MDSplus Quality Improvement Project

T.Fredian[1], J. Stillerman[1], G. Manduchi[2], A. Rigoni[2], K. Erickson[3]

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
3 Princeton Plasma Physics Laboratory, Princeton, NJ 08543, USA

MDSplus is a data acquisition and analysis system used worldwide predominantly in the fusion research community. Development began 29 years ago on the OpenVMS operating system. Since that time there have been many new features added and the code has been ported to many different operating systems. There have been contributions to the MDSplus development from the fusion community in the way of feature suggestions, feature implementations, documentation and porting to different operating systems. The bulk of the development and support of MDSplus, however, has been provided by a relatively small core
developer group of three or four members. Given the size of the development team and the large number of users much more effort was focused on
providing new features for the community than on keeping the underlying code and documentation up to date with the evolving software development standards. To ensure that MDSplus will continue to provide the needs of the community in the future, the MDSplus development team along with other members of the MDSplus user community has commenced on a major quality improvement project. The planned improvements include changes to software build scripts to better use GNU Autoconf and Automake tools, refactoring many of the source code modules using new language features available in modern compilers, using GNU MinGW-w64 to create MS Windows distributions, migrating to a more modern source code management system, improvement of source documentation as well as improvements to the www.mdsplus.org web site documentation and layout, and the addition of more comprehensive test suites to apply to MDSplus code builds prior to releasing installation kits to the community. This work should lead to a much more robust product and establish a framework to maintain stability as more enhancements and features are added. This paper will describe these efforts that are either in progress or planned for the near future.

Keywords: Data Acquisition Systems, Data Management, Data Formats, MDSplus

Introduction

MDSplus[1][2] is a collection of libraries and applications used for acquisition, access and storage of scientific data. It provides a wide range of functionality including:

- Data acquisition from measurement devices; organization of the storage of both experimental measurements, analysis results and various metadata associated with those data items
- Access to the data and metadata from a wide variety of programming languages and

utilities
- Remote data access using a variety of transport mechanisms.

In the early 1980's, Tom Fredian and Josh Stillerman developed an data system called MDS for use on the Alcator and Tara fusion experiments at the Massachusetts Institute of Technology. That system was quickly adopted by several other fusion research facilities. It was also evaluated for possible use at two new experiments under construction, the RFX experiment in Padova, Italy and the ZTH in Los Alamos, New Mexico. Software developers at these two facilities had some interesting ideas for greatly enhancing the functionality of MDS so in 1987 a collaboration between MIT, RFX and LANL was initiated to develop a new data system providing significantly extended capabilities compared to MDS. Over the next 3 years the original core of MDSplus was developed and was ready for use at the startup of Alcator-C and RFX experiments in 1990. The ZTH experiment lost its funding and never began operation. Originally developed on the OpenVMS[3] platform, MDSplus has since grown in functionality and has been ported to numerous computing platforms. MDSplus is used by scientists and engineers worldwide mostly in the field of fusion energy research. Currently we're seeing upwards of 2500 downloads of MDSplus installation packages per year. This includes both new installations and updates to existing installations.

The development and support of MDSplus has been accomplished by a relatively small core group and much effort was targeted on adding new functionality to the system requested by the user community and providing support for a growing number of computing platforms. More recently we have increased the number of core developers adding more expertise in modern coding standards and compiler capabilities and have shifted the focus to improve the quality of the core MDSplus product utilizing a variety of techniques discussed in this paper.

OpenVMS origins

As mentioned above, MDSplus was originally developed on the Digital Equipment Corporation's OpenVMS operating system. This system provided a wide range of utilities which developers could build upon to provide specialized applications. MDSplus was designed and built to take advantage of the tools provided by the operating system. When it was later decided to port MDSplus to other operating systems it was anticipated that OpenVMS would be the predominant OS for MDSplus use. So rather than making major changes to the MDSplus code, the port to other operating systems was accomplished by emulating the utilities provided by OpenVMS. Surprisingly, at the time, OpenVMS use began to rapidly decrease and soon most sites migrated from OpenVMS to linux based operating systems. Today essentially all sites using MDSplus have moved off of OpenVMS. While MDSplus functions and performs well on linux systems, it's code base is still littered with OpenVMS concepts. Part of this quality improvement project is to remove unused sections of code specific to OpenVMS and to replace calls to OpenVMS library functions, which are emulated on the other platforms, with standard C library functions.

