

Department of Veterans Affairs

Open Source Electronic Health Record Services

MTools Installation and Usage Guide



Version 1.0

June 2013

Table of Contents

1.	Installation	3
1.1.	Prerequisites.....	3
1.2.	Download Latest Files.....	3
1.3.	Install KIDS Packages and Update VistALink	3
1.4.	Run VistALink Job	3
1.5.	Install the Plug-in	3
1.6.	Configure the Plug-in	5
2.	Usage	5
2.1.	Eclipse Projects.....	5
2.1.1.	Creating a New Project.....	5
2.1.2.	Opening an Existing Project	5
2.2.	Using MEditor	6
2.3.	Loading Routines	6
2.3.1.	Backup Files.....	7
2.4.	Saving Routines	7
2.5.	Debugging.....	7
2.5.1.	Adding Breakpoints.....	8
2.5.1.1.	Line Breakpoints.....	8
2.5.1.2.	Tag Breakpoints.....	9
2.5.1.3.	Watchpoints	9
2.5.2.	Starting a Debug Session	9
2.5.3.	Variables.....	10
2.5.4.	Interactive Console.....	11
2.6.	MTools Context Menu Tools.....	11
2.6.1.	Report Assumed Variables	13
2.6.2.	Report Errors	15
2.6.3.	Report Occurrences	15
2.6.4.	Report Quit Types.....	16
2.7.	M Refactor Context Menu Items.....	16

1. Installation

1.1. Prerequisites

MTools integrated development environment (IDE) supports both Windows and Linux.

Application	Version	Notes
Veterans Health Information Systems and Technology Architecture (VistA)	Any	This is currently tested with VistA-Freedom of Information Act (FOIA), available here . VistA installation instructions are available on the Open Source Electronic Health Record Agent (OSEHRA) site here .
Eclipse	Indigo	
Java Runtime	7	

1.2. Download Latest Files

Download the latest version of MTools [here](#) and unpack the file. It contains the plug-in jar files needed and the Kernel Installation and Distribution System (KIDS) packages.

1.3. Install KIDS Packages and Update VistALink

From the unpacked file, the KIDS packages are located under */MiscDependencies/KIDS/*. Install all three packages: MDebugger, MEditor, and Utilities. KIDS installation instructions can be found [here](#).

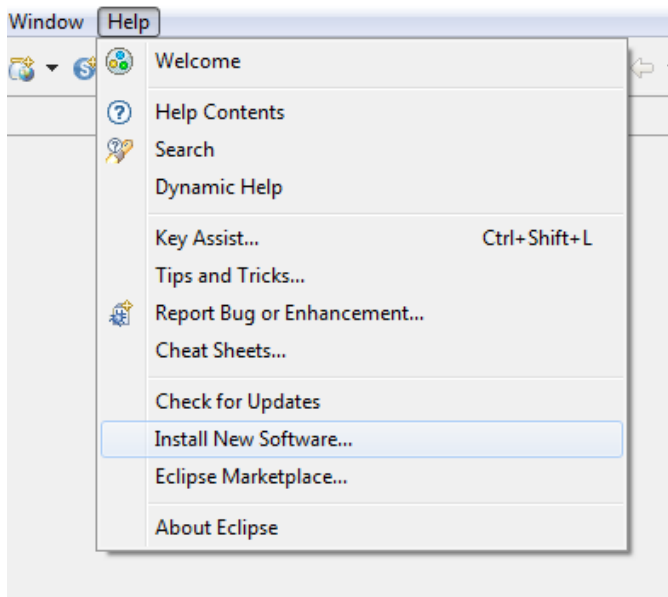
If a user is working in Linux, VistALink routines need to be updated with those found [here](#); these files can be loaded using *D ^%R/* utility from a MUMPS Terminal.

1.4. Run VistALink Job

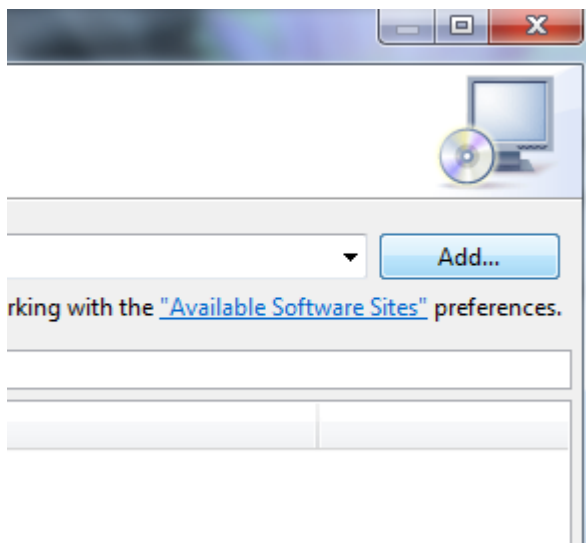
Enter the MUMPS command `JOB LISTENER^XOBVTCPL(8001)` to start VistALink. This action is required for the plug-in to connect with and talk to the server.

