MDev Eclipse Plugin for M Development

*Release 1.00*

PricewaterhouseCoopers, LLC (PwC) on behalf of the

Department of Veterans Affairs (VA)

April 3, 2013

**Abstract**

As part of the Open Source EHR Services project funded by the VA, we have worked on a Eclipse plug-in for M development.

# Contents

**1 Introduction 1**

**2 MDev Project Type 1**

# Introduction

Eclipse is used as the primary Integrated Development Environment in many development projects in software industry. The goal of this project is to deliver on an open source Eclipse plug-in that can become an IDE for M development with emphasis on VistA.

There is already an plug-in developed by VA. This project builds on that. However we made significant changes and in some cases complete rewrites. In the rest of the document the older plug-in will be referred as M-Editor and M-Debugger and our work will refer as MDev and MDev-Debugger.

# MDev Project Type

This work adds a new project type MDev. Most of discussion in this document assumes that you work on a project of type MDev. There is also an associated perspective MDev which adds the menu items and buttons discussed here.

MDev projects keep an index of all the files and also a flag of synchronization between the server. They also have project wide settings for M server information and various validations. Note that each MDev project owns its server information. Some validations create temporary intermediate files to improve the performance of validations.

MDev projects works on a different paradigm than the M-Editor. M-Editor is server centric rather than Eclipse project centric. In M-Editor the server location are shared by all the projects. Typically a project named “mcode” is automatically created and all routines are added to this project at the root directory. You can specify the project name to work with the servers but again this is workspace specific. In addition M-Editor assumes all the routines are stored in a flat directory structure. You can specify multiple servers for “mcode” or other named projects and saving will be done for all servers when you save a file.

MDev-Editor is project based and each project has one server associated with it. You can work offline without a server on your M code if you wish. You can also change the server setting anytime you want. You can have as many files in your project as you want with any directory structure. You can run a number of validation tools right from Eclipse without any need for a server. This flexibility comes with the price of keeping track of server/client synchronization. As discussed before this is one of the main reason to have a MDev project type. A number of tools are provided to establish and verify the scynchronization. These are discussed elsewhere in the document.

MDev projects can be created either from Import Project from Git repository perspective or directly from New Project. MDev project themselves will not do any version managements (except keeping tract of server routines that are changed from client) so it is recommended that you create a git repository from creating the MDev project and import.

# Loading From Server

In your MDev project you can load files from the server using “Load From Server” menu item or button. You can either specify a specific routine name or a set of routines by using wild card characters ‘\*’ and ‘?’. These two cases are handled somehow differently so will be described separately.

When you load a single routine from the server and if the routine already exists in the project then the existing file will be updated. It is assumed that versioning system such as egit will keep track of the changes so no effort is given to keep track the original version. If the file is not found then user will be asked to choose a (possibly new) directory in the project to save the file.

When you load multiple files using wildcards user will be requested to select a (possibly new) directory to save the files. First all the routine names that match is loaded from the server. If there are already files in the project with the same in a directory other than the selected one user will be given an error and loading will not continue. Otherwise all the routines will be loaded to the selected directory.

# Saving To Server

If server specification is done for the project then saving to server work seamlessly in edit and save workflow. If server specification is not done or work is being done offline then no attempt will be made to save the file. In this case the version of the file that is last saved to server is kept track. To work these cases a “Save To Server” menu item or button is available in MDev project where user can use multiple files and/or directories and save all th m files to the server. For all saving to the server the replaced server routines are internally saved in MDev project so that the last server version will be available to recover. A menu item or button “Restore Server Routine” will be available.

# Synchronization With Server

At this point it is outside the scope of this work to fully standup a server from an existing repository. There are instructions to do that for example for OSEHRA VistA-FOIA. MDev projects, when created from an exisiting repository like VistA-FOIA assumes that the M files in the repository are synchronized with the server. Once the MDev project is created user can use “Load From Server” and “Save to Server” to further achive the synchronization manually.

To that end however an additional tool “Report Server Synchronization” is provided on the menu. This tool compare all the client and server version of the selected routines and reports the differences. Based on the report user can decide to synchronize whatever routines he chooses.

Note that synchronization is very important for validation of the routines that fanouts to other routines and debugging. Handling of those cases are described in respective sections.

# Debugging

MDev projects improves the Eclipse user interface to be more inline with with debuggers in other languages such as Java. Additional features and improvements shall be made too.

