
MTools, an Eclipse-Based IDE for VistA

Editor, Debugger, and IDE Tools for VistA Development

Release 1.0

The Department of Veterans Affairs (VA)

June 27, 2013

Abstract

Eclipse is widely used as an integrated development environment (IDE) for many projects that leverage mainstream programming languages such as Java and Python. However, its usage for Veterans Health Information Systems and Technology Architecture (VistA) development has been limited. This document describes the work performed by the Open Source Electronic Health Record (EHR) Services Project Team, on behalf of the Department of Veterans Affairs (VA), on an IDE for VistA development that is built on the Eclipse platform. This work builds on the existing VA-developed MTools and adds support for hierarchical directory structures, a rewrite of client portion of the MDebugger to follow general Eclipse guidelines with additional features such as code tracing, a number of analysis and refactoring tools, and various fixes.

Contents

1	Introduction	2
2	MTools Overview	4
2.1	MConnector	4
2.2	MEditor	4
2.3	MDebug	5
2.4	M Validations, Code Analysis, and Refactoring Tools	6
3	Supporting Documentation	7
4	Repositories	7
5	Issues and Future Work	7
6	Conclusions	8

1 Introduction

The core of VistA is developed in the MUMPS programming language, and deployed to either InterSystems Caché or GT.M systems. MUMPS is an interpreted language, therefore no compilation tools are needed. The process for developing VistA includes editing text files containing syntactically correct MUMPS code. Each file is a MUMPS “routine” that includes entry point tags. The files, or routines, are deployed to a server such as [Caché](#) or [GT.M](#). Debugging is typically done by inspecting the output of the actual system using output channels such as standard output (a console) or inspecting the values of the database during and after the code runs.

Currently there are limited IDEs to aid VistA development. There is at least one proprietary tool which provides an IDE with a color syntax editor and a debugger as well as saving and loading routines from a server. However, because that tool is proprietary, enhancing it as a product directly is not possible. There are also open source MUMPS-specific editors and various code analysis tools typically written in MUMPS; however, these are all stand alone.

[Eclipse](#) is widely used as an IDE for many projects that leverage mainstream programming languages such as Java and Python. It is an extendible open source platform that provides everything necessary in an IDE such as editors, debuggers, version control systems, search facilities, build, and refactoring tools. The overarching goal of this work is to promote the Eclipse platform for VistA development and testing by providing a viable open source MUMPS IDE. By working against this objective, the following benefits have been discovered:

1. Direct enhancements and major features to an open source MUMPS IDE product can be added quickly with direct initiative and feedback from the open source community and VA, contributing also to broad user acceptance. This means that specific tools and features which can strategically benefit a series of projects can be added to this IDE product. For example, our team has previously delivered to the Open Source Electronic Health Record Agent (OSEHRA) a [Roll-and-Scroll Recorder](#) (RASR) to develop automated tests for VistA on the Eclipse platform.
2. The Eclipse platform already contains other tools that are vital to software development. As an example EGit, the Eclipse plug-in for the git versioning system, is readily available for the Eclipse platform and git is the version control system of choice for OSEHRA. The Eclipse platform includes sophisticated search mechanisms and comparison tools for files and a well-established project structure that allows for the ability to work on multiple projects at the same time. With built-in support for software development, an Eclipse based MUMPS IDE is expected to increase programmer productivity.
3. Since IDEs on the Eclipse platform are familiar to most mainstream programmers, a MUMPS IDE on the same platform would decrease the learning curve for mainstream developers that want to become skilled at and develop for VistA. Since Eclipse has such a high acceptance rate and is a widely-adopted industry standard, it proves to be a viable choice as an IDE. The Eclipse IDE itself is language agnostic and implanting features specific to another language is a very standard concept with regards to Eclipse plug-in development. This will allow users get to utilize the same Eclipse they are familiar with while having MUMPS specific support built in.