1.5. Install the Plug-in

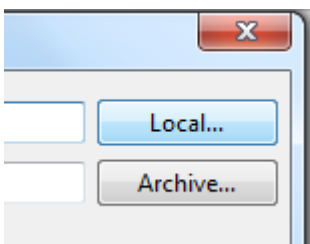
1. Open Eclipse
2. Click 'Help' → 'Install New Software'



3. Then Click 'Add' in the top right



4. Select the button named 'Local'; this is towards the top right



5. Select the directory where the zip file was unpacked. Then choose the 'MToolsUpdateSiteProject' directory.
6. After clicking 'OK', the newly added update site will automatically be selected
7. Click 'Next' and follow the prompts to install the plug-in

1.6. Configure the Plug-in

- 1) From Eclipse's main menu (the top most bar) Click 'Window' → 'Preferences' → 'VistA' → 'Connection'
- 2) Remove the dummy "Primary" connection value
- 3) Add a new connection in the format of *[Name];[IP Address or hostname];[port number];[blank or an eclipse project name]*; a typical value would be `local;127.0.0.1;8001;`
- 4) Click 'OK'

2. Usage

2.1. Eclipse Projects

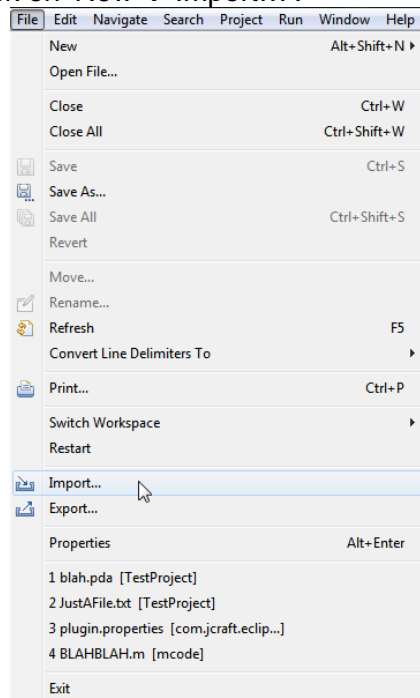
All files in Eclipse must belong to a project since Eclipse can be described as project centric. To begin using MTools IDE there are two options: (1) a new project can be created and files imported or (2) an existing project can be opened containing existing files and the capability to import new files. All files in Eclipse can be removed and/or deleted, or imported via a link as opposed to being in the same directory where the project's root is. The project root is a single directory which contains the .project file. A workspace, however, is a parent to a project and contains 0 or many projects. Projects may or may not exist in the workspace directory root.

2.1.1. Creating a New Project

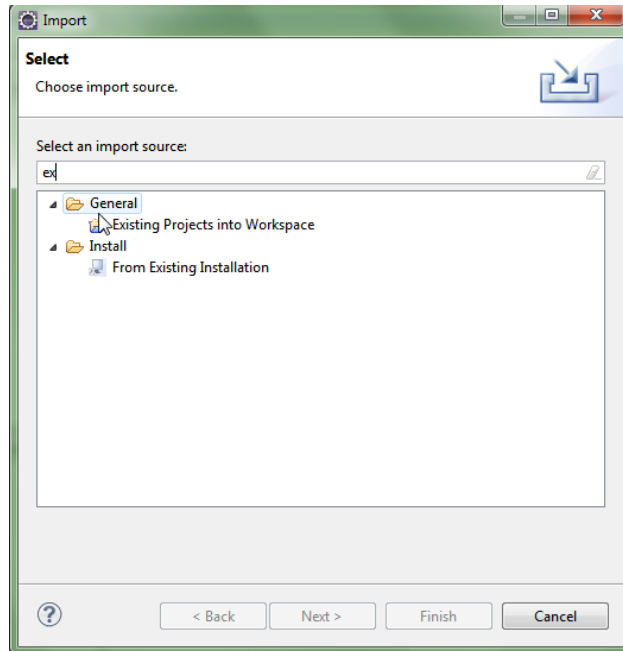
A standard Eclipse project can be created via 'New' → 'New Project...'. This opens the new project wizard. The standard Eclipse project can be chosen from 'General' → 'Project'.

2.1.2. Opening an Existing Project

To open an existing project, click on 'New' → 'Import...'



From the Import Wizard choose 'Existing Projects into Workspace' under the 'General' folder.



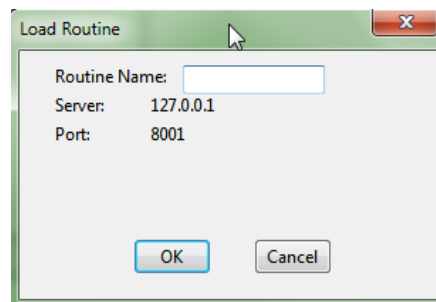
This can be filtered easily by typing 'ex'. Using the file dialog navigate to the directory which contains an existing project (a directory containing a .project file). This will typically be from a version controlled system such as Git, which will typically contain a parent root directory that has the .project file in it, and other files with binaries and source files in children folders.

2.2. Using MEditor

To begin using MEditor (a custom editor for MUMPS language, part of MTools IDE), simply open a file ending with the suffix '.m'. A custom editor which includes syntax coloring, new contextual menus, and the outline view (by default the vertical right pane) will display tags in the current routine.

2.3. Loading Routines

Routines can be loaded from the server. Either click the green 'M' button in the Eclipse toolbar or on the top menu click 'VistA'→'Load M Routine'.



When a routine is loaded from the server a file is created on disk, either in the root directory if the project has no directories, or a dialog is displayed to prompt the user which directory to load into if there are sub directories in the project. There is also support for the VistA-FOIA structure, which uses a packages.csv file to determine where VistA routines are loaded. For example, if a user attempts to load a routine such as GMPL, it will place this file into *Packages/Problem List/*

automatically. Lastly, if the file already exists in the project, it will always load to this location and attempt to sync showing a diff if anything changes and prompting which file to choose.

2.3.1. Backup Files

Whenever a file is successfully loaded, a backup file is either created or updated in the /backups folder of the project. This backup file contains the latest server version locally, which is used for comparing when a save occurs. The file format is “[routine name] yyyyMMdd.m”. If a backup file exists from a previous day, that file will be replaced with the one containing today’s date; there will never be multiple backup files for the same routine.

These backup files show the latest known server version. Thus, when a routine is either loaded from or saved to the server, this file is updated to what is on the server. It is only used by MEditor to later compare a routine being saved to the server. Users are free to browse the contents of these files if they want to see what MEditor last pushed or pulled from the server.

