************
"""

def read_low(pin):
    """Read 'LOW' from pin"""
    #print 'Here it uses test library to interact with Ell-i runtime C code
    #"""Also returns value for pin"""
    return digitalRead.readLow(pin)

def check_low(pin, low):
    """Check 'LOW' from pin"""
    #print 'Here it uses test library to interact with Ell-i runtime C code

"""
TEST CASES END
"""

```

Line 1, Column 1 Spaces: 4 Python

Python-C extension (C code integration)

digitalRead.c

```c
#include "wiring_digital.h"
#include <Python.h>

static PyObject* readHigh(PyObject* self, PyObject* args) {
    char* pin;
    if (!PyArg_ParseTuple(args, "s", &pin)) {
        return NULL;
    }

    int pinNum = atoi(pin);
    //char high[10];
    //sprintf(high, "%d", 1);


    pinMode(pinNum, 3);
    digitalWrite(pinNum, 1);
    int high = digitalRead(pinNum);


    return Py_BuildValue("s", high);
}

static PyObject* checkHigh(PyObject* self, PyObject* args) {
    return Py_BuildValue("s", "checkHigh, Python extensions!!");
}

static PyObject* readLow(PyObject* self, PyObject* args) {
    char* pin;
    if (!PyArg_ParseTuple(args, "s", &pin)) {
        return NULL;
    }

    int pinNum = atoi(pin);
    char low[10];
    sprintf(low, "%d", 0);

    return Py_BuildValue("s", low);
}

static PyObject* checkLow(PyObject* self, PyObject* args) {
    return Py_BuildValue("s", "checkLow, Python extensions!!");
```

Line 1, Column 1

Example python script for running tests

DigitalRead.py

```python
#!/usr/bin/python

import os
from subprocess import call


TESTNAME='DigitalRead'
TESTPATH=os.environ['HOME']+'/Ell-i/Ell-i-Pybot/ELL-i-PyBot-Tests/test-scripts/'

print TESTPATH

call(
    'pybot'+
    ' --variable TESTPATH:'+TESTPATH+
    ' --log ../test-results/'+TESTNAME+'/log.html'+
    ' --report ../test-results/'+TESTNAME+'/report.html'+
    ' --output ../test-results/'+TESTNAME+'/output.xml'+
    ' ../test-suites/ELL-i-PyBot-Tests.wiki/'+TESTNAME+'.rest',
    shell=True
    )
```

Line 1, Column 1 Tab Size: 4 Python