
MTools, an Eclipse-Based IDE for VistA

Editor, Debugger, and IDE Tools for VistA Development

Release 1.1.7

The Department of Veterans Affairs

September 23, 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; an alternative debugger based on debug commands that are provided by MUMPS implementations; 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	7
3	Supporting Documentation	7
4	Repositories	7
5	Issues and Future Work	8
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 InterSystems [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) via an [OSEHRA Technical Journal submission](#) 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 are now similar to other debuggers on the Eclipse platform; code tracing is available, debug configuration and launch are in the same menu items as other debuggers, 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 was 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 (please reference the repositories section within this document) XTDEBUG project. Use Team/Show in History in Eclipse.</p>

Identified Issue	Improvement Made
The custom-written back end for MDebugger still has multiple issues, especially for user interface (UI) heavy MUMPS programs. In particular, currently it is not possible to debug Roll-and-Scroll Interface programmer entry point ^XUP which limits its benefit for VistA development.	An alternative debugger has been implemented based on ZBREAK command provided by InterSystems Caché or GT.M. This new debugger essentially serves as a UI for ZBREAK and related commands; it is now possible to debug ^XUP.

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. A core plug-in exists for common functionality and expressions but otherwise the plug-ins can be grouped according to their functionality: MConnection, MEditor, MDebug, and MUMPS validation, refactoring, and code analysis.

2.1 MConnector

MConnector is a single plug-in which is used by other plug-ins to communicate to a MUMPS Server. This plug-in existed in VA versions of MEditor and MDebugger. It was implemented in a server centric paradigm and projects were added later in the design. This version of MTools has shifted towards a more project centric paradigm.

2.2 MEditor

MEditor now consists of multiple plug-ins which are responsible for various aspects of the functionality: MUMPS file editing, saving to and loading from server, and wizards. 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
 - Ability to use Eclipse Local History for comparisons*
 - Bulk routine import by namespace filter
 - Ability to save multiple routines
 - XINDEX analysis on saved routines
 - Ability to save files directly from the Project Explorer 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*
- Wizard to create a new VistA file*

2.3 MDebug

MDebug consists of a core plug-in and a 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 can be considered as three distinct debuggers: the first "Generic" version is similar to the original MDebugger where the back end is a custom M Interpreter implemented in MUMPS; the other debuggers are respectively UIs for ZBREAK functionality in GT.M and InterSystems Caché.

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 (not available for GT.M version)
 - 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 (not needed in GT.M or InterSystems Caché versions)
- Interactive console
 - Displays WRITE commands and collects user input for READ commands (problematic in Generic version)

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.)

For the Generic version, 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 back end of Generic 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 are a number of bugs that exist. As part of our efforts 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 sub-projects as part of the main VA contract. 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.

The GT.M and InterSystems Caché MDebug versions provide the Eclipse versions of the GT.M and InterSystems Caché terminal interfaces. These MDebug versions use commands native to GT.M and InterSystems Caché and also work for the Roll-and-Scroll Interface programmer entry point ^XUP, one of the main objectives of this project.

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 repository is located [here](#) for all Eclipse plug-ins.

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 have started 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. The GT.M and InterSystems Caché versions of MDebug are notably superior to the Generic version, and thus the Generic version is expected to be phased out based on community feedback and input. However, users should note that these versions have only been completed recently and additional testing is required.
2. For a debugging session the client version of MUMPS files are shown in Eclipse while code tracing while the debugger uses the server version; it is currently at the user’s discretion to make sure both versions are the same. This has identified the need of an improved mechanism to verify that the server and client versions of the files are identical; the most ideal option would be a “build” step where files that are changed on the client are saved to the server.
3. MEditor does not partition the MUMPS files according to its tags and other language elements. While this feature is available in most other Eclipse editors for additional languages, and Eclipse provides a framework, a necessity has been identified that this be implemented for MUMPS as well.
4. Context menus for DO and GOTO commands and extrinsic functions should provide the ability to open target tags directly.
5. The addition of a MUMPS-specific project type would prove advantageous, especially to create project-specific settings and provide more powerful server synchronization features.
6. Further refactoring tools such as code beautification, variable, tag, and routine renaming are desired to replace current manual processes and to improve programmer efficiency.

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.