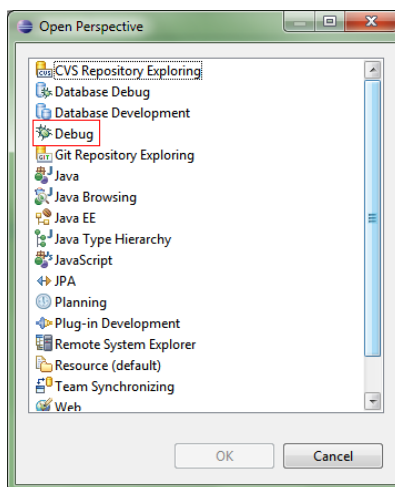
2.4. Saving Routines

Routines are saved, just like all Eclipse files, to the disk. Additionally, when the save button is clicked, it will also attempt to sync with the server, unless it is working in offline mode, as defined in the preferences (‘Windows’ → ‘Preferences’ → ‘VistA’).

When a routine is to be saved to the server, it will first fetch a copy of that routine from the server. MEditor will then compare what is on the server to the latest backup file for that routine. The latest backup file is a copy of what was last on the server. If this is different, a prompt will be shown. If the file to be saved (the actual routine file, not the backup) is different, no comparison selection prompt is shown because the files should be different as there should be changes to submit. An informational dialog will be shown if the routine to be saved is the same as what is on the server.

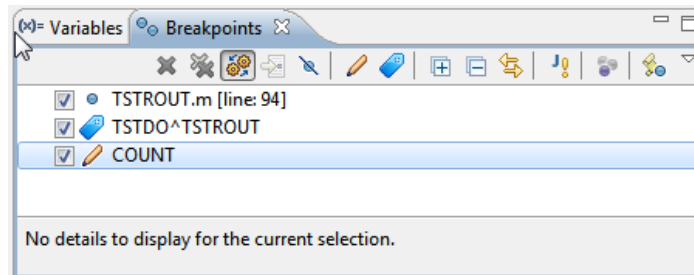
2.5. Debugging

To use the debug features, enter the Eclipse Debug Perspective. Click the change perspective menu in the top right of the Eclipse IDE. Then choose ‘Debug’.



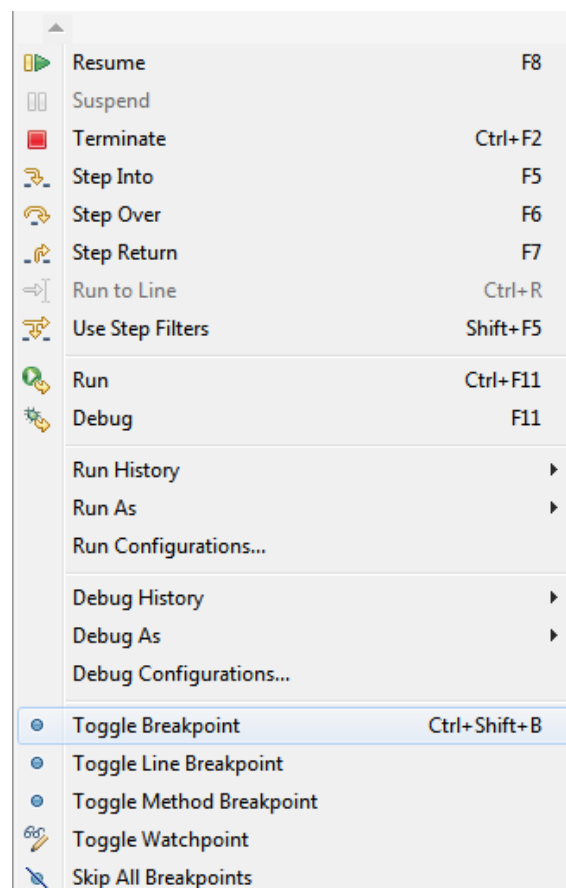
2.5.1. Adding Breakpoints

The MDebug Plug-in has three types of breakpoints: line breakpoints, tag breakpoints, and variable watchpoints. All breakpoints can be seen from the breakpoints view under the debug perspective.

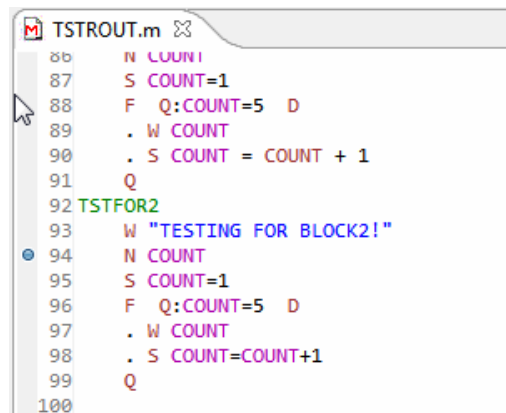


2.5.1.1. Line Breakpoints

Open the M Code in an editor that the user wants to debug. Either press 'Ctrl + Shift + B' or go to the Run Menu at the top of the Eclipse application and select 'Toggle Breakpoint' or 'Toggle Line Breakpoint'.



In addition, users can also double click the vertical bar left of the editor to add a line breakpoint. It is important to add breakpoints otherwise the debugger will run until it completes and terminates.



2.5.1.2. Tag Breakpoints

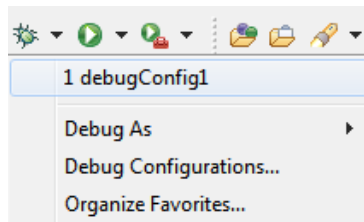
Arbitrary tags can also be entered (e.g.: *TAG^ROUTINE*). To add an arbitrary tag, click on the blue tag icon from the breakpoints view.

