PSFC/JA-09-07

MDSplus Objects – Python Implementation

Fredian, T.W., Stillerman, J.A., Manduchi, G*

* Consorzio RFX, Euratom-ENEA Association, Padova, Italy

Plasma Science and Fusion Center
Massachusetts Institute of Technology
Cambridge MA 02139 USA

This work was supported by the U.S. Department of Energy, Grant No. DE-FC02-99ER54512. Reproduction, translation, publication, use and disposal, in whole or in part, by or for the United States government is permitted.

MDSplus Objects – Python Implementation

T. Fredian [1)*], J. Stillerman [1)], G. Manduchi [2)]

1) Massachusetts Institute of Technology, 175 Albany Street, Cambridge, MA 02139, USA

2) Consorzio RFX, Euratom-ENEA Association, Corso Stati Uniti 4, Padova 35127, Italy

Abstract

MDSplus is a data acquisition and analysis software package used widely throughout the international fusion research community. During the past year, an important set of enhancements were designed under the project name of "MDSobjects" which would provide a common, powerful application programming interface (API) to MDSplus in programming languages with object oriented capabilities. This paper will discuss the Python language implementation of this API and some of the capabilities that this implementation provides for data storage and retrieval using the MDSplus system. We have implemented a new MDSplus Python module which exposes the MDSplus objects features to the language. The internal MDSplus programming language, TDI, has also been enhanced to be able to invoke Python commands from the TDI language. Now that Python is aware of the complex data structures in MDSplus such as Signals, the language becomes a very good candidate for applications ranging from data acquisition device support to analysis and visualization.

This work was supported by the U.S. Department of Energy, Cooperative Grant No. DE-FC02-99ER54512 and by the European Communities under the contract or Association between EURATOM/ENEA.

* Corresponding author. Tel: (617) 253-7623; fax: (617) 253-0627

E-mail address: twf@psfc.mit.edu.

1. Introduction

MDSplus[1,2,3] is a data acquisition, data handling and analysis system used widely in the fusion community. It provides numerous powerful capabilities to simplify the process of storing measurements or analysis results in archives and to enable scientists to retrieve these data using a wide variety of programming languages and data manipulation tools. It provides a powerful expression evaluator which can be used to manipulate the data on retrieval and expressions can be stored in the archive to define new data items which are computed when retrieved. MDSplus also provides a remote data access capability enabling scientists to access data from anywhere on the Internet. The MDSplus system runs on many computing platforms including Windows, MacOS, Linux and several other variants of UNIX. It is used at over 30 fusion experiment or modeling sites worldwide.

When MDSplus was designed in the late 1980's, it included many object oriented features such as complex data items which are essentially objects with named properties and methods for operating on the objects. However the application programming interfaces (API) developed for MDSplus did not expose these features since the primary API's were developed for non-object oriented languages. The only method for accessing these feature was the internal MDSplus expression evaluator language, TDI, which has somewhat limited capability as a programming language.

In 2008 a project was initiated by the authors to provide an object oriented API to MDSplus. This project, named "MDSobjects" is discussed in more detail in another paper presented at this same IAEA Topical Meeting. One of the programming languages targeted for an object oriented interface to MDSplus was Python[4]. This paper will describe some of the features of the Python MDSplus objected oriented API and some ways it can be used to extend the capabilities of the MDSplus system.

2. MDSplus Objects API

The MDSplus objects API consists of numerous classes of objects. The three main super classes are the Tree class, the TreeNode class and the Data class. The Tree class corresponds to an MDSplus data storage set of files containing a hierarchical structure of nodes which may contain data. Establishing an instance of a Tree class opens the MDSplus tree for access. The Tree class provides methods for finding nodes in the tree structure.

The TreeNode class represents a single node of an MDSplus tree. The TreeNode class has numerous properties and methods which permit access to information such as when data was stored in the node, the length of the data, the data type of the data stored. Some of the methods enable the programmer to easily retrieve or store data in a tree node.

The Data class is the super class of any data that can be stored in an MDSplus TreeNode. It provides a long list of methods and operator overrides for manipulating MDSplus data. Numerous subclasses of the Data class are available which represent the entire set of MDSplus data types. These include arrays and scalars of basic data types such as various sized integers, floats and textual data. More complex subclasses of Data are available to represent some of the more complex data types used in MDSplus such as Signal, Functions, Dimensions, Ranges and many more.

