Export Control - Security classification
No Export Licence Required - Not Protectively Marked

TECHNICAL REPORT

A Rolls-Royce plc and Goodrich Corporation joint venture

Discipline or report series#10 Student Projects	**Date**#40 September 2010
Document number N/A	**Issue** 1

Title#15

Procedure to Python Translator Definition Document

Summary#60

Summary#60 This document serves to provide the students of Purdue University Computer Science course 490 with guidelines and requirements for developing a software tool that will translate verification procedures into Python scripts that will be run in a real-time simulation environment. **Approval signatures**

Approval signatures

Author#20 Chris Pfeiffer Verification Engineer	**Signature**	**Approved by** Aditya Mathur Professor of Computer Science	**Signature**
Approved by Chris Selby Software Engineer	**Signature**	**Approved by**	**Signature**

Additional keywords#90 System Verification, Python, Real Time Simulation	**Retention category** **B**

Circulation

Spring 2011 CS490 Students

Table of Contents

Change History

Issue	Author	Date	Comment
1	Christopher Pfeiffer	November 2010	First issue
2	Christopher Pfeiffer	January 2011	Change to Python Syntax

Definitions

Term/Acronym	Definition
AEC	Aero Engine Controls
EEC	Electronic Engine Controller
RTS	Real Time Simulator
SGT	Small Gas Turbine
VCP	Verification Cases and Procedures

Referenced Documents

Reference	Title
JSGT/0040	System Verification Procedures Standard

1 Introduction

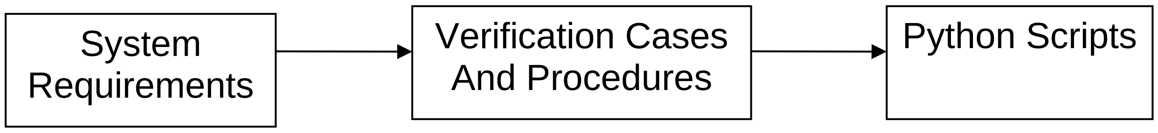
This document serves to provide the students of Purdue University Computer Science course 490 with guidelines and requirements for developing a software tool that will translate verification procedures into Python scripts that will be run in a real-time simulation environment. Sections 2 and 3 contain details and examples of Verification Cases & Procedures and Python Scripts respectively. Section 4 details the project requirements.

1.1 Background Information

At Aero Engine Controls (AEC), the System Verification Team is responsible for the “Black Box” testing of the Electronic Engine Controller (EEC) (i.e. the brains of a gas turbine engine). The System Verification team produces Verification Cases and Procedures (VCP) to verify that the EEC meets all of its System Requirements. The System Verification Team verifies that the correct outputs are produced by the EEC for a given stimulus as determined by the VCP. Many of the procedures produced by the System Verification team are simulated on the Real Time Simulator (RTS) using Python Scripts. The RTS acts like a flight simulator for the EEC while the Python Scripts acts like the pilot. For each procedure, there is a different “pilot” providing different combinations of inputs.

An illustration of the process from System Requirement to Python Script is shown below. Verification Engineers add value by translating the System Requirements into Verification Cases and Procedures. The translation from Verification Cases and Procedures to Python Scripts can be automated. The capability to auto-translate verification procedures into Python scripts has an immediate business effect as it reduces the man-hours required for verification and consequently greatly reduces the verification cost to engine programs.

FIGURE 1: Process from Requirements to Python Scripts



1.2 Purpose of the Project

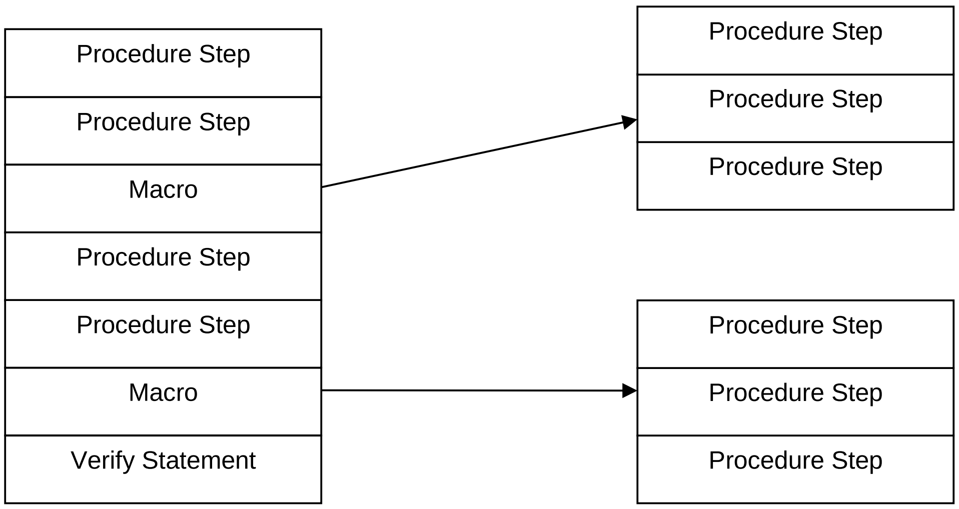
The process of standardizing the verification procedures is already underway at AEC so that all Verification Cases and Procedures will be written according to the System Verification Procedures Standard (reference JSGT/0040). Also already underway is the development of Python Macros which can be called by Python Scripts to run common procedures (e.g. starting the engine, throttling to takeoff speeds, and flying to some altitude; at which point a few tests can be simulated). These macros enable the ability to standardize simple procedures that can be used throughout the testing process.