Rewrite OpenVMS based utilities

Porting MDSplus to non-OpenVMS platforms entailed emulating some major tools provided by OpenVMS. The best example of this was the implementation of the command line based utilities found in MDSplus. These were all based on the DCL[4], “Digital Command Language”, utility provide by OpenVMS which enabled a developer to describe the command syntax in a special language and build a command interpreter with the command parsing, syntax validation and execution code dispatching all provided by the DCL utility. To provide this same capability on non-OpenVMS platforms, an emulation of the DCL utility was implemented. The entire implementation of the utility was done from scratch and utilized some questionable techniques which turned out to be quite susceptible to compiler and platform differences. Only after running some modern compiler based code analysis features was it discovered that this code was making assumption about things like memory layout and call stacks that could produce intermittent failures or break entirely if the compilers chose to behave differently . As part of the quality improvement project, this entire DCL emulation utility has been rewritten using standard tools such as flex[5], bison[6] and xml[7] to provide the command definitions and command line parsing. One major additional benefit of this project was the addition of a “HELP” command for the utilities which provides detailed descriptions of the available commands. This help feature was in the original OpenVMS DCL utility but was omitted from the port to the other platforms since it was assumed that one could always log into their OpenVMS system if they needed it.

GNU compiler standardization

MDSplus was ported to many other operating systems in the late 1990’s when most other platforms had platform specific compilers each with their own idiosyncrasies. The MDSplus code became riddled with conditional compilation sections based on the particular compiler being used. Today, all of the platforms that MDSplus currently supports, use GNU compilers by default or at least have GNU compilers available. By standardizing on the GNU compiler, many if not all of the complicated conditional compilation sections of the MDSplus code can be removed, making th[1][2]e code much easier to read and support. The GNU library functions are for the most part standardized across most of the platforms so many of the configuration tests for operating system features can be eliminated as well. In addition, the code will be analyzed using the compiler’s ability to display a wide variety of warning messages indicating potential problems or extraneous house cleaning problems such as old variables being declared but no longer being used.

It is now even possible to compile and link windows applications using the GNU MingW-w64[8] compiler. The original port to Windows was done using Microsoft’s Visual Studio product. This also required numerous constructs in the source code to do different operations if the source was being compiled with Visual Studio compilers versus linux based compilers. In addition Visual Studio used very different compile and link options so the work done to use automake tools was not compatible with the Windows build of the software. The Windows compilers also did not abide by many of the compiler standards which limited the use of many advances in the programming languages. The MingW-w64 compiler is a cross compiler so all of the compile and linking of Windows libraries and applications can be performed on a linux system which can use all of the same build and development tools as all of the other linux based distributions. The libraries built with the MingW-w64 compiler are compatible with third party applications such as

LabView and Python which have often been a problem with the other tools such as cygwin which is better suited for developing standalone applications for Windows. There is currently still one issue with using MinGW-w64 compilers to build libraries for use with Visual Studio built applications. The exception handling in C++ applications differ between Visual Studio applications and C++ libraries built with MinGW-w64. Currently there is only one object oriented library in MDSplus for use with C++. To resolve this incompatibility, we have been including in the installation kits two versions of the MDSplus C++ library, one compatible with MingW-w64 built applications and one built with Visual Studio to be compatible with user applications built with Visual Studio. Everything but this one C++ Visual Studio dll is built on a linux based system.

Moving from the support of many different compilers to standardizing on GCC compilers has enabled us to greatly reduce the number of special conditional sections of the code resulting in a much cleaner and readable code base.

Standardized code indentation

During the ongoing development of MDSplus over the last 25 years there have been many different contributors to the code each with their own favorite coding styles. There was no dictated standard for indentation or commenting in the code. To improve the ability of all the developers to support all the code in MDSplus we are now using a common indentation style and using an utility called “indent”[9] which understands the coding language and can automatically indent the source code based on a selected indentation style. This has been used to indent all of the MDSplus code.

Standardized code documentation

Since most of the MDSplus code was developed, new tools have been designed which can parse the code and find documentation fragments with specialized markup language that can be used to produce nicely formatted documentation, either web based or printed. We are in the process of adding better code documentation and standardizing on the use of the Doxygen[10] tool to parse annotated sources to produce source code documentation.

Code testing

During the current automated build and release process several basic regression tests are performed on the MDSplus code to ensure that the basic functionality is working. These tests are very limited in scope and only test a small subset of the utilities and function provide by MDSplus. Part of the improvement project is to expand the scope of these tests to provide improved test coverage. The tests will be incorporated as an option to the standard build procedures of MDSplus if appropriate[3][4]. Using the *gcov*[5][6]*[11]* tool (“--coverage” option) of the GNU compiler it is possible to measure the percentage of the executable code is exercised by tests. Other test utilities such as valgrind[12] and the “-fsanitize” compiler option will be used to identify problems such as memory leaks and references to uninitialized memory which can produce intermittent problems. These tests should greatly improve the quality of the MDSplus releases and will ensure that major rewrites of existing code do not alter the existing

behavior of the code. We also hope to explore the possibility of adding automated performance testing to ensure that changes to the code do not cause performance problems.

