PYTHON SOFTWARE FOR MEASURING WAVELENGTH AT OPTICALLY PUMPED POLARIZED ION SOURCE (OPPIS)*

P. Kankiya[†], J. Jamilkowski, Brookhaven National Laboratory, Upton, NY 11973 USA

Abstract

Often diagnostic tools are packaged with proprietary software and it is challenging to integrate with native environment. The HighFinesse Angstrom Wavemeter used at OPPIS experiment for laser wavelength measurement is controlled using commercial software not supported by RHIC style controls. This paper will describe the integration of such a complex system and use of python for cross platform data acquisition.

INTRODUCTION

The control system environment at BNL's Collider Accelerator Division is a rich repository of control system applications developed in-house and supports complicated supervisory controls and data acquisition tasks, although is limited to supporting these tasks in the Linux environment. Very rarely do we have a requirement to support systems that require software development in a non-UNIX based platforms. We have explored a few solutions in meeting this limitation [1]. In this paper a solution to integrate windows binaries using python is presented. By implementing this solution it will be possible to log data and generate alarms, use synoptic displays etc after gathering measurements from foreign binaries.

SYSTEM IN USE

The polarized beam for RHIC spin physics experimental program is produced in the Optically-Pumped Polarized H-Ion Source (OPPIS), the RHIC polarized H-ion source is upgraded to higher intensity (5-10 mA) and polarization by using a very high brightness fast atomic beam source for enhanced luminosity RHIC operation. Pulsed laser is used to optically pump the rubidium vapour. It is important to measure laser wavelength synchronous to AGS cycle to ensure polarization in desired direction [2]. A high precision device that can measure pulsed laser parameters with high accuracy at a fast rate is needed. To meet this requirement Armstrong meter model shown in Figure 1 was procured with a vision that it will be integrated into RHIC control applications.

Data Acquisition Procedure

Wavelength meter is setup in pulsed operation and externally triggered mode. These control set up tasks are performed by using proprietary software supplied with the hardware. Operator can provide preferred unit of measurement. Operation is started by adjusting the exposure manually or selecting automatic manner.

*Work supported by Brookhaven Science Associates, LLC under Contract No. DE-AC02-98CH10886 with the U.S. DOE
† pkankiya@bnl.gov

For the purpose of integrating measurement results from this device into custom applications, all program values and settings are accessible for user via calls to a dynamic linked library (DLL) called wlmData.dll routines. The wlmData.dll uses shared memory for inter-process communication. The software requires that the server (the Wavelength Meter application) and the clients (any access and control programs) access the same DLL file and not just identical copies which leads to limiting the use of the interface software to one instance at a given time. Elimination of restriction of use of DLL by one client at a time and integrating the whole setup with in-house controls is important to make use of existing infrastructure and correlate data measurements.

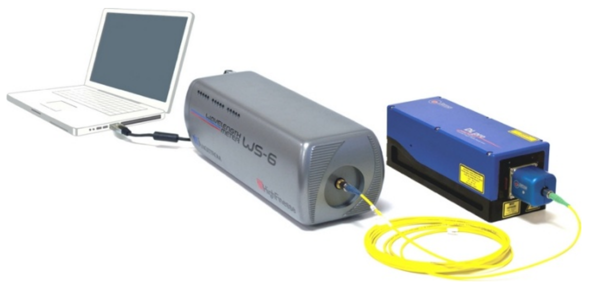


Figure 1: High Finesse Wavelength measuring device interfacing a windows computer and a light source.

SOLUTION

There are three common approaches to solving the aforementioned problem:

1. Write your own low level library to communicate with the device.
2. Write an extension in C code to bridge the gap between the libraries and use a cross platform compiler for generating supported executable format.
3. Wrap the library using your **foreign function interface** (FFI) support. Foreign refers to the fact that the functions come from another language and environment.