# Server Synchronization

# When starting and stopping a debug target

When the debugger launches, it checks all the routines that the break points are on to make sure that the server and client versions are the same. Because in the eclipse model, metadata is saved about line locations on the clients machine. This means that if a line number changes, the client side may keep the old line number while the server (after the routine is saved) will receive no content for the routine, possibly moving the lines up or down.

# When entering stepping mode

When the users steps into a different routine (fanout) the server and client versions of the fanout routine is compared and if they are different user is asked to choose a version. Depending on the what user selects either server or client updated. In the former the action before debugger continues is identical to “Save To Server” menu item and in the latter it is identical to to “Load from Server” menu item. After the update debugger continues as usual.

# Stepping Improvements

Various issues with stepping shall be resolved. The backend debugger must be handled correctly as to know when to suspend the debugging, thereby entering stepping mode. Prior to this the debugger was entering step mode whenever a response was received from the backend. Other improvements will include implementing ‘Step Over’ and ‘Step Return’, as is common in other mainstream debuggers. Currently these are not implemented on the backend. Other imrpovements to be made are to not enter stepping mode immediately but to let the code run until it enters a breakpoint and to readily displace to the user that the program has in fact terminated in the UI.

# Stack Tracing

The previous MDebugger does show a stack trace, however it is a simple text list and the display could be made to be easier to read and therefore more productive. The new MDev stack tracing will use the default eclipse ui. And like modern debuggers, it shall also provide contextual information for each stack such as variables created and the current code location from where this stack was last running (what line was it last pointing to, for parent stacks this will be either a fanout or block).

# Displaying Variables

Variables shall be displayed in the default Variables view. This view operates in the context of the currently selected stack frame. In most languages variables are scopped to a stack, however in MUMPS it is true that one a variable is created it is accescible anywhere. We shall go with what Cache Studio has implemented and that is to show variables that were created on a particule stack. This is contextually related and therefore intuitive and also very informative to show to the user.

# Varibles of a stack frame

When a user clicks on a stack frame, all variables that were created on that stack frame, and any parent stack frames shall be shown. So for the current or top stack, this will be all variables that were created from the bottom stack and up. If the user clicks on say, the second bottom most stack, it will show any variables that were created on the bottom stack and second most bottom stack (the one selected in this example). The bottom most stack will not include variables that were created prior to the running of this debug session.

# All Variables

All variables shall be displayed in a new view specifically for our debugger. This will show all the current variables on the server. It shall be filtered with a text box. This same behavior exists in the previous MDebugger but the UI needs improving.

# Breakpoint Improvements

The previous Mdebugger allowed setting arbitrary tag locations to break on, as well as variables to break on if their value is changed. Our new debugger will enhance the UI display of the breakpoints view by using the standard breakpoints debug view from Eclipse. It also consolidates both breakpoints and watchpoints into a singleview, which is more productive and keeps more information on on 1 sceeen. Breakpoints shall also be set from an editor by double clicking on the vertical bar to the left. This is a very helpful visual cue and a modern feature of all mainstream IDEs.

# Interactive console

The previous MDebugger had an interactive console that would display output (ie: menus) and send back user input (ie: selecting a menu option) to the server. However the backend appears to be failing when it encounters a READ command. This will be fixed in the newer version. Also the newer version shall also provide support for terminal emulation so that the menus are actually readable. This is something that cache studio does not offer either.

# Userinterface Improvements

# Aesthetic improvements

The prior debugger was all custom made User interfaces exposed as custom Eclipse views. Our new debugger will integrate itself into the default Eclipse Debug views. This is a common debugger many developers are experienced in as well as a proven graphical interface for debugging many languages.

# Performance improvements

MDev Debugger shall have a very perfomant user interface. When buttons are clicked on they shall not cause any other part of the userinterface or backend to halt and wait for a response. This means that when a user clicks on step, it shall not freeze the UI and lock it until the backend finishes. All modern graphical UIs will leverage some combination of multi-threading and asynchronous event based processing to prevent graphical interfaces from locking up. The original debugger had some support for this but only for the interactive console.

# Validation

MDev projects provides a number of validation option on M files. These are listed in the subsection. There are two types of validation. One is server based XINDEX runs that were also available from M-Editor. The second type validations are client based and use a Java M parser. Both types can be specified to be run when a file is saved. Server validations can only be run if server specification is made.

Some of client based validations also need the fanout routines. If the fanout routine is in the project it self it is available. However sometimes the routine might not be in the project but available on the server if the server specified. For these case s the routines are read from the server during the validation process based on a setting.