2.5.1.3. Watchpoints

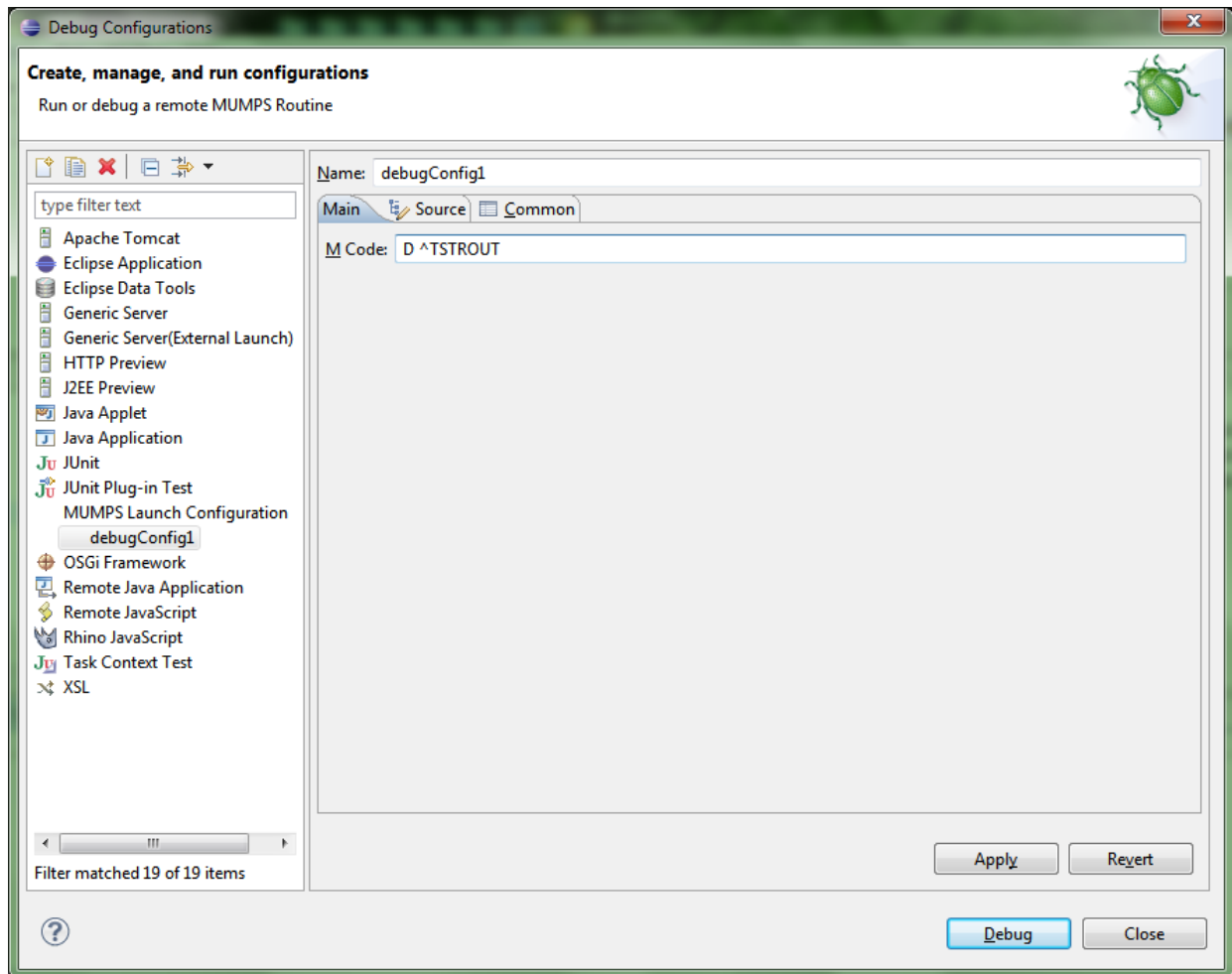
Variable watchpoints can be added. They are triggered when a variable value changes. They too are added from the breakpoints view, by clicking the pencil icon.

2.5.2. Starting a Debug Session

To start the debugger, a new launch configuration must be created. Create a new debug configuration by:

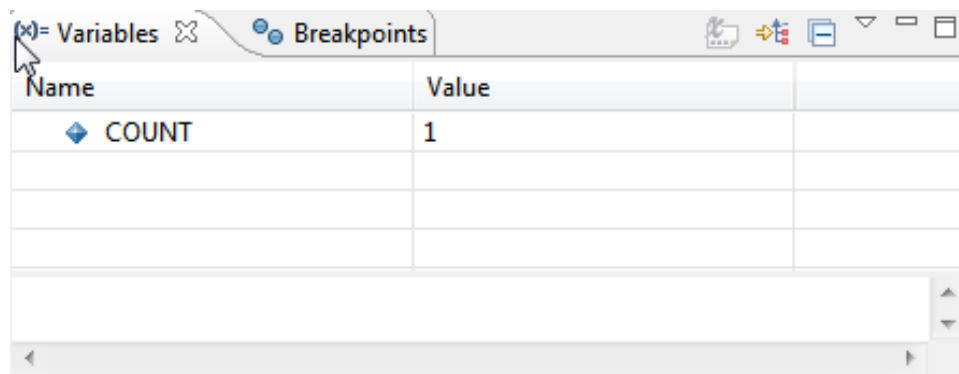


Enter the M code which the user wants to debug and which ideally will encounter the user's breakpoints.