Refactored code

Some of the major internal code of MDSplus could benefit by a complete rewrite of the code. One area of focus of the quality improvement project will be the reimplementation of the TDI (tree data interface) expression evaluator. This expression evaluator is a very important feature of MDSplus allowing data to be expressed as a mathematical expression of other measurements stored in the data files. It is also provides the possibility to reference information stored externally to the MDSplus data system such as accessing relational databases and data stored using other data acquisition software. MDSplus was designed using an object oriented approach and the expression evaluator is a good example of this. Unfortunately when it was developed most object oriented programming languages were in their infancy and tended to perform poorly. Today the c++ language is widely used and is very suited for developing the TDI expression evaluator. It is crucial that a comprehensive test suite is developed for TDI to ensure that there is no change in execution behavior after it is replaced by a more modern implementation in c++.

Improved build tools

The configure and build process for MDSplus is also a focus of the quality improvement project. The hand coded construction of the configure and Makefile's used in MDSplus will be replaced by the use of tools such as automake[13] and libtool[14] which provide standardized methods for describing how to build and install libraries and executables. The MDSplus automated build and release system has also been upgraded to use docker[15], a utility which enables you to construct pseudo virtual machines in which to build releases for different platforms. These docker virtual machines are much easier to construct and operate for this purpose than true virtual machine products.

Improved source code management

As part of the quality improvement project, the code management of MDSplus has been moved from cvs to git[16]. Git provides many features that makes it much easier to work in an environment with multiple developers and to handle release management. Much of the MDSplus release management scripting is much simpler after migrating to the use of git instead of cvs. Utilizing the central repository provided by github.com also simplifies the management of source code downloads.

Improved user documentation

A new MDSplus tutorial is being developed which will include working examples of applications and scripts which interact with MDSplus. These examples will be available as source downloads so users can build and run the examples on their local systems. As mentioned above new user documentation has been added to the MDSplus command utilities via a help command.

Conclusions

The MDSplus system has proved to be quite useful for the fusion community over the past quarter century and is used in a variety of ways at most fusion energy research sites worldwide. Beginning its existence as a package developed solely for the OpenVMS operating system, it has been ported to many different computing platforms. It has grown considerably in functionality over the years and most of the focus of the developers was placed on adding new features and providing support for more and more platforms while fixing bugs in the code when they were detected. More recently this focus has changed toward improving the quality of the code which makes up MDSplus utilizing more modern coding standards and taking advantages of available software utilities which did not even exist in the early days of development of MDSplus. This change of direction occurred mostly because of very useful suggestions and comments from the MDSplus user community. We added two new core developers, Keith Erickson of PPPL and Andrea Rigoni of Consorzio RFX, who are assisting in this work and bring a lot of exciting new expertise which the original developers were lacking.

The use of more readily available standard coding utilities for software project building, coding and documentation will enable us to throw out large portions of the original MDSplus code making it a much more robust product while reducing the difficulty of maintenance. Using better documentation tools for embedding comments in the code will not only improve the experience for users of the product but will also ensure the development and maintenance of MDSplus can continue into the unforeseeable future. At this point it is difficult to predict the scope of this project. As we go back and investigate much of the MDSplus code base we discover more areas that would benefit from refactoring using more modern techniques and coding standards. It is anticipated that MDSplus would greatly benefit by focusing at least two to three man years on this quality improvement effort to ensure that MDSplus will continue to be a useful tool for the fusion research community for many years to come.

Acknowledgements

The success of MDSplus stems largely from the suggestions and comments from the user community and this project is just another example of this collaboration of the developers with the users to provide a useful tool for the community.

A portion of this work is being performed under US Department of Energy Award Number: DE-SC0012470.

[1] MDSplus, http://www.mdsplus.org
[2] Stillerman, J.A., Fredian, T.W., Klare, K.A., Manduchi, G., "MDSplus data acquisition system", Review of Scientific Instruments, January 1997 68(1) pp 939-942.
[3] OpenVMS, http://en.wikipedia.org/wiki/OpenVMS
[4] DCL, Digital Control Language, http://en.wikipedia.org/wiki/DIGITAL_Command_Language
[5] Flex, http://flex.sourceforge.net/

[6] Bison, http://www.gnu.org/software/bison/
[7] XML, http://www.w3.org/XML/
[8] MinGW-w64, http://mingw-w64.org/doku.php
[9] Indent, http://www.gnu.org/software/indent/manual/indent.html
[10] Doxygen, http://www.stack.nl/~dimitri/doxygen/
[11] Gcov, https://gcc.gnu.org/onlinedocs/gcc/Gcov.html
[12] Valgrind, http://valgrind.org/
[13] Automake, http://www.gnu.org/software/automake/
[14] Libtool, http://www.gnu.org/software/libtool/
[15] Docker, http://www.docker.com/
[16] Git, http://git-scm.com/