

Department of Veterans Affairs

Open Source Electronic Health Record Services

MTools Integrated Development Environment System Design Document



September 2013

Services and solutions provided by the Open Source EHR Services Project Team are supported by, and on behalf of, the Department of Veterans Affairs (VA), Office of Information and Technology (OIT) via contract VA118-12-C-0056.

Please reference the OSEHRA Technical Journal posting for more information: <http://hdl.handle.net/10909/85>.

Table of Contents

1.	Introduction	3
1.1.	Reference Documentation	3
2.	Enhancing Original MTools Plug-ins	3
2.1.	MEditor	4
2.2.	MDebug	4
3.	Implementation Specifications for MTools Features.....	5
3.1.	MEditor	5
3.2.	MDebug	5
3.2.1.	Concurrent Processing of the Eclipse UI Concerns and XTDEBUG Back End Concerns	5
3.2.2.	Case Example: How to Work with Separate User Interface and Core Plug-ins	6
4.	MTools Class Diagrams	6
4.1.	MEditor	6
4.2.	Loading Routines	7
4.3.	Saving Routines	8
4.4.	MDebug Generic Implementation	9
4.5.	MDebug GT.M Secure Shell and InterSystems Caché Telnet Implementation	10
5.	References	12
5.1.	Acronyms and Definitions	12
5.2.	Software Licenses	13
5.2.1.	Software Under License	13
5.2.2.	License Locations	13

1. Introduction

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 (VistA) development has been limited. This document describes the work performed by the Open Source Electronic Health Record (her) 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.

1.1. Reference Documentation

The following artifacts may be used in parallel with this System Design Document (SDD):

- MTools, an Eclipse-Based IDE for VistA (OSEHRA Technical Journal [OTJ] overview)
- MTools Installation and Usage Guide

2. Enhancing Original MTools Plug-ins

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.	Fixes and improvements include: <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

Identified Issue	Improvement Made
	<ul style="list-style-type: none"> • 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>
<p>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.</p>	<p>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.</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 OTJ.

2.1. MEditor

MEditor's routine saving and loading to and from the server was re-implemented from scratch. This also involves the logic for backup, comparing routines, hierarchal directory support, and the dialogs that occur when saving or loading. These were refitted completely because the prior code was too difficult to maintain – it had grown and become patched over many times, obscuring any clarity to its design. In addition, the prior code was using the older Eclipse Actions infrastructure which is now deprecated; new code now uses Eclipse Command infrastructure. In addition, third party diff tools were also removed because the new structure allows use of Eclipse's "Local History" diff features. The commit differences on the [git repository](#) clearly show the new code replacing the old code in detail.

2.2. MDebug

The original debug plug-in (MDebugger) was migrated to two new plug-ins; only a very small portion of the original code remains in the new plug-ins. Many new features were added and all existing features were brought over. The main difference is that MDebugger was written as Eclipse Actions, which would then update and change Eclipse Views and their Standard Widget Toolkit (SWT) components.

MDebugger's graphical user interface (GUI) was not as finely presented as other debuggers, and a debugger's ability to present highly-readable details is critical. Thus, the decision was made to migrate to the Eclipse Debug Model, the variation in which all other languages that use Eclipse implement debuggers. This model provides a widely adopted and familiar debug GUI and behavior, as well as fast response and detailed information. Additionally, making this migration was the only viable way to implement the newer desired debug features, specifically adding breakpoints to a line in MEditor.

3. Implementation Specifications for MTools Features

3.1. MEditor

Eclipse as a GUI-based IDE works from a single main, or user interface (UI), thread; it also manages a thread pool to handle background processing. (Note: Only the main, or UI thread, can create or update SWT components.) Generally, all other processing should occur in background jobs – otherwise the main thread will have to wait for the non-SWT processing to finish before the user can do anything to the Eclipse workbench window. Although background jobs cannot directly update SWT components, they can schedule a job for the UI thread to do any SWT widget creation or updates.

Given Eclipse's multithreading workflow, changes were made to MEditor to put many of the actions and other processes that it creates into background jobs. This enhances the UI response when editing text or clicking any of the MEditor actions. As for MDebug, it does not rely on these contrived Eclipse Actions, but instead uses the Eclipse Debug Platform to ensure all actions are put into background jobs. Therefore clicking an action, such as "step over" or "resume", does not cause Eclipse to become unresponsive.