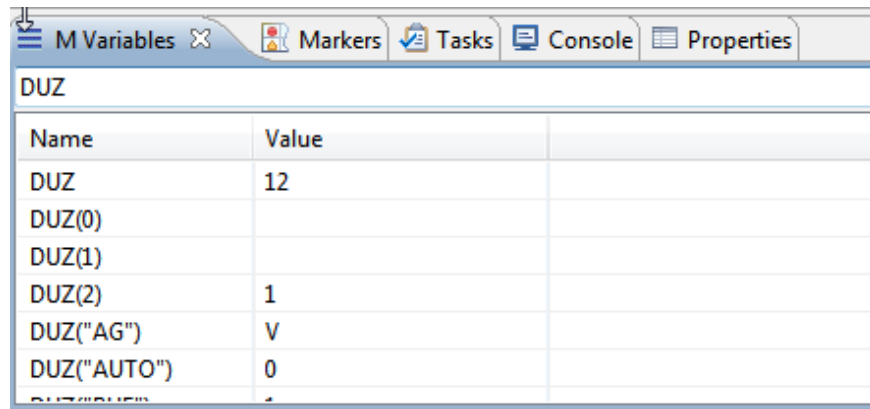


2.5.3. Variables

Variables will be displayed in the default Eclipse Debug Variables view. But only those variables created after the debug session started are shown.



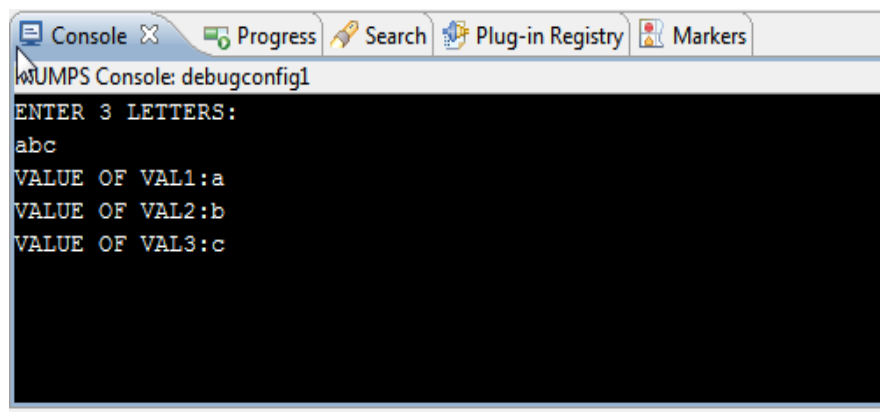
Since the default view isn't showing all variables currently defined on the server, there is a new custom view created for that. Open it by clicking on 'Window' → 'Show View' → 'Other...' and choose 'M Variables'. It can also be filtered by using the text box on top of the variable viewer.



Name	Value
DUZ	12
DUZ(0)	
DUZ(1)	
DUZ(2)	1
DUZ("AG")	V
DUZ("AUTO")	0

2.5.4. Interactive Console

MDebug support is an interactive console which displays MUMPS WRITE commands and can capture input from READ commands. The console uses the default console view and automatically displays itself as WRITE commands output new input or when a READ command is ready to accept input.