The purpose of this project is to introduce the students of Purdue’s college of Computer Science to the concepts of “Black Box” testing of highly regulated embedded software, and to spearhead the development of a highly valuable tool for AEC that will automatically translate procedures into Python Scripts.

2 Verification Cases and Procedures

Procedures are written by verification engineers and are a series of steps and conditions that the verification engineer believes will verify the requirements of the control system. The requirements that are being verified by a procedure are linked to that procedure. Hundreds of procedures are often needed to verify that a control system meets all of its requirements. Each procedure may verify one or more requirements. Some requirements may need more than one procedure to show that they are fully met. Each procedure step can be one of three types of steps: A procedure step starting with a standard verb, a Macro call, or a verify/ensure statement. A procedure step starting with a specified verb is the most basic building block for a procedure. A macro is a pre-determined function that is a combination of highly re-used procedure steps that start with the standard verbs. An ensure/verify statement serves the purpose of stating what is expected to happen as a result of the procedure steps that have been performed. Ensure/Verify statements will be included in the python script, but only as a comment. The Verification Cases and Procedures document is platform independent. This means that it could be run as a procedure on either the Real Time Simulator or any other verification vehicle such as an engine test cell.

FIGURE 2: Example Composition of a Procedure



2.1 List of Verbs

As mentioned, the most basic building blocks of a procedure begin with a verb that is a member of a list of verbs that are allowable when writing a procedure. The following is that list of Verbs per the AEC Procedures Standard and what it expects.

The procedure shall use the term "**Insert**" followed the fault type to specify the creation of an electrical fault condition. E.G. **Insert** [Fault Type] **on** [Signal_ID]. There are seven allowable states for Fault Type. They are as follows: NoFault, OpenCircuit, GroundFault, TestFault, FaultA, FaultB, and FaultC.

The procedure shall use the term "**Ramp**" for all linear excursions in either the deceasing or increasing direction. E.G. **Ramp** [Parameter] **to** [Value] [Units]**over** [Time Duration].

The procedure shall use the term "**Set**" to command a change in the state of a switch or other variable that requires a step change in value. Example: **Set** [Parameter] **to** [Value] [Units]

The procedure shall use the term "**WaitTime**" followed by a specific time before executing the next procedure command. Example: **WaitTime** [Time Duration]

The procedure shall use the term "**WaitUntil**" to indicate a specific time period dependant on a specific, logical event that must occur before executing the next procedure command. Note: there will be a required time limit specified to ensure that, if the condition is not met, the next command will be executed, **WaitUntil** [Logical Event] **or** [Time Out Duration]

The procedure shall use the term "**WaitWhile**" to indicate a specific time period followed by a specific logical condition that must cease before proceeding to the next procedure command. Note: there will be a required time limit specified to ensure that, if the condition is not met, the next command will be executed, **WaitWhile** [Logical Event] **or** [Time Out Duration]

2.2 Example

The following is a very simple example of a possible requirement and a procedure that a verification engineer could have written to verify the requirement via black box testing.

Requirement: When collective pitch is ramped from 40 degrees to 80 degrees in 5 seconds the control system shall control Np to 100% ± 2%

Procedure:

Set Power to ON

Waittime 20 Seconds

Start_Engine_To_Ground_Idle

Ramp Collective Pitch from to 40 degrees over 20 seconds

Waittime 15 seconds

Ramp Collective Pitch to 80 degrees over 5 seconds

Verify that Np stayed within 2% of 100% during ramp

In this situation the engineer read the requirement and came up with a set of procedure steps that would prove that requirement is satisfied by the control system. That process represents the first arrow of Figure 1.

3 Python Scripts

Once a procedure is written, it is often run on the Real Time Simulator if it does not require an actual engine test. The RTS can reduce the cost and time associated with verifying the control system requirements. In order for a procedure to be run on the RTS, the RTS needs to be controlled to create the conditions of the procedure. A new vehicle for this at AEC is a python script. The python script corresponding to the above procedure is given in the next section.

3.1 Example

The following is an example python script that responds to the procedure in section 2.

