

Department of Veterans Affairs

Open Source Electronic Health Record Services

MTools Integrated Development Environment System Design Document



Version 1.0

June 2013

Revision History

Date	Revision	Description	Author
06/19/2013	0.1	Initial Draft	Jimmy Spivey
06/21/2013	0.2	Peer Review	Afsin Ustundag
05/24/2013	0.3	Peer Review	Meredith Watkins
05/25/2013	0.4	Peer Review	Kathleen Keating
05/25/2013	0.5	Updates per Peer Review	Afsin Ustundag
05/26/2013	1.0	Formal Review	Kathleen Keating

Table of Contents

1.	Introduction	4
1.1.	Scope	4
1.2.	Reference Documentation	4
2.	Enhancing Original MTools Plug-ins	5
2.1.	MEditor	6
2.2.	MDebug.....	6
3.	Implementation Specifications for MTools Features.....	6
3.1.	MEditor	6
3.2.	MDebug.....	7
3.2.1.	Concurrent Processing of the Eclipse UI Concerns and XTDEBUG Backend Concerns.....	7
3.2.2.	Case Example: How to Work with Separate UI and Core Plug-ins	7
4.	MTools Class Diagrams	7
4.1.	MEditor	7
4.2.	MDebug.....	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
	Approval Signatures	14

1. Introduction

The Department of Veterans Affairs (VA) has contributed the latest U.S. Department of State Freedom of Information Act (FOIA) release of the Veterans Health Information Systems and Technology Architecture (VistA) codebase to Open Source Electronic Health Record Agent (OSEHRA), the custodial agent that serves as the central governing body of a new open source community. The Open Source Electronic Health Record (EHR) Services project includes VistA Data Comparison, VistA System Test Platform, VistA Refactoring, VistA System Test Scripts, Veterans Benefits Administration (VBA) System Test Platform, Eclipse plug-in tool, and VistA Meaningful Use Certification.

1.1. Scope

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 VistA development has been limited. This document describes the work performed by the Open Source EHR Services Project Team, on behalf of the 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.

1.2. Reference Documentation

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

- MTools, and Eclipse-Based IDE for VistA (OSEHRA Technical Journal [OTJ] overview)
- MTools Usage and Installation 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-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">a. Support added for indirection in DO commandsb. Support added for routines that do not end with QUIT on the last linec. Support added for quoted extrinsic functionsd. Fixed the issue of commented lines being executede. Support added for expressions in WRITE commandsf. Support added for extrinsic functions to other routinesg. Support added for expressions in FOR loopsh. 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 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, hierarchical 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. 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 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 the original plug-in was written as Eclipse Actions, which would then update and change Eclipse Views and their Standard Widget Toolkit (SWT) components.

The original debugger'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. 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", doesn't cause Eclipse to become unresponsive.

3.2. MDebug

3.2.1. Concurrent Processing of the Eclipse UI Concerns and XTDEBUG Backend Concerns

MDebug was created by following an [article](#) on how to create a custom Eclipse Debugger for any language. This article was written in 2004 and is partially out of date with regards to asynchronous processing. The article 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 back-end 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 UI and Core Plug-ins

A custom console for handling READ and WRITE commands was added to MDebug. This console relies on the Eclipse's 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, 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

4.1. MEditor

The MEditor plug-in has an almost purely procedural implementation in lieu of an object oriented approach. This means that instead of many objects relating to each other and each handling

separate responsibilities, the code is only organized into procedures and java methods. These procedures are spread into a logical class which has many related procedures, but rarely any instance (object) variables. Some use instance variables, but the object is often a singleton.

The MEditor class diagrams show dependencies, while the MDebug diagrams do not, because the MEditor is procedural and users want to see if the classes call each other. Otherwise, since there are not any instantiation relationships in the MEditor classes, there would be no relations to show in the diagram.