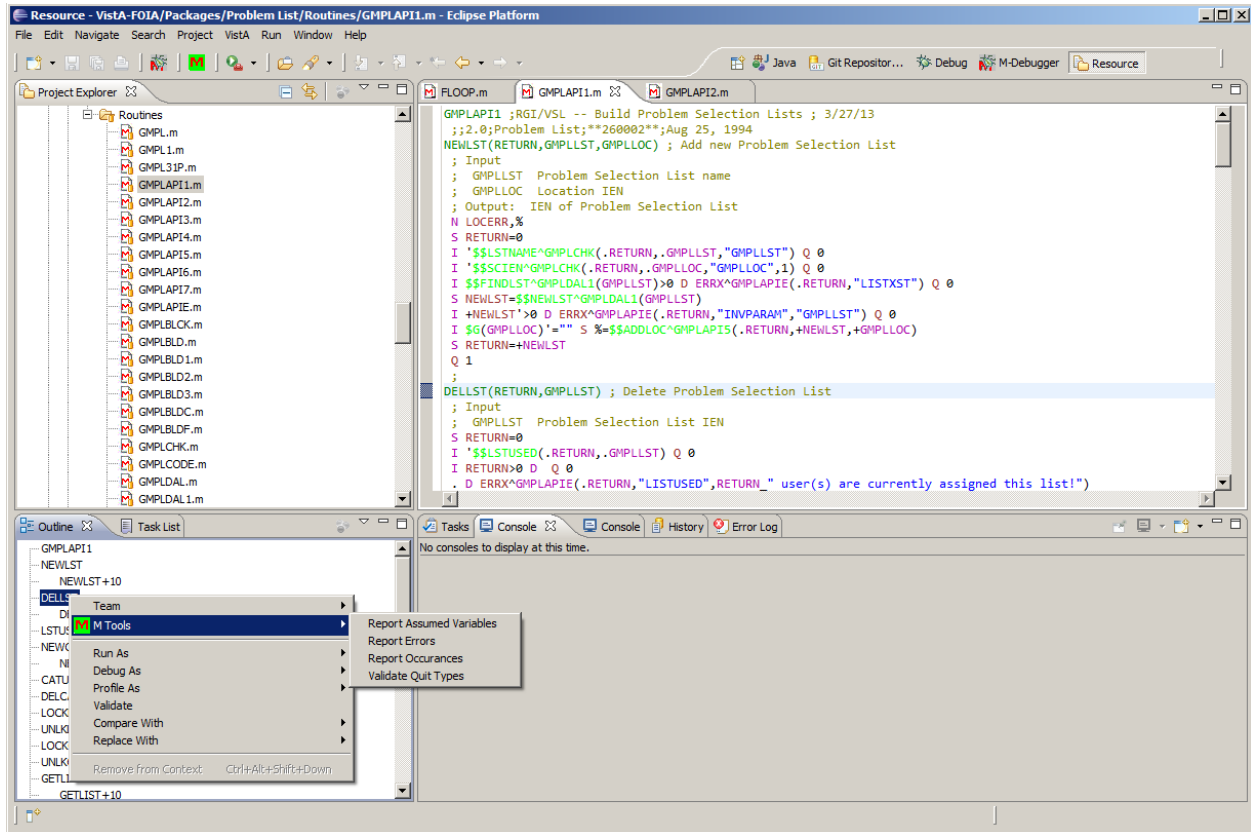
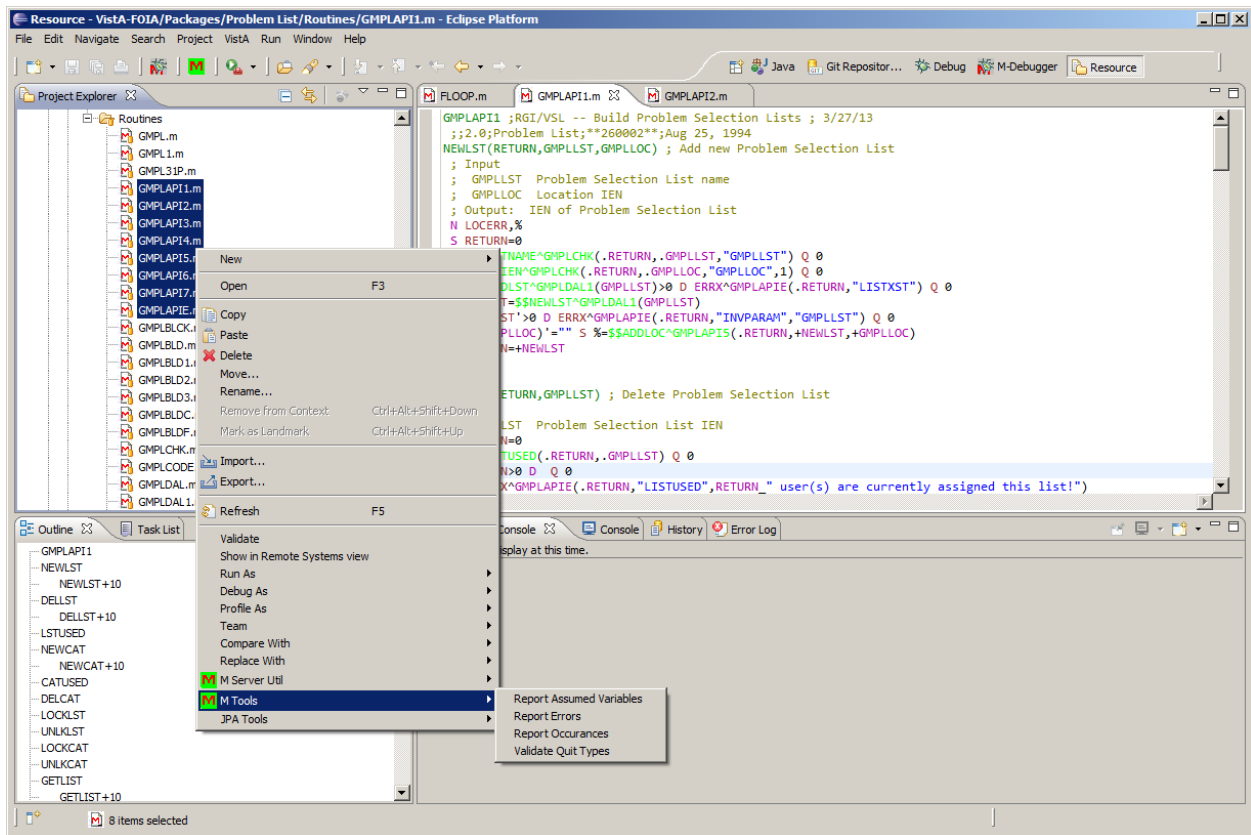
```

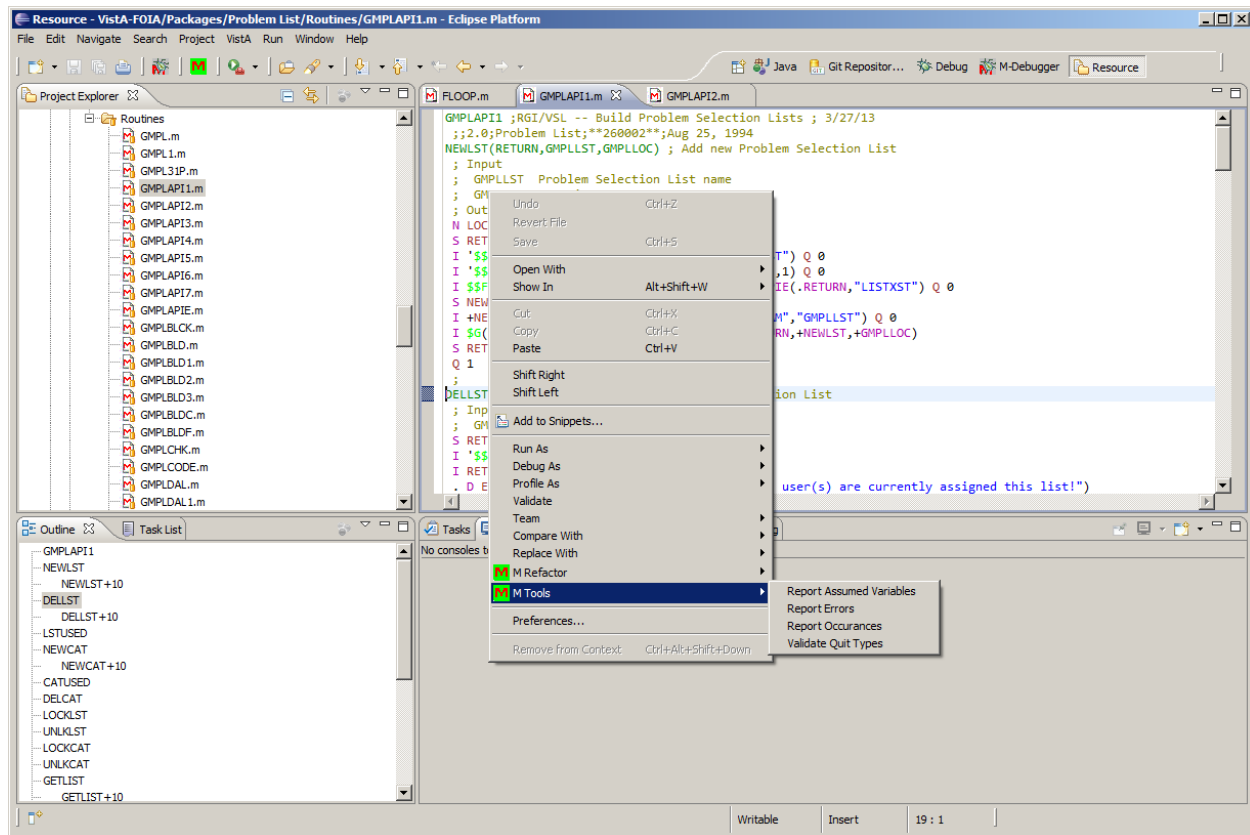
MUMPS Console: debugconfig1
ENTER 3 LETTERS:
abc
VALUE OF VAL1:a
VALUE OF VAL2:b
VALUE OF VAL3:c