3.2. MDebug

3.2.1. Concurrent Processing of the Eclipse UI Concerns and XTDEBUG Back End Concerns

MDebug was created by following an [article](#) on how to create a custom Eclipse Debugger for any language. Please note that this article was written in 2004 and is partially out of date with regards to asynchronous processing; specifically, it mentions that the step feature in Eclipse is invoked from the UI thread, which is incorrect with the version of Eclipse our team currently uses. In our team's version, stepping is invoked as a background job and will not cause Eclipse's workbench window to hang, even if the backend debug system is synchronous.

In our case, the XTDEBUG Remote Procedure Call (RPC) calls are synchronous; they send a single request and wait for its response to come back. All of this is processed in an asynchronous job on Eclipse so the UI will not have any hanging problems. However, it is also worth noting that although multiple jobs can run concurrently, access to the RPC must not be concurrent because the MUMPS background process can only handle a single request at a time. The Java keyword "synchronize" is used on the XTDEBUGHandler class to make sure that multiple Eclipse jobs do not call this RPC while it is processing a current job.

3.2.2. Case Example: How to Work with Separate User Interface and Core Plug-ins

A custom console for handling READ and WRITE commands was added to MDebug. This console relies on the Eclipses Console View and Console Manager components and not the Debug Platform. As a result, the plumbing to get this rendering and working on the screen is not provided by default. Therefore, implementing it required less lightweight delegate-based classes.

This custom console has been implemented in the UI plug-in mostly with some UI agnostic listener classes existing in the core plug-in. The custom console works by registering a Launch Configuration listener to the MDebug UI plug-in; this listener will react if any new launch configurations are launched by the user. Custom listeners for the custom console are added to both the debug target and the console as the debug target must listen for when the console has finished collecting input and determine what that input is. The console needs to then know when it is ready to receive input and if any output is to be written. These listeners are registered in a generalized way as opposed to having the classes directly invoke each other by the specific Java class name; this is due to the fact that it is not possible for the core plug-in to see the UI console class directly as its specific Java type. Instead, the console implements a core listener and is registered to the debug target, a class from the core plug-in. This type of event-based processing is common for design paradigms which completely separate the UI and core or model concerns into separate projects.

4. MTools Class Diagrams

This section gives brief class diagrams for the most important functionalities of MTools.

4.1. MEditor

MEditor extends the Eclipse TextEditor class and most of the functionality is inherited. From an editor point of view, the main contribution of MEditor is MUMPS colored syntax support which follows the general Eclipse Editor design.

The Outline view is implemented primarily in MContentOutlinePage class and displays Entry tags of the routine. The other primary feature is saving routines to a MUMPS Server automatically and this shares code with Save Routine commands through SaveRoutineEngine class. Context menu support is implemented through Eclipse command infrastructure using extension points and no code exists within MEditor code for the same.

Figure 1 below illustrates implementation of Outline view and automatic saving of routines:

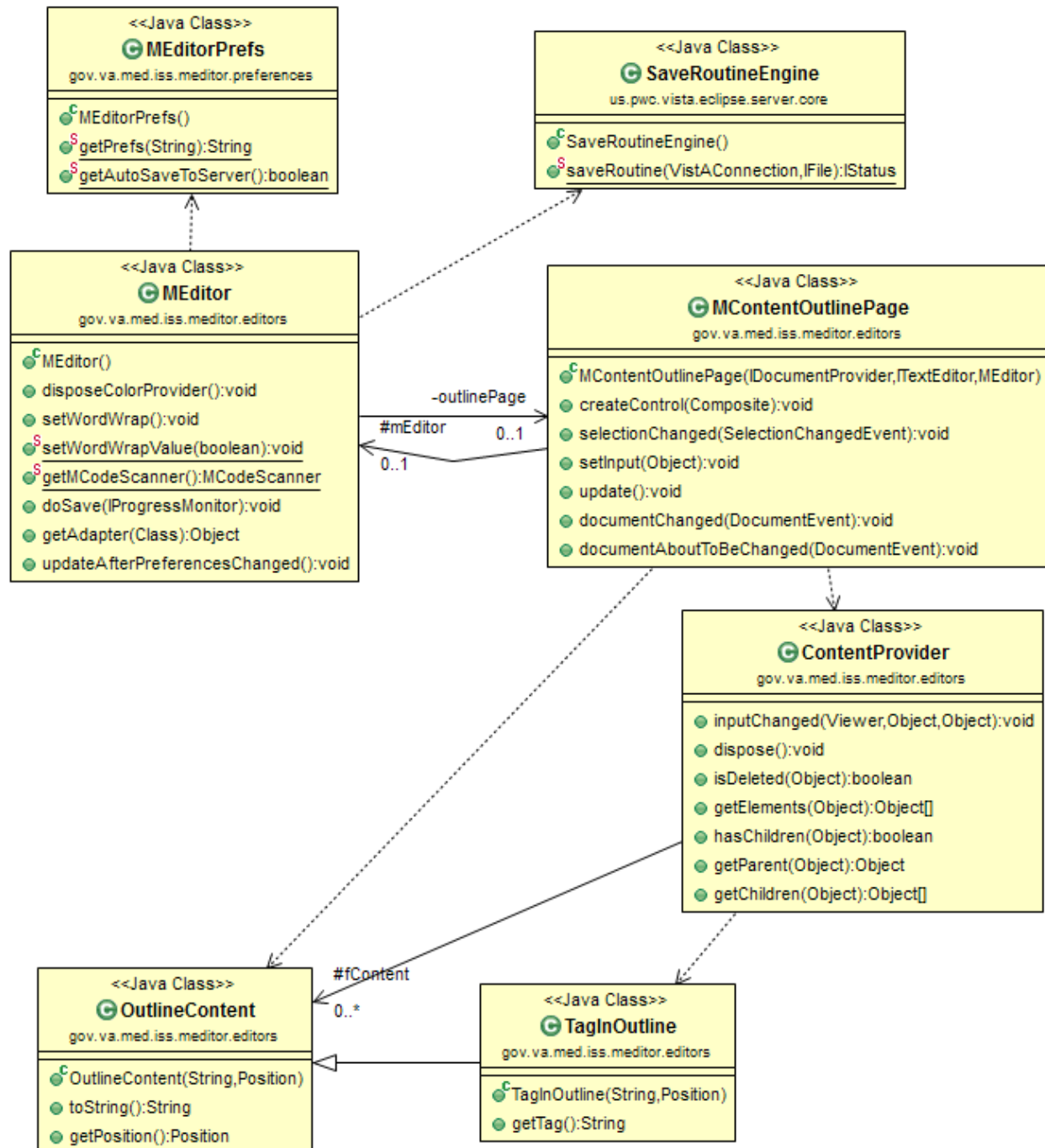


Figure 1: MEditor Outline View and Automatic Routine Save Classes

4.2. Loading Routines

Load Routine functionality is implemented using Eclipse Command infrastructure. Top classes all extend Eclipse Abstract handler class. Heavy lifting is done by the `MServerRoutine` class which represents a routine on a MUMPS server and responsible to load routines from server

and create the backup files. All server communication is handled by VistaConnection class. Figure 2 below illustrates the implementation:

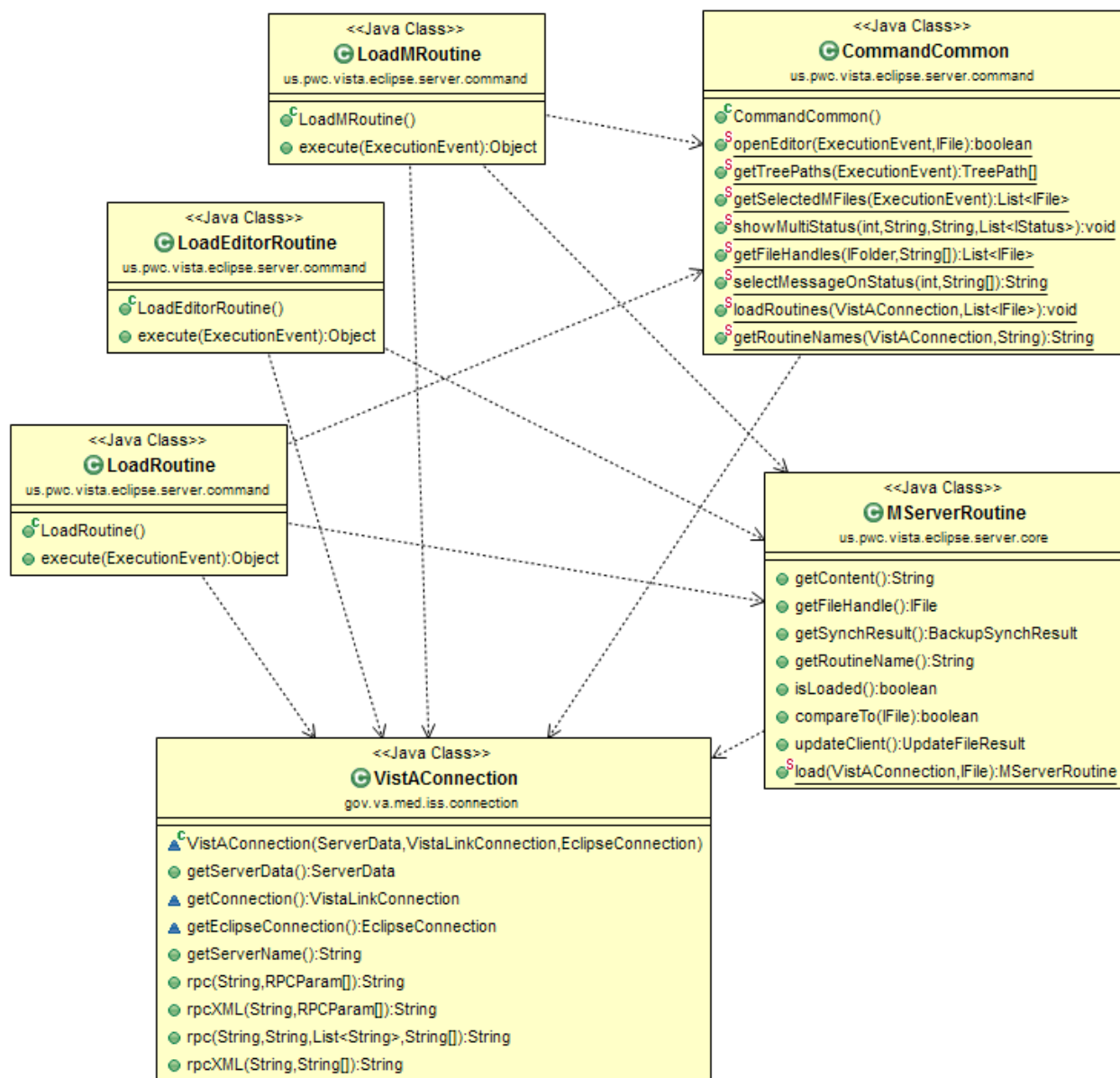


Figure 2: Classes for Loading Routines

4.3. Saving Routines

Save routine functionality is implemented using Eclipse Command infrastructure. Top classes all extend Eclipse Abstract handler class. MServerRoutine is also used since before saving a routine a comparison is done with the existing routine on the server. All server communication is handled by VistaConnection class. Figure 3 below illustrates the implementation:

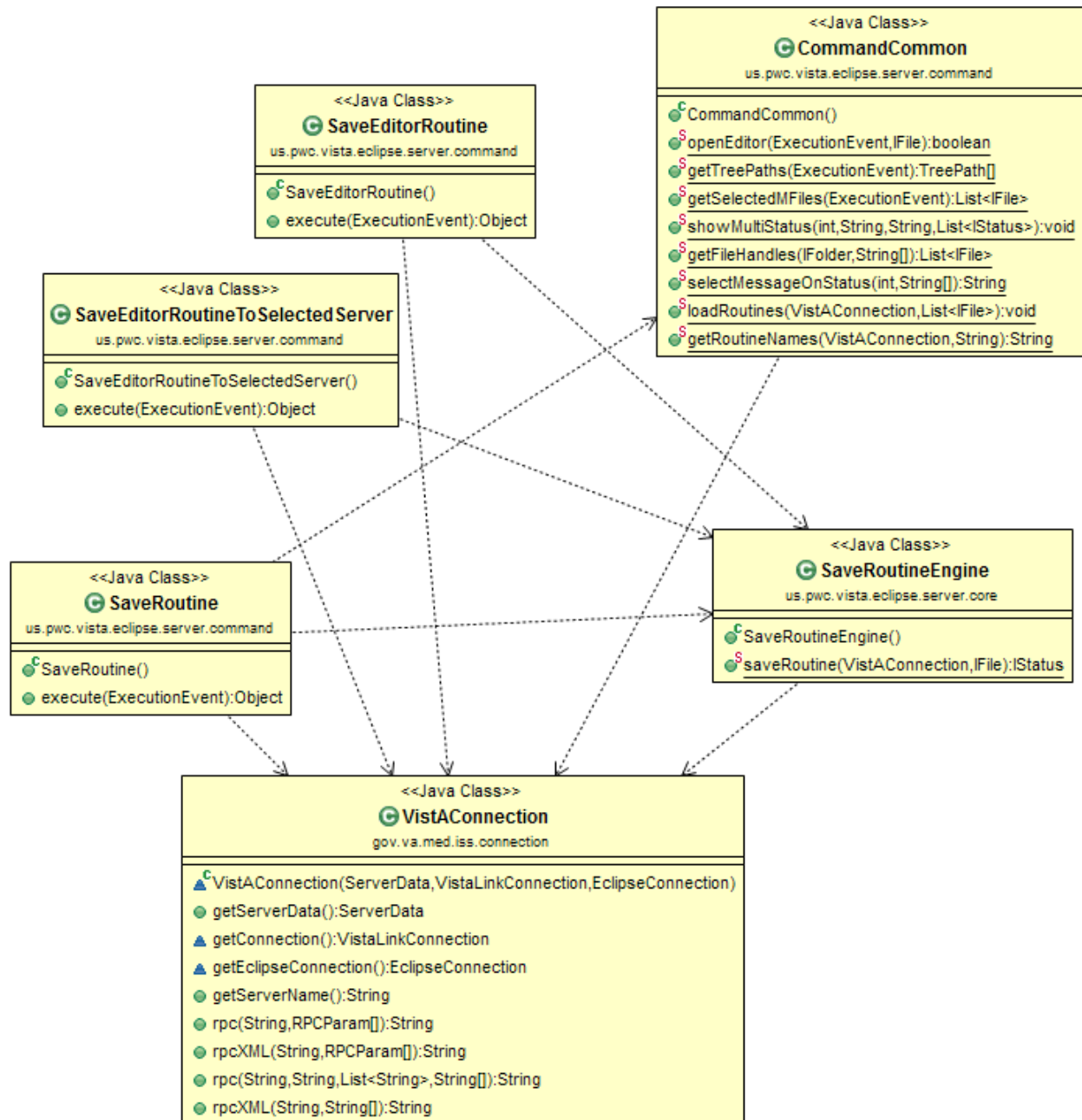


Figure 3: Classes for Saving Routines

4.4. MDebug Generic Implementation

MDebug is implemented as two distinct plug-ins: a core and a UI plug-in. The Eclipse framework will handle most of the heavy lifting and delegate the implementation at various points to MDebug. Because of this separation of duties, MDebug's implementation can be described as several small delegate implementations scattered around being called by Eclipse at several points.

[illegible]

4.5. MDebug GT.M Secure Shell and InterSystems Caché Telnet Implementation

Open Source EHR Services

MTools SDD

Eclipse and the servers; it uses the outbound stream to send debug commands and filters inbound streams to recognize break messages. All the characters related to debug commands and responses are filtered out to a Console View while the actual program input and outputs are handled in the Terminal View.

The following are GT.M debug commands used (assumes ^XUP being debugged):

- **Launch command:** S DUZ=1 D ^XUP W \$(0,0,0),"!!"
- **Define break point:** ZBREAK label+offset^routine:"W \$(0,0,0) ZWRITE BREAK"
- **Resume:** ZCONTINUE
- **Clear break points:** ZBREAK -*
- **Step into:** ZSTEP INTO: "W \$(0,0,0) ZWRITE BREAK"
- **Step over:** ZSTEP OV: "W \$(0,0,0) ZWRITE BREAK"
- **Step return:** ZSTEP OU : "W \$(0,0,0) ZWRITE BREAK"

Here \$(0,0,0) is used as a character combination that does not happen in a typical MUMPS program run. MDebug searches for this combination to differentiate the actual program inbound stream from the debug command response, and marks the start of the debug command response stream. The end of the debug stream is always found from the prompt, which is always assumed to be "GTM>" for GT.M. (Note: the launch command writes a "!!" after the \$(0,0,0). This signals to MDebug that the program has finished and the debug can be terminated.)

ZWRITEs are used to print all the variable values; the output of ZWRITE is parsed to populate the Variables View. Additionally, during the first connection to GT.M MDebug connects to a shell stream and assumes that the stream prompt is "\$"; currently this prompt is not configurable.