3. Python Implementation

The Python implementation of the MDSplus objects consists of a single Python package called "MDSplus". This package will be made available as part of the normal MDSplus software downloads from the MDSplus web site, http://www.mdsplus.org/ . Updates to the package will be available using the easy_install capabilities provided by the Python setuptools package which uses the Python Package Index[5] web site. The MDSplus package is written entirely in Python. It uses a package called ctypes[6] to call into the MDSplus libraries. The MDSplus Python package does require that the full MDSplus installation kit be installed on the system before it can be used. The MDSplus package also uses the NumPy[7] package which provides efficient manipulation of numeric and textual scalars and arrays.

The MDSplus Python package is still under development but most of the commonly used functions are complete and functional. The package has been installed on tested with Python versions 2.4 and 2.5 on both Windows and Linux platforms. The package contains some test modules for testing before installing. These test modules

will be expanded to test as many of the MDSplus capabilities.

4. Sample Python session

The MDSplus package makes it very easy to utilize the capabilities of the MDSplus system. Included below are some simple examples which demonstrate how easy it is to use.

```
>>> from MDSplus import *
>>> tree=Tree('cmod', 1080326005)
>>> ip=tree.getNode('\\IP')
>>> print ip
 \MAGNETICS::IP
>>> print ip.dtype_str
 DTYPE_SIGNAL
>>> ip_data=ip.record
>>> print ip_data
 Build_Signal( Build_With_Units( …
>>> print type(ip_data)
 <class 'MDSplus.compound.Signal'>
>>> print ip_data.data()
 [-1248.51367188 -1246.48620605  -935.371521 ...
```

This example demonstrates the opening of an MDSplus tree and the referencing of the plasma current node called \IP. That node is stored as an MDSplus signal in the C-Mod experiment tree. In the example above the *tree* variable becomes an instance of the Tree class, the *ip* variable an instance of a TreeNode class and the ip_data variable an instance of the Signal class which is a subclass of the Data class.

5. Calling Python from TDI

The built in expression evaluator language in MDSplus called TDI has been enhanced to enable the execution of Python statements during expression evaluation. A simple PY function has been added which takes a single argument of the text of a Python statement. Data can be passed to and from the Python environment to the TDI environment using TDI variables. There is a method of the Python Data class to set or get the values of these TDI variables. With the capability to call into Python from the MDSPlus expression evaluator one can utilize the powerful programming capabilities provided by the Python language within the context of the MDSplus expression evaluator.

6. New MDSplus Data Types

Two new MDSplus data types have been added to complement the capabilities of Python; DTYPE_LIST and DTYPE_DICTIONARY. These two types correspond to the MDSplus Data classes of Dictionary and List in Python which are subclasses of the Python dict and list classes. Dictionaries and list are used frequently in programming with the Python language. A Python list is simply an object which lets you collect heterogeneous objects in an ordered list. A Python dict is an object that lets you collect heterogeneous objects and provide them with indexes so you can reference these objects using a key value. The easiest way to explain this concept is to use a simple python example:

```
>>> mydict = {'a':42,'b':"This is a string"}
>>> print mydict['a']
  42
>>> print mydict['b']
  This is a string
```

In this example we construct a dict object by assigning string keys (a and b) to data objects (42 and 'This is a string'). Then objects contained in the dict object can be referenced by indexing the object using the keys.

Adding these data types to MDSplus enables the user to store lists and dictionaries in an MDSplus tree and then retrieve them and index into the lists or dictionaries. MDSplus already had the concept of lists heterogeneous objects implemented in its internals using a data structure called an APD which is an array of descriptors. Unfortunately there was no support in the TDI data language for constructing or manipulating this feature. This data structure was ideal for storing the new list and dictionary data types.

With the Dictionary class it is possible to store arbitrary structures of data in an MDSplus data file. To illustrate how this might be done we have included a simple example.