```

2.6. MTools Context Menu Tools

A number of M code validation and analysis tools are available in MTools IDE. These tools can be found under M Tools menu item in context menus of Project Explorer view, Outline view, and MEditor.



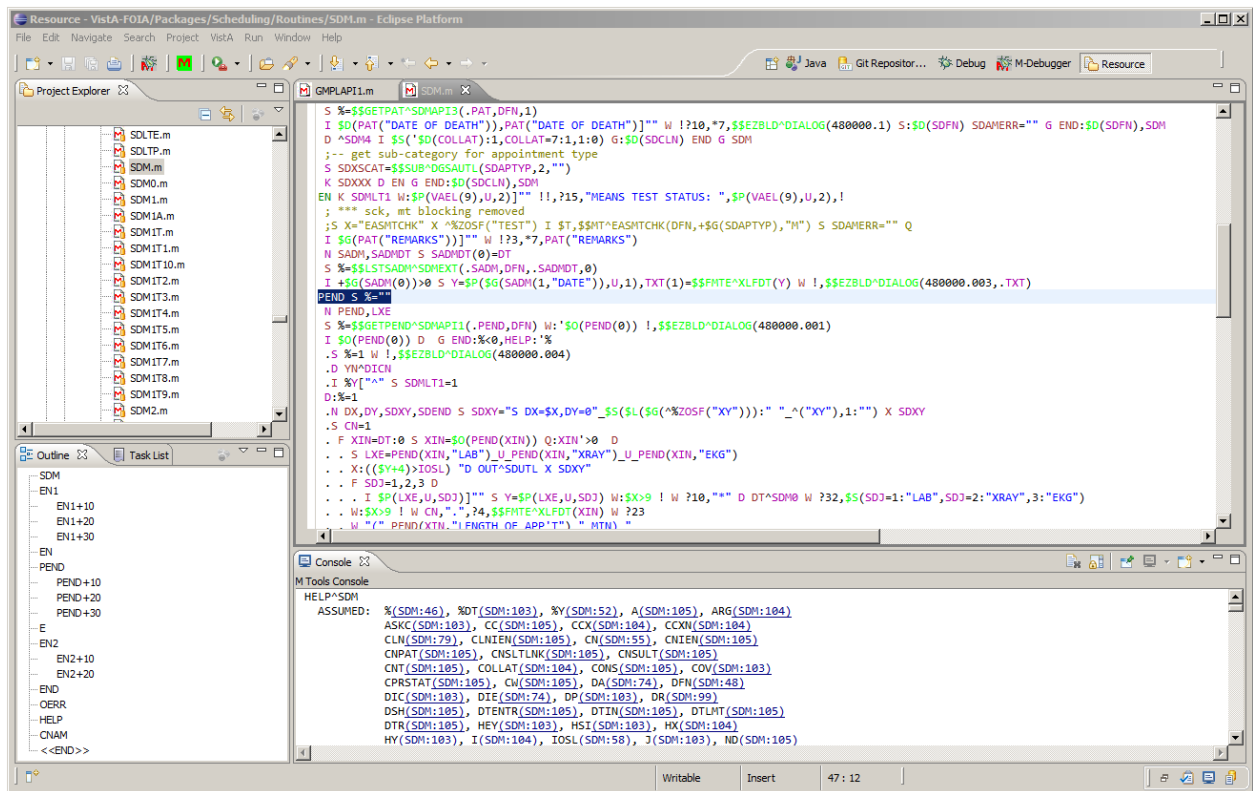


Currently there are four tools: Report Assumed Variables, Report Errors, Report Occurrences, and Validate Quit Types. Each of these tools is run for each entry point tag in routines. When selected from Outline view, they are run for a specific tag while; when they are run from MEditor, they are run for all the tags in the routine that is being edited in MEditor. All the tools can be run on multiple file selections from Project view.