Figure 1 below illustrates loading and saving routines from and to the server:

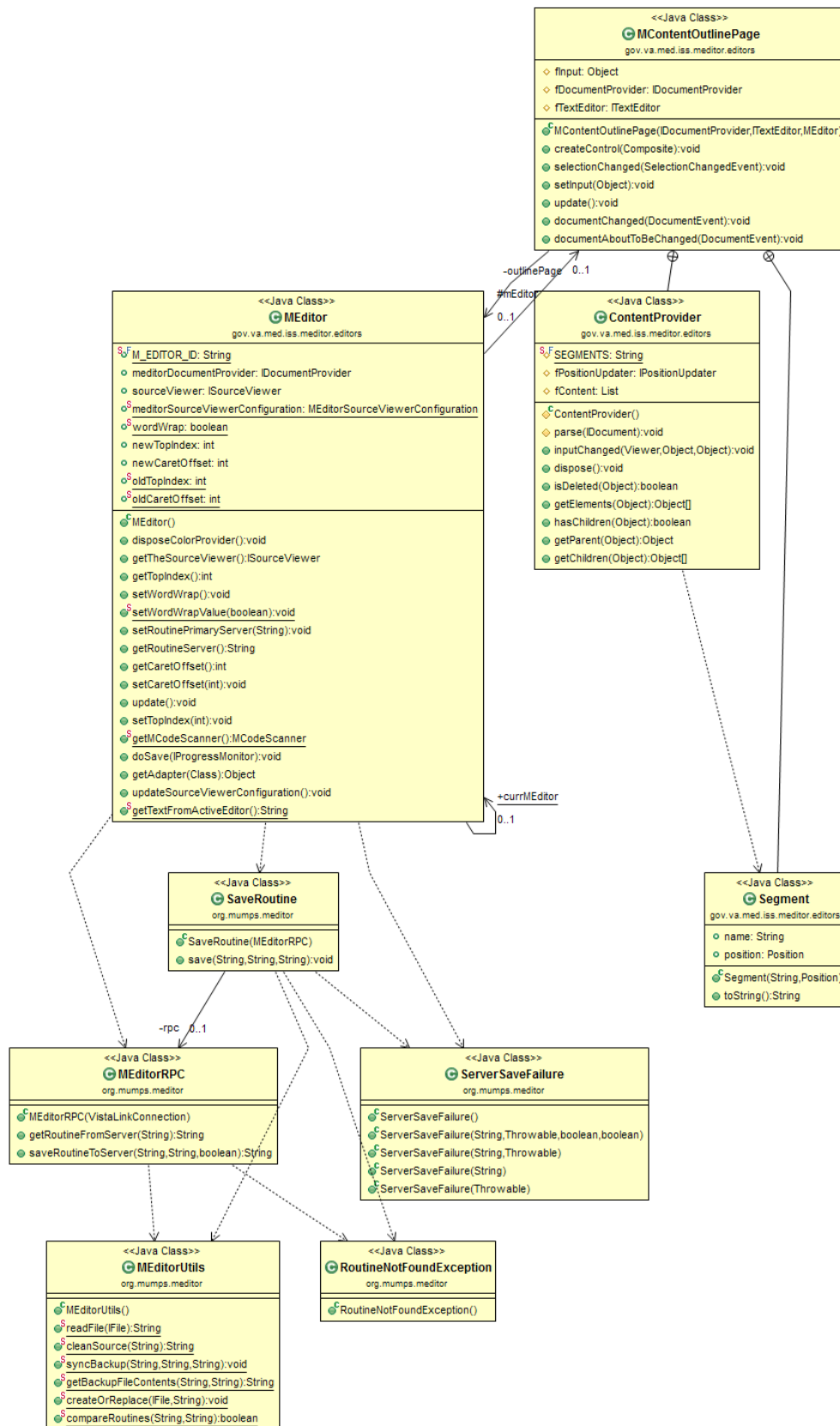


Figure 1: MEditor Loading and Saving Routines

The classes in Figure 2 below are for all of the MEditor actions. Actions are the icons that the user can select from the Vista menu, such as “Global Directory”. This also includes Routine Load, the class RoutineEditAction. Many of the actions are self-contained and do not rely on procedures from other classes.