Construct a Dictionary instance with nested Dictionaries and store the data in the 'MYNODE' node of an MDSplus tree. Note that simply assigning the data item to the node's record property results in writing the data to the MDSplus tree.:

```
>>> from MDSplus import *
>>> s1=Dictionary({'signal':Signal(sig1val,None,sig1dim),
        'comment':'this is a comment'})
>>> s2=Dictionary({'signal':Signal(sig2val,None,sig2dim),
        'comment','this is another comment'}
>>> mydata=Dictionary({'sig1':s1,'sig2':s2})
>>> tree=Tree('mytree',shot)
>>> mynode=tree.getNode('\\TOP.DATA:MYNODE')
>>> mynode.record=mydata
```

Retrieve the Dictionary data in the 'MYNODE' node and access parts of the Dictionary instance using keys:

```
>>> from MDSplus import *
>>> tree=Tree('mytree',shot)
>>> mynode=tree.getNode('\\TOP.DATA:MYNODE')
>>> mydata=mynode.record
>>> s1sig=mydata['sig1']['signal']
>>> s1com=mydata['sig1']['comment']
```

Currently only the Python MDSplus package can be used to easily manipulate these new data types. We will explore adding TDI functions for accessing these data types as well as adding these data classes to MDSplus object implementations in other languages.

7. Remote Access with MDSplus Objects

In the current design and implementation of MDSplus objects there are no provisions for explicitly connecting to a remote MDSplus server to access data. Until this capability is added you are advised to use other packages which implement the thin client capabilities (mdsconnect, mdsopen, mdsvalue, etc.) such as the pmds[8] Python package. One can still use the MDSplus 'distributed client' mode to access remote data by specifying a server in the tree path definitions which tell MDSplus where to locate the MDSplus trees. The thin client connection is available through the TDI expression language however which is in turn accessible via the MDSplus objects. In fact, some remote fetch and remote store functionality is being explored utilizing the TDI thin client functions and the Dictionary and List classes of the Python MDSplus objects. Using this capability it is possible to construct a list of expressions to be evaluated on a remote host and to return the answers to those evaluations in a single transaction. Similarly it is possible to construct of dictionary of node names and data to send to a remote host in one transaction and then stored in MDSplus trees on the remote host. With this capability one could imagine much more efficient transmission of data especially when transfer data over high latency long distance Internet connections.

8. Python MDSplus Device Support

The MDSplus data acquisition system is very extensible. There are mechanisms for adding support for new data acquisition devices which require specialized code for configuring the devices and reading or write data from the devices and storing the results in the MDSplus trees. Prior to this MDSplus Python Objects work, there were two methods for implementing specialized device support; compiling and linking code into shared libraries or writing the support code in the MDSplus TDI programming language. During the implementation of the Python MDSplus objects package it was discovered that it would relatively simple to add the capability of developing device support written in Python. This capability was added to the MDSplus system so now one can develop device support using only the Python programming language. Since the MDSplus Python package lets you construct MDSplus data types natively using the Python language this provides a very powerful platform for developing the specialized code for supporting new data acquisition hardware. A tutorial on how to develop device support using the Python programming language is provided on the MDSplus web site.

9. Future Directions

It is anticipated that work will continue to enhance the MDSplus Objects implementation adding feature such as tree editing and perhaps some form of thin client methods or objects. It is hoped that the MDSplus Object API will be consistent across all of the object oriented languages that will be supported. As these implementations are performed it is expected that additional capabilities will be added to take advantage of some of the features of the different languages. If appropriate, these capabilities will then be adopted in the existing implementations.

10. Conclusions

The “MDSobjects” project was very successful. Initial experiences with interfacing MDSplus with object oriented languages have been quite positive and there appears to be a much more natural programming interaction with MDSplus using an object oriented approach. The object oriented characteristics designed into MDSplus over 20 years ago are finally exposed to languages that can take full advantage of them.

11. Acknowledgements

We have received numerous suggestions from MDSplus users worldwide which continue to lead us in directions to improve the system making it more useful for a wider variety of applications. We like to thank Tom Osborne from San Diego for his work on the original Python interface to MDSplus. We would also like to thank Brian Nelson from the University of Washington for his suggestions on the Python implementation.

---

[1] T.W. Fredian, J.A. Stillerman, M. Greenwald, “Data acquisition system for Alcator C-Mod”, *Rev. Sci. Inst.*, January 1997, **68**(1), pp 935-938.

[2] T. W. Fredian and J. A. Stillerman, "MDSplus: Current Developments and Future Directions", *Fus. Eng. Des.*, **60**, (2002), 229

[3] http://www.mdsplus.org/

[4] Python Programming Language, http://www.python.org/

[5] Python Package Index, http://pypi.python.org/

[6] Ctypes Python Package, http://python.net/crew/theller/ctypes/

[7] NumPy Package – http://numpy.scipy.org

[8] Pmds Python Package developed by Tom Osborne, http://diii-d.gat.com/~osborne/python