VA developer Joel L. Ivey previously created [MTools](#) for the Eclipse platform, consisting of the Eclipse plug-ins MEditor and MDebugger, as well as a number of server communication, routine load/save, and global utilities. MEditor is a relatively stable MUMPS editor with color syntax and “Load From”/“Save To” server features. MDebugger is a work in progress MUMPS debugger which features a custom interface. Our team’s work provides improvements on these two tools mainly for the following issues:

Identified Issue	Improvement Made
The original MEditor assumes a flat directory structure and does not play well with hierarchical repositories such as OSEHRA’s VistA-Freedom of Information Act (FOIA) instance.	It is now possible to use MEditor with hierarchical repositories: files in any repository directory can be saved to the server and files loaded from the server are linked to the correct file in the client hierarchical repository.
The look and feel of the original debug plug-in, MDebugger, deviates from other mainstream Eclipse platform debuggers in such key components as code tracing, breakpoints, variable watching, and debug configuration and launch.	The look-and-feel and client functionality is now similar to other debuggers in Eclipse platform; code tracing is available, debug configuration and launch are in the same menu items as other debugger and standard Breakpoint and Variables views are used.
The back end for MDebugger, which is essentially a MUMPS interpreter written in MUMPS, has a number of bugs and missing implementations for various MUMPS language constructs.	<p>Fixes and improvements include:</p> <ul style="list-style-type: none"> • Support added for indirection in DO commands • Support added for routines that do not end with QUIT on the last line • Support added for quoted extrinsic functions • Fixed the issue of commented lines being executed • Support added for expressions in WRITE commands • Support added for extrinsic functions to other routines • Support added for expressions in FOR loops • Support for naked globals inside parameter expressions <p>The full list can be found in M-Tools-Project repository (see repositories section in this document) XTDEBUG project. Use Team/Show in History in Eclipse.</p>

In addition to these improvements, MTools also adds MUMPS validation and refactoring tools implemented in Java that are available from Eclipse menus. These tools use the same Java code base as the [M Routine Analyzer](#) previously submitted by the Open Source EHR Services Project Team as an OSEHRA Technical Journal.

2 MTools Overview

MTools is installed as a single Eclipse “Feature”, which is a set of Eclipse plug-ins.

2.1 MConnector

MConnector is a plug-in which is used by other plug-ins to communicate to an M Server. This plug-in existed in VA versions of MEditor and MDebugger and remains virtually unchanged for this version of MTools.

2.2 MEditor

The primary enhancements made were to support hierarchical directories and, in particular, OSEHRA’s VistA-FOIA repository structure. In the original MEditor version, routines were automatically loaded and all the routines were expected to be in the root folder or a particular subfolder of the Eclipse project. Now files can reside at any directory and routine load and save logic was re-implemented to support the same. Additional features included with this work are marked by an asterisk (*).

The MEditor plug-in provides the following features:

- Color syntax editing
- Tag outline view
 - Ability to jump to a desired tag from this view
 - Ability to invoke code analysis tools on a desired tag*
- Routine synchronization via loading and saving
 - Automatic routine visual comparison, displaying the color coded differences when routines are out of sync*
 - Bulk routine import by name filter
 - Ability to save multiple routines
 - XINDEX analysis on saved routines

- Ability to save files directly from the Project view which lists all project files in a tree*
- Support for hierarchical project structures*
 - Specialized support for VistA-FOIA package structure with regards to loading routines*
- Server listing of routines and globals
- Ability to invoke code analysis and refactoring tools on the routine from the Context menu*

2.3 MDebug

MDebug consists of a core plug-in and a user interface (UI) plug-in. These two plug-ins replace the original MDebugger and use the native platform Eclipse debugger's UI and background process. This gives it the same look and feel as the popular Java and C++ debuggers.

MDebug provides the following features:

- Visuals via a graphical user interface (GUI) debugging
- Persisted launch configurations
- Breakpoints
 - Visual breakpoint markers added to a line within a routine file
 - Breakpoints added via a tag name
 - Watchpoints on variable changes
 - All breakpoints and watchpoints are persisted
- Real-time stack tracing
 - Highlights the code line currently awaiting execution
- Variable inspection
 - Ability to see variables created during code execution
 - Ability to see all variables defined, able to filter results
- Interactive console
 - Displays WRITE commands, and collects user input for READ commands

MDebug is a complete rewrite of the original MDebugger and asynchronously runs the debug logic in the background while providing a responsive UI. It adds new features such as highlighting the line containing the next executing command. The original MDebugger did not provide a link between the code in the editor and the debugger, but now MDebug adds breakpoints to MEditor by linking them on the left vertical bar to their respective line numbers. (Note: Breakpoints can also be added and removed this way.)

Console improvements include using the standard Console view in eclipse and the ability for this view and its specific MUMPS console to come to the foreground of Eclipse when new input comes in.