Python Script:

```
POWER.set(1)

WaitTime(20)

start_engine_to_ground_idle

CP.Ramp(40, 20)

WaitTime(15)

CP.ramp(80, 5)

#Verify that Np stayed within 2% of 100% during ramp
```

Each line in this script corresponds to a line in the procedure. It is simply reformatted to meet the python language specifications so that it can be run on the RTS. Note that the Verify statement is directly copied but commented and the macro call, Start_Engine_to_Ground_Idle is directly copied but in all lower case.

3.2 Macros

The Python script will call the Macros as functions. The functions will be stored in other modules but will be able to be called from the Python script due to import statements at the beginning of each test script. Each function or Macro will simply be a series of the most basic commands that start with the specified verbs.

4 Project Requirements

The role of the Procedure to Python Translator is to fill in the second arrow of Figure 1. Engine expertise and procedure design from system level requirements (the first arrow) requires a verification engineer, but a simple translation from a platform independent standard procedure to a Python Script to be run on the RTS (the second arrow) should be automated to save time and check to ensure the procedures are written according to the standard.

4.1 User Interface

The Procedure to Python Translator shall be in the form a functional user interface that can be used by verification engineers who are unaware of the characteristics of the source code that makes up the project. The user interface shall allow the user to specify the location of the procedure it wants to convert and the location to which the converted python files will be saved. It should also contain features that allow the user to edit the list of macros, edit the header of python files, and edit nouns.

4.2 Translation

The Procedure to Python Translator shall translate each line of the procedure that is written correctly according to standard into a line of python code. There will be three types of lines that will need to be translated: Lines that start with a standard verb, Macros, and Ensure/Verify Statements.

4.2.1 Lines that Start with a Standard Verb

Lines that start with a standard verb shall be translated according to the following rules:

The Noun that the verb is acting upon shall start the line of code. If there is not a noun, e.g. WaitUntil, the verb shall start the line.

Immediately following the Noun, if there is one, shall be a period. The period shall be immediately followed by the verb with its first letter capitalized. Immediately following the verb shall be the arguments passed to the function, separated by commas and enclosed with parentheses.

Example 1: Standard Verb

Ramp Collective Pitch to 40 over 5 seconds **translates to** CP.Ramp[40, 5]

Example 2: Standard Verb No Noun

Waittime 20 Seconds **translates to** WaitTime(20)

Example 3: Standard Verb with a Logical Event

WaitUntil Ng > 80% or 5 seconds **translates to** WaitUntil(Ng>80, 5)

4.2.2 Noun Translation

Noun translation will not necessarily be a 1:1 process. For example collective pitch, which is supposed to be translated to cp, could be referred to in a procedure step as CP, Collective Pitch, or CPAircraft. The Procedure to Python Translation tool needs to be robust enough to recognize at least 3 forms of one noun and translate all three of those forms into the correct python variable. All three forms of collective pitch in the above reference would translate to cp in the python script. The nouns and their translations can change from application to application so there must be a mechanism by which the end user can configure the python variables and at least three references to that variable that can be used in the procedure steps.

4.2.3 Macros

The role of the Procedure to Python Translator when it comes to macros is to contain an easily editable through the interface list of macros, recognize those macros in a procedure line, and print the macros along with any arguments. Macros can be added at any time so it is extremely important that it is easy for the user to add a Macro along with its expected argument(s) to the Procedure to Python Translator's list of Macros it can interpret. An example macro translation is as follows:

Toggle Power **translates to** toggle(power)

The python script will call the function toggle from another python module that will contain a set of simple steps. No knowledge of this module or steps is required by the Procedure to Python Translator.

4.2.4 Ensure/Verify Statements

Ensure/Verify statements will be the easiest for the Procedure to Python Translator to handle. They shall simply be translated verbatim as a comment in the python script. The only modification is the # symbol at the beginning of the line. An example is as follows:

Ensure Ng reached 100% **translates to** #Ensure Ng reached 100%

The translator has no responsibility to further analyze or translate any other aspects of this type of line in a procedure

4.2.5 Error Accommodation

The Procedure to Python Translator will surely come across procedure steps that have not been written uniformly according to the standard. How well the tool handles these lines will have a large impact on the tool's ability to increases the efficiency of the verification engineers. The tool shall notify the user whenever it comes across a line that cannot be translated. The goal of the tool is to make it as easy to fix and edit the lines that return errors as possible. This is a large part of the design required by the programmers of this tool. The designers of this tool are tasked with creating the paradigm for finding, reporting, and fixing errors.

4.3 I/0 Format

The Procedure to Python Translator shall accept a .csv file with each procedure step separated by a new line and each attribute of a procedure step separated by a comma. The procedure step will be the second attribute. An example of this .csv will be provided. The output file shall be a .py Python Script that can be run on the RTS. The user interface shall allow the user to put something at the top of the file. This is especially important if batching is an available feature because many files will contain the same imports and other information at the top of the file. This shall be able to be done automatically by the translator.