The following are InterSystems Caché debug commands used:

- **Launch command:** S DUZ=1 D ^XUP W \$(0,0,0),"!!"
- **Define break point:** ZBREAK label+offset^routine:"B": "1": "W \$(0,0,0) ZWRITE"
- **Define variable break point:** ZBREAK *VARIABLE:"B": "1": "W \$(0,0,0) ZWRITE"
- **Resume:** BREAK "C" G
- **Clear break points:** ZBREAK /CLEAR
- **Step into:** BREAK "S+" ZBREAK \$:"B": "1": "W \$(0,0,0) ZWRITE"
- **Step over:** BREAK "L" ZBREAK \$:"B": "1": "W \$(0,0,0) ZWRITE"
- **Step return:** BREAK "L-" ZBREAK \$:"B": "1": "W \$(0,0,0) ZWRITE"

The prompt for the InterSystems Caché terminal is the name of the InterSystems Caché namespace; this is configurable in VistA debug preferences.

5. References

5.1. Acronyms and Definitions

Term	Definition
Branch	Exists in a repository and contains a set of revisions in a chronological order. There may be multiple branches in a repository, tracking the same files in parallel; these branches may later be merged into the main branch.
Background job	In eclipse, a background job is processed by a thread pool allows the UI window to be responsive. Additional reference information may be found here. http://www.vogella.com/articles/EclipseJobs/article.html
Eclipse	An IDE primarily used for Java software development.
Eclipse View	A tab within Eclipse which provides application features to aid in software development. Eclipse provides many by default (e.g. search, directory explorer, and console). The Roll-and-Scroll Recorder (RASR) and JCTerm plug-ins provide their functionality inside of their own Eclipse views.
Eclipse plug-in	A downloadable extension to the Eclipse application which gives Eclipse new features for software development
EHR	Electronic Health Record
EPL	Eclipse Public License
FOIA	Freedom of Information Act
Fork	A copy of source code from one software project which creates a new separate project. Unlike a branch, there is no absolutely no intention of merging this back into its parent. Additionally, unlike a branch, it is a new project with new goals.
Git	Widely utilized Source Code Management (SCM) software used to store and track the code for MTools well. It is a distributed version control system.
Git Repository	In Git, a repository is where all the code history is stored. The code itself, at any time, is created from this meta-information.
GUI	Graphical user interface, as opposed to a text only based interface
IDE	Integrated Developer Environment: a robust, text editing application which allows software developers to write and test code
Model object	In context of this document, data-centric classes which encapsulate closely related items. Under Eclipse, these objects are separated from interface related concerns, and may exist solely in core or model plug-ins that are separate from UI plug-ins. The UI plug-ins depend on the model or core plug-ins.
Open Source	Software which is licensed under an open source license. This typically allows unrestricted modification and distribution of such licensed software.
OSEHRA	Open Source Electronic Health Record Agent
OTJ	OSEHRA Technical Journal
RPC	Remote Procedure Call: often using network connection, allows remote invocation and result gathering of a process. In the context of this document, it refers to XTDEBUG and how it is invoked from the Eclipse plug-ins.
SCM	Source Code Management
SDD	System Design Document
Software revision	A set of changes made to a software's source code; one or more (typically the latter) revisions make up a software version
SSH	Secure Shell
SWT	Standard Widget Toolkit: a GUI framework that Eclipse uses to render and handle GUIs

Term	Definition
Terminal Emulator	An application that renders text-based UIs and accepts input from a command line; no graphics, only text is supported
Thread	In the context of this document, the term thread specifically refers to threads used in the Java language. It allows for con-current processing in Java, such as handling a user interface and also parsing syntax in the background.
Thread pool	A thread pool is a collection of threads that operate on tasks in their queue. The major advantage is that it is re-uses threads instead of creating them and throwing them away.
UI	User Interface
VA	Department of Veterans Affairs
VBA	Veterans Benefits Association
Version Control System	An application which manages all revisions and branches of revisions for a software project
VistA	Veterans Health Information Systems and Technology Architecture
XTDEBUG	A MUMPS routine that communicates with the eclipse plug-in MDebug

5.2. Software Licenses

5.2.1. Software Under License

MTools	Apache License, Version 2.0
VistA-FOIA	Apache License, Version 2.0

5.2.2. License Locations

Eclipse Public License (EPL) v 1.0	http://www.eclipse.org/legal/epl-v10.html
Apache License, Version 2.0	http://www.apache.org/licenses/LICENSE-2.0.html