Because the wlmdata.dll is not an open source file, option one cannot be implemented. Option 2 is more time consuming and limitations has been addressed in another attempt described in paper [1]. Option 3 is considered to be a pragmatic solution. From multiple foreign function

ISBN 978-3-95450-148-9

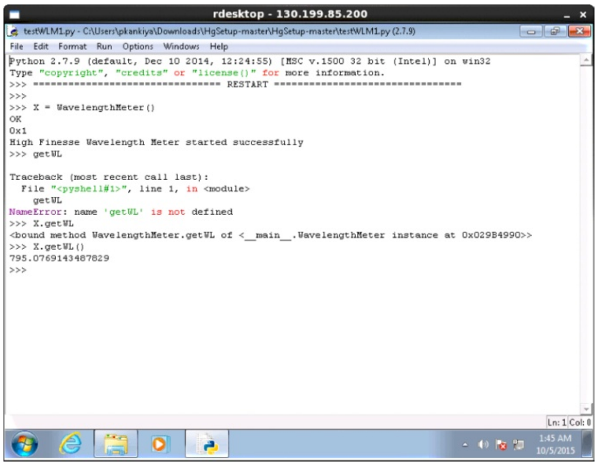
interface (FFI) languages available python was considered for deploying interface software due to availability of knowledge in the field.

IMPLEMENTATION

A python module known as ctypes [3] allows easy integration of C libraries directly without writing wrapper code in C. With help of ctypes the concerned DLL is imported into a python program called WLM. A class named wavelength meter is defined. All functions of the DLL are available for execution once an object of this class is instantiated. The source code provided with the DLL package contains header files with error codes and function definitions. The header files are re-written to be python modules. The return types of each method must be explicitly specified and must match the original definition. By not complying to translating C header to ctypes exactly will lead to segmentation faults at runtime.

Data acquisition is made simple by executing routines such as callback (CallbackProc/Ex) and wait-event mechanisms (WaitForWLMEvent/Ex) which are prototyped one to one on the python side with help of ctypes. The mapped functions are called using standard C calling conventions as opposed to windows stdcall calling conventions [3]. These routines receive any measurement results and WLM state changing information.

The data collected from the python interface program can be then packaged and sent across Linux environment with help of RPC module and adoIf protocol implemented in python [4].



```
Python 2.7.9 (default, Dec 10 2014, 12:24:55) [MSC v.1500 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
>>> X = WavelengthMeter()
OK
0x1
High Finesse Wavelength Meter started successfully
>>> getWL

Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    getWL
NameError: name 'getWL' is not defined
>>> X.getWL
<bound method WavelengthMeter.getWL of <__main__.WavelengthMeter instance at 0x029B4990>>
>>> X.getWL()
795.0769143487829
>>>
```

rdesktop - 130.199.85.200
testWLM1.py - C:\Users\pkankiya\Downloads\HgSetup-master\HgSetup-master\testWLM1.py (2.7.9)
File Edit Format Run Options Windows Help
Ln: 1 Col: 0
1:45 AM
10/5/2015

Figure 2: Simple object created from wavelength meter class and used to measure wavelength from get method.

DEVELOPMENT ENVIRONMENT

As described in a previous paper [1], to support C++ style ADO manager on windows we would have to port the existing code base on windows side with help of clear case remote client. Whereas, in case of what can be called a python manager a clearcase client is not required and any version control can be utilized due to non-dependency on legacy code. IDLE is used for editing and compilation of code. Figure 2 illustrates the use of python object to measure laser wavelength from an IDLE shell.

Python compilers are much lightweight applications as compared to CYGWIN like cross platform applications that have many OS dependencies and are complicated to install.

STATUS AND PLANS

A one to one map of wavelength meter package is developed in python with help of ctypes module. Measurement of data using the call-back and synchronous acquisition is tested. Work is under progress to port data to Linux side. This approach serves as a quickly deployable solution to integrating windows binaries to RHIC controls environment. For future projects it is advisable to find a solution such that developers do not have to fully repeat the C declarations. We plan to test this solution with operational data in the upcoming run of RHIC.

REFERENCES

[1] P. Kankiya, et al, “Integration of Windows binaries in the Unix-based RHIC control system environment.
[2] A. Zelenski, et al, “The RHIC polarised ion source upgrade with neutral ion source injector.
[3] https://docs.python.org/2/library/ctypes.html
[4] L.T. Hoff, J.F. Skelly, “Accelerator devices as persistent software objects”, Proc. ICALEPCS 93, Berlin, Germany, 1993.

ISBN 978-3-95450-148-9