Department of Veterans Affairs

*Open Source Electronic Health Record Services*

MTools IDE

System Design Document



Version 0.1

June 2013

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Revision | Description | Author |
| 06/19/2013 | 0.1 | Initial Draft | Jimmy Spivey |
|  |  |  |  |
|  |  |  |  |

Table of Contents

[1. Introduction 4](#_Toc359447737)

[2. Enhancing the original MTools plugins 4](#_Toc359447738)

[2.1. MEditor 4](#_Toc359447739)

[2.2. MDebug 4](#_Toc359447740)

[3. Implementation specs for MTools features 4](#_Toc359447741)

[3.1. MEditor 4](#_Toc359447742)

[3.2. MDebug 5](#_Toc359447743)

[4. MTools Class Diagrams 5](#_Toc359447744)

[4.1. MEditor 5](#_Toc359447745)

[4.2. MDebug 7](#_Toc359447746)

[5. Appendix 8](#_Toc359447747)

[5.1. Acronyms and Definitions 8](#_Toc359447748)

[5.2. Software Licenses 9](#_Toc359447749)

[5.2.1. Software under License 9](#_Toc359447750)

[5.2.2. License Locations 9](#_Toc359447751)

# 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.

# Enhancing the original MTools plugins

## 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 cleanness to its design. The commit differences on the git repo can clearly show the new code replacing the old code in detail.

## MDebug

The original debug plugin was migrated to two new plugins. Only very small portion of the original code remains in the new plugins. Many new features were added, and all the existing features were brought over. The main difference is that the prior plugin was written as Eclipse Actions, which would then update and change Eclipse Views and their SWT components. The GUI was not as finely presented as other debuggers, and being able to present highly readable details for a debugger is very important. So the migration was to the Eclipse Debug Model. This is how all other languages which use Eclipse implement debuggers. It provides a widely adopted and familiar debug GUI and behavior. As well as fast response and lots of information. Additionally there was no other viable way to get the newer debug features that were desired, specifically adding breakpoints to a line in MEditor.

# Implementation specs for MTools features

## MEditor

Eclipse, as a GUI based IDE(TODO, define), works from a single main or UI thread. It also manages a thread pool (TODO, define) 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 this work.

Given Eclipse’s multithreading workflow, changes were made to put much of the actions and other processes that MEditor creates into background jobs. This enhances the response when editing text or clicking any of the actions. By default, the Eclipse Debug Platform puts all of its actions into background jobs already. Therefore clicking an action, such as “step over” or “resume” doesn’t cause Eclipse to hang at all.