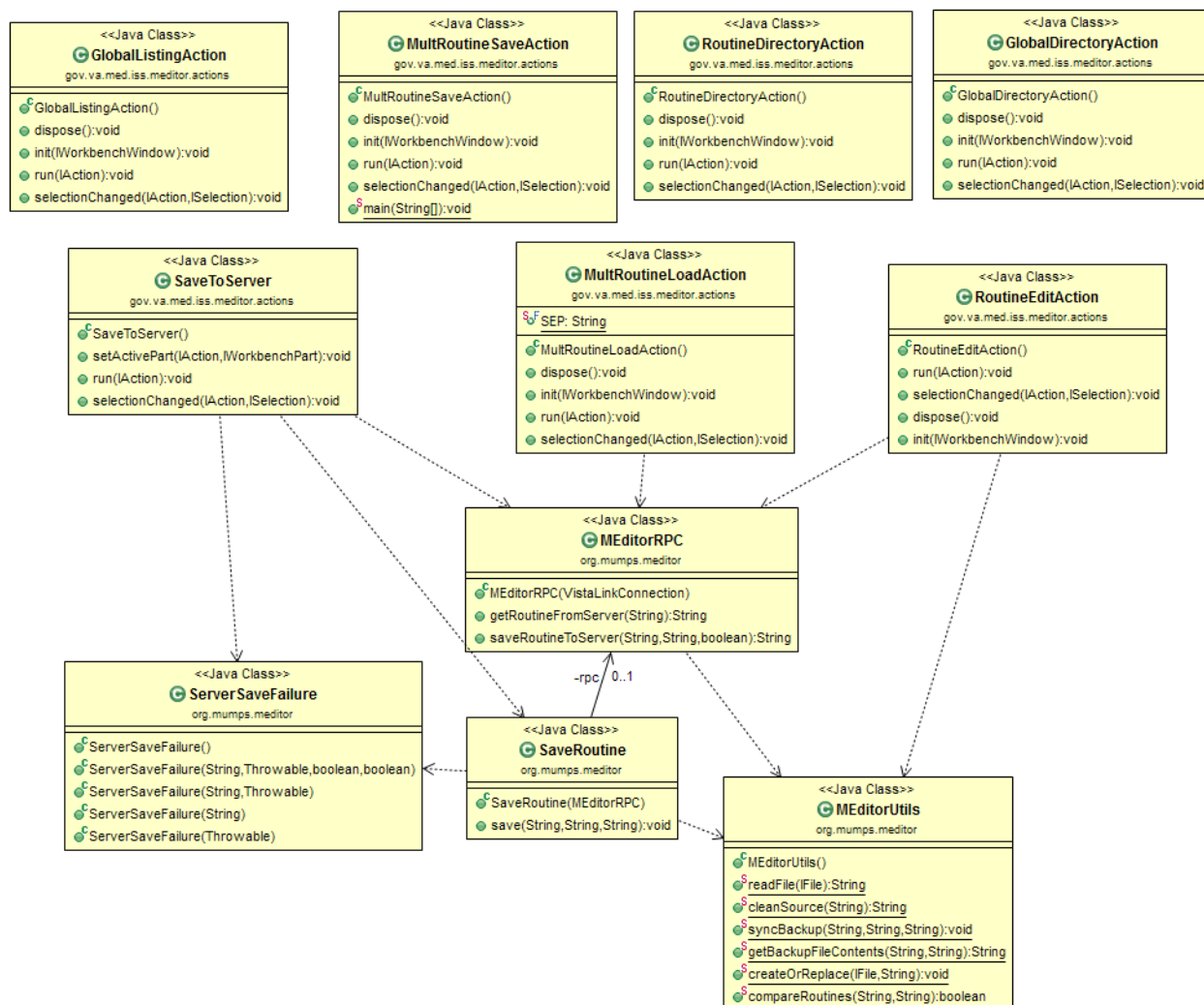


Figure 2: Classes for MEditor Actions

4.2. MDebug

MDebug is implemented as two plug-ins: a core and a UI plug-in. These plug-ins are quite different in design than the other plug-ins because they work closely with the Eclipse Platform and its underlying classes. These classes are object oriented in design, and as a result rely on polymorphism. 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 (whereas, MEditor is just a few single entry points which handle all the work with little use of Eclipse’s platform and is procedural instead of object oriented).