The backend of MDebug is the same set of MUMPS routines (XTDEBUG*) that are used by the original MDebugger. XTDEBUG routines essentially implement a MUMPS interpreter in MUMPS on the server. However, not all constructs of the MUMPS language are implemented and there exists a number of bugs. As part of this work we addressed numerous bugs and were able to use the debugger on our team's "silent" refactored code that was submitted to OSEHRA for other projects. However, we were not able to use the debugger on more complex MUMPS code found elsewhere (i.e. Fileman). It has been recognized that more fixes are necessary to be able to use the debugger on complex Mumps code, but the as-is debugger provides value as an improvement from the original MDebugger.

2.4 M Validations, Code Analysis, and Refactoring Tools

MTools also provide a set of validation, analysis, and refactoring tools available from the Context menus on the Editor, Outline, and Project Explorer views. The available tools are Expand Keywords, Report Syntax Errors, Report Assumed Variables, Report Quit types, and Report Occurrences. All tools are client tools and only operate on the client files. Reference the tools listed below for more detail:

- **Expand Keywords:** This tool changes mnemonic forms of MUMPS commands and intrinsic functions to their full form (e.g. N -> NEW) in the selected Routine.
- **Report Syntax Errors:** This reports all MUMPS syntax errors found in the selected Routines. This is similar to XINDEX and is provided as an alternative on the client. The report it generates is clickable for easy access to the errors.
- **Report Assumed Variables:** Generates a clickable list of all variables that are used without being "newed". It is similar to what XINDEX provides, but also recursively analyzes all the tags that the selected entry tags call.
- **Report Quit Types:** Reports invalid calls to extrinsic functions, invalid set commands to tags that do not return a value, and inconsistent quit commands (return values vs. not return values).
- **Report Occurrences:** This report gives a clickable list of occurrence of write commands, read commands, indirections, naked globals, intrinsic text functions, execute statements, goto commands, and exclusive kills.

All MUMPS analysis, validation, and refactoring tools use a MUMPS parser that is implemented in Java. The parser also provides an "almost" complete MUMPS parse tree and a visitor which makes development of additional tools relatively easy.

3 Supporting Documentation

A detailed installation and usage guide is included as a separate document labeled “MTools Installation and Usage.docx” in the Technical Journal submission.

4 Repositories

The development repositories are <https://github.com/kthlnkeating/M-Tools-Project> for all Eclipse plug-ins and <https://github.com/kthlnkeating/MParseAnalyze> for validation, code analysis, and refactoring tools. The latter is included in the former as an external jar.

5 Issues and Future Work

To date this version of MTools has only been tested by our team and is recognized as a work in progress based on the feature improvements and extensions previously mentioned. We are starting to use this tool in our development and testing process for the [Open Source EHR Services refactoring work](#) and expect to add additional tools and bug fixes to that effort as necessary. An issue tracker is available to open source community [here](#) under component Eclipse Support and includes known bugs.

Future work on the plug-ins is expected to focus on the following areas:

1. MDebug uses the MUMPS interpreter implemented in XTDEBUG routines as the back end. This interpreter still has unimplemented features of MUMPS and requires a significant amount of additional testing to identify all remaining bugs. It also interacts with VistALink and Remote Procedure Call (RPC) Broker routines requiring special handling to keep MUMPS local variables used in those packages from affecting the debugging process. However, leakages still remain and it may be advantageous to use ZBreak features of Caché and GT.M instead. The possibility to use ZBreak functionality will first need to be investigated and, if viable, the switch will need to be made.
2. Unfortunately due to back end issues, our team was not able to successfully run programmer entry point (D ^XUP) for the Roll-and-Scroll interface from the debugger. If backend issues are resolved, our team will work to run the Roll-and-Scroll interface and simultaneously use the debugger as user interaction occurs.
3. The addition of a MUMPS-specific project type would prove advantageous, especially to help keep settings project-specific and provide more powerful server synchronization features.
4. Further refactoring tools such as code beautification, variable, tag, and routine renaming are desired to replace current manual processes to save programmers time.

6 Conclusions

In this document the Open Source EHR Services Project Team has presented steps taken towards developing a fully functioning MUMPS IDE for VistA development on the Eclipse platform. Additional steps have been identified, but we believe this work paves the way for the acceptance of Eclipse as a development and testing environment in the open source and VistA communities.