## MDebug

The Eclipse Platform Debug Model—look, behavior, performance, acceptance

how the XTDEBUG RPC works wrt to how eclipse invokes it

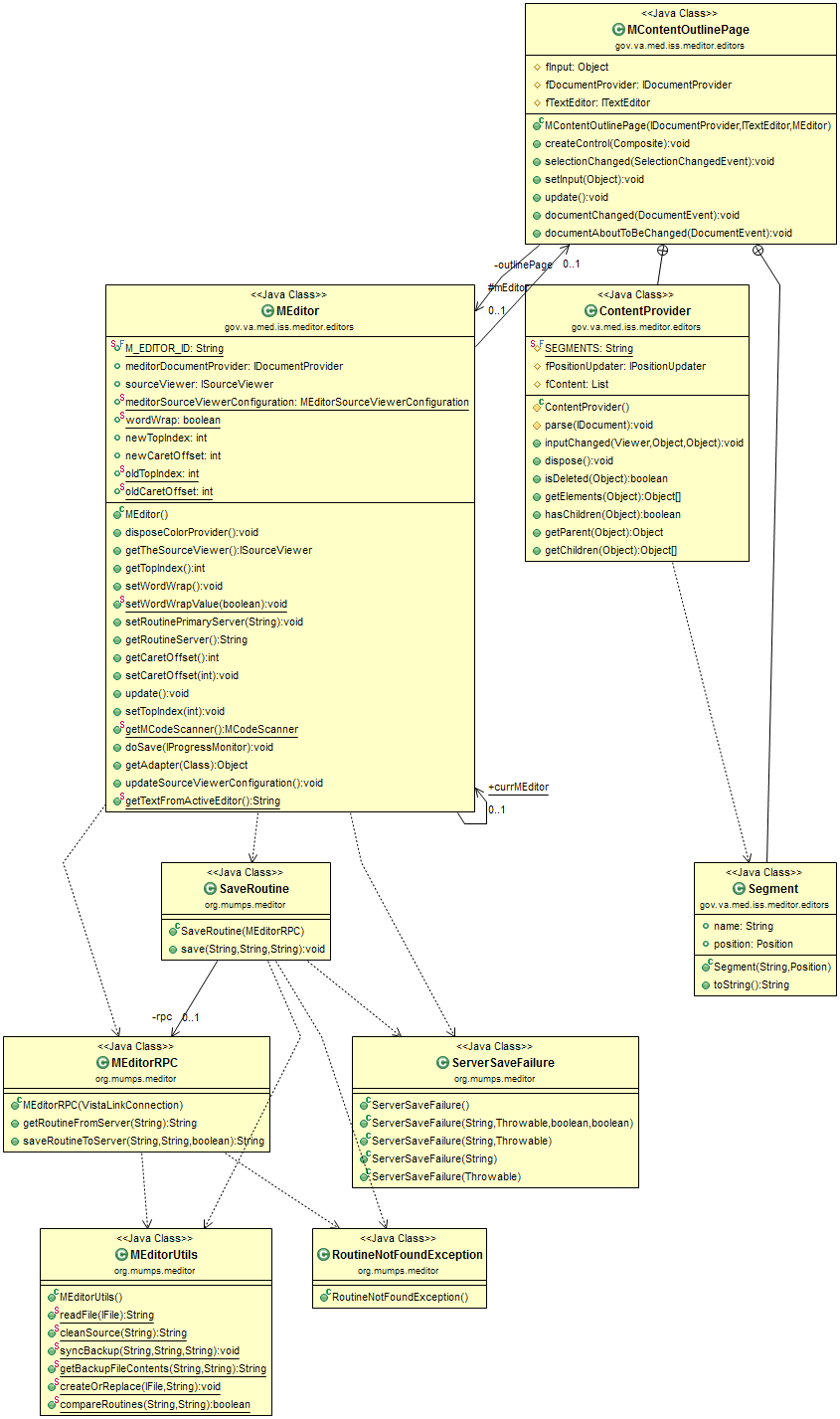
How the console was added: uses Eclipse Console View, how it registers listeners