Despite MDebug having mostly a delegate based, lightweight implementation, there is at least one area where it must do heavy lifting. It cannot possibly understand how to debug any given language, so all of those implementation details are defined in various XTDEBUG packages and utilized in the MDebugTarget class. This class is in the core plug-in, and there are many classes which relate to it as seen in figure below MDebugTarget, and those classes which relate to it, are where the debugger is implemented at a core or model level. Figure 3 below shows MDebug's Model objects:

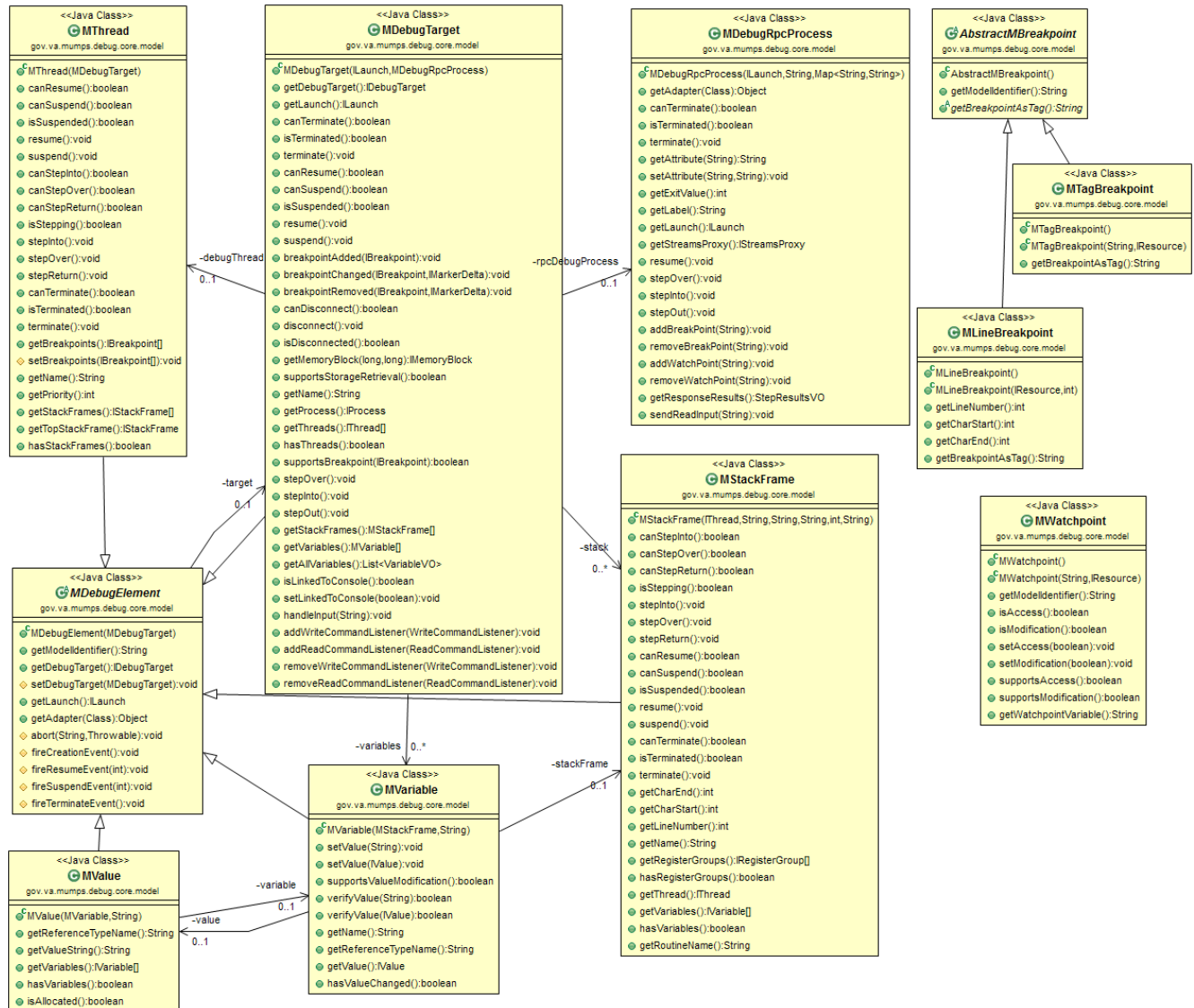


Figure 3: Model Objects of MDebug

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.
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

Approval Signatures

This section is used to document the initial approval of the draft Open Source EHR Services MTools Integrated Development Environment System Design Document; updates will be submitted as needed.

All members of the governing Open Source EHR Services Management Team are required to sign. Please annotate signature blocks accordingly.

Signed:

Date:

LaWanda Wells, Contracting Officer's Representative

Signed:

Date:

Maureen Coyle, Program Manager