# MTools Class Diagrams

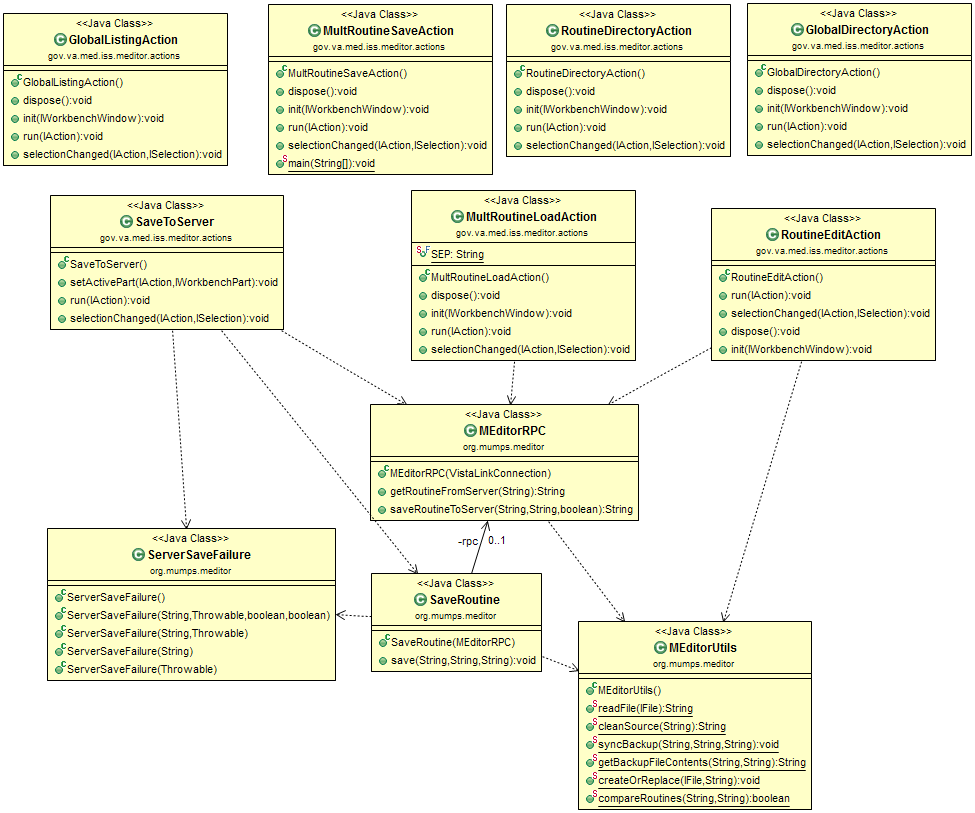
## MEditor

The MEditor plugin 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 organized into procedures, java methods, only. These procedures are spread into a logical class which has many procedures related to it, but rarely any instance (object) variables. Some use instance variables but the object is often a singleton.

The MEditor class diagrams show dependencies, whereas the MDebug do not. This is because the MEditor is procedural, and we want to see if the classes depend (call) each other. Otherwise since there aren’t any instantiation relationships in the MEditor classes, there would be no relations to show in the diagram. The diagram bellow is for loading and saving routines from and to the server.



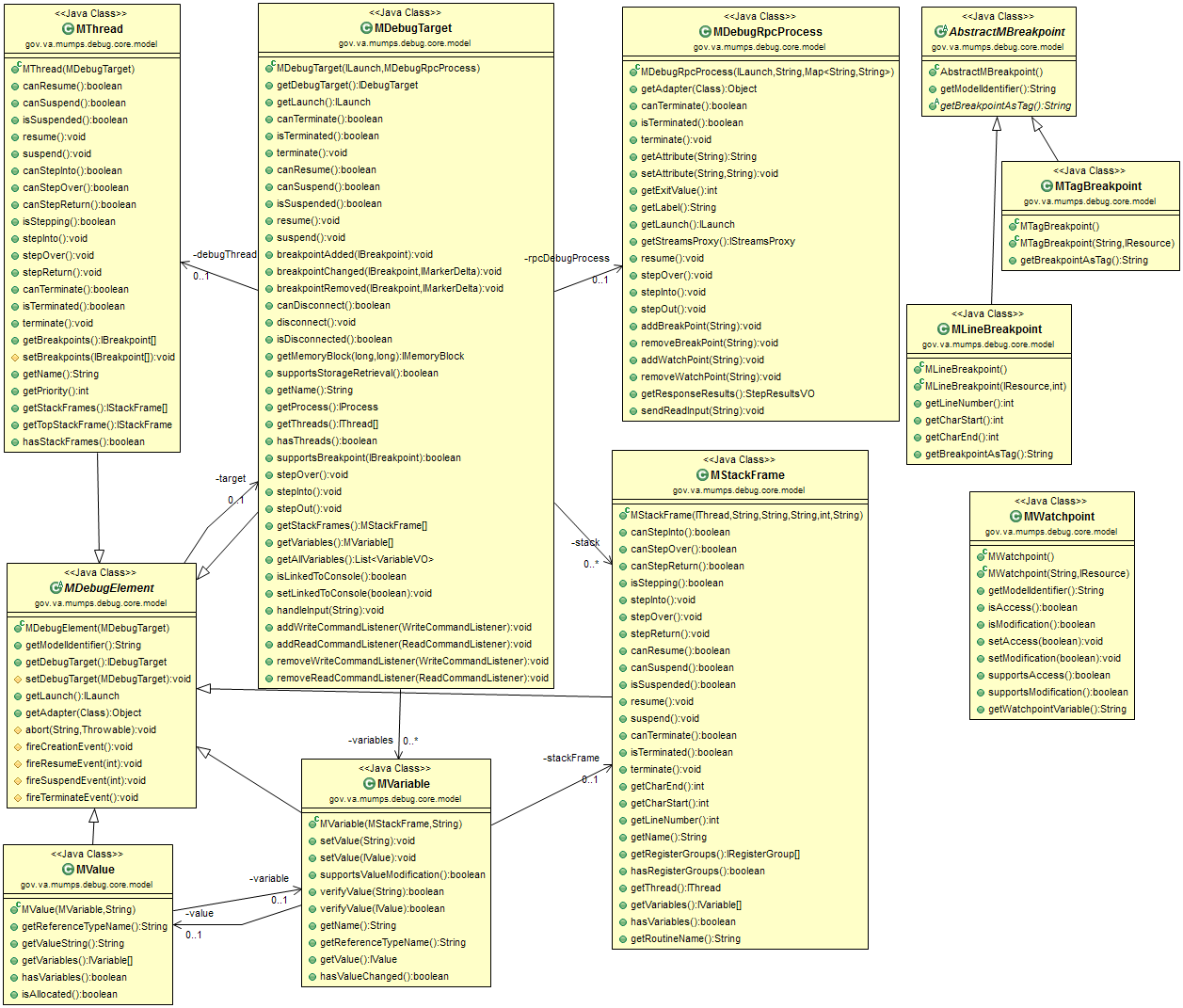
The classes in the bellow class diagram are for all of the MEditor actions. Actions are the icons that the user can select from the VistA menu, such as “Global Directory”. This also includes Routine Load, the class RoutineEditAction. Many of the actions are self-contained and do not rely on procedures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



## MDebug

MDebug is implemented as 2 plug-ins, a core and a ui (user interface) plug-in. These plug-ins are quite different in design than the other plugins, because they work so closely with the Eclipse Platform and its underlying classes. These classes are Object Oriented in design, and as a result of that they also 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 can be implementation can be described as several small delegate implementation 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 plugin, and there are many other classes which relate to it, as seen in figure TODO. MDebugTarget, and those classes which relate to it, are where the debugger is implemented at a core or model level. The figure bellow shows MDebug’s Model objects. (TODO: add model objects to defintions)



# Appendix

## Acronyms and Definitions

|  |  |
| --- | --- |
| Branch | Exists in a repository. 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. |
| Eclipse | An IDE primarily used for Java software development. |
| Eclipse View | A tab inside of Eclipse which provides application features to aid in software development. Eclipse provides many by default (e.g. search, directory explorer and console). RASR and JCTerm Plug-in provide their functionality inside of their own Eclipse Views. |
| Eclipse Plug-in | An extension to the Eclipse application, which can be installed. It gives Eclipse new features for software development. |
| EPL | Eclipse Public License |
| 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. |
| 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. |
| Open Source | Software which is licensed under an open source license. This typically allows unrestricted modification and distribution of such licensed software. |
| 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 | TODO |
| Terminal Emulator | An application that renders text-based user interfaces and accepts input from a command line. No graphics, only text is supported. |
| 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 |

## Software Licenses

### Software under License

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

### License Locations

|  |  |
| --- | --- |
| Eclipse Public License 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 |