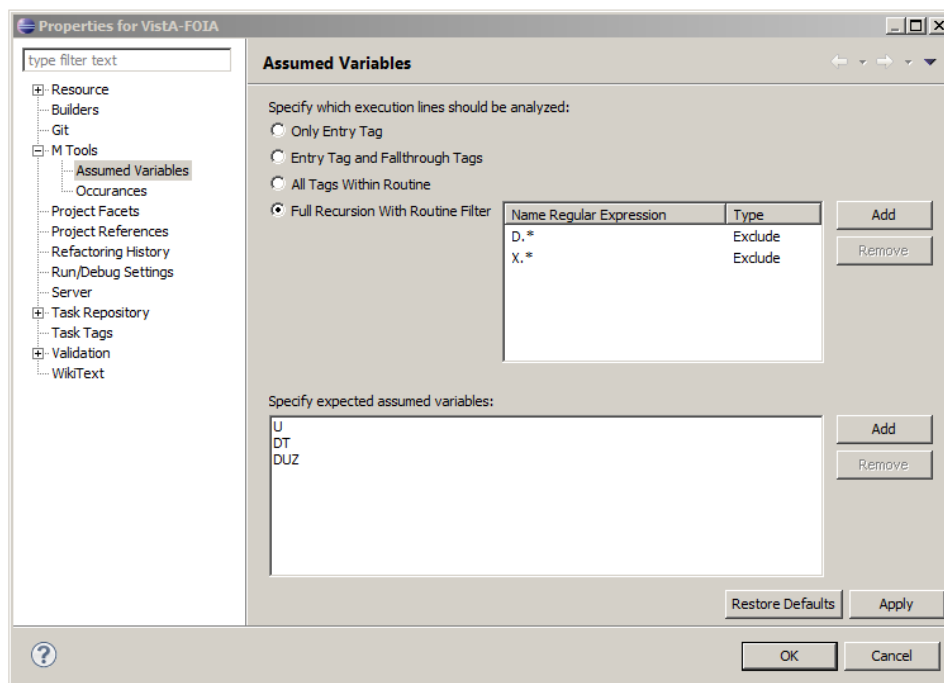
2.6.1. Report Assumed Variables

The tool finds all the variables that are used in the entry tags that are not “newed”. Using no “newed” variables results in un-maintainable code and should be avoided. Running the tool writes the list of assumed variables in the Console per entry point tag. The output also includes clickable location of the assumed variable and when selected it brings the line that the assumed variable is on to focus.

Note that some of these tools do recursive analysis. However, recursive analysis is limited to the routines that are on the client Eclipse project and does not consider routines that might be on the server, but not on the client.



Similar reports are also generated by XINDEX; however, the reports here are generated by a Java-based MUMPS code analyzer that can also recursively search for all the tags that entry tag under test calls. Recursion specifics are configurable, and configuration can be found under Project Properties' M Tools section. (Note: Project Properties is available under the top Project menu item.)



Users can disable recursive analysis by selecting “Only Entry Tag” options. Selecting “Entry Tag and Fallthrough Tags” adds the tags that follow the entry tag to the analysis when entry tag does not end with an unconditional QUIT command. When option “All Tags within Routine” is selected, all the tags which are local to the routine and called by DO or GOTO commands and extrinsic functions are also included in the analysis. Full recursion is available with “Full Recursion with Routine Filter” option where can include all DO and GOT calls and extrinsic functions in the analysis. Users can specify routines that would be included or excluded to limit the extent of the recursion. The routines are identified by name regular expressions.

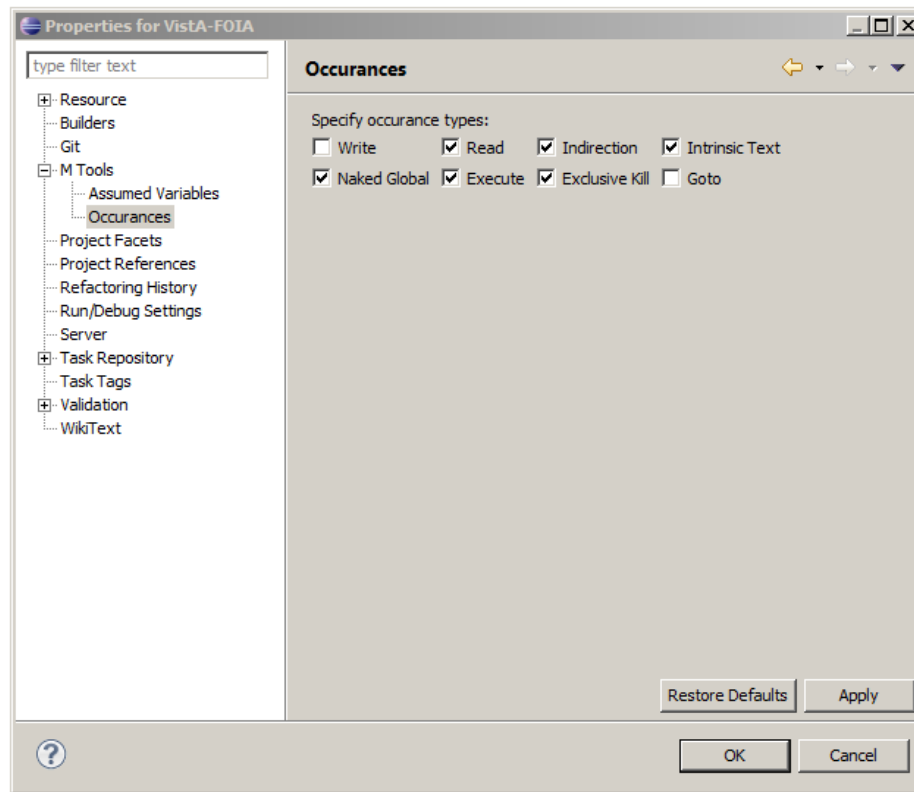
In addition to the setting that configures the recursive analysis, users can also specify the variables that should be excluded from the analysis. This is useful when there are variables that are external by design.

2.6.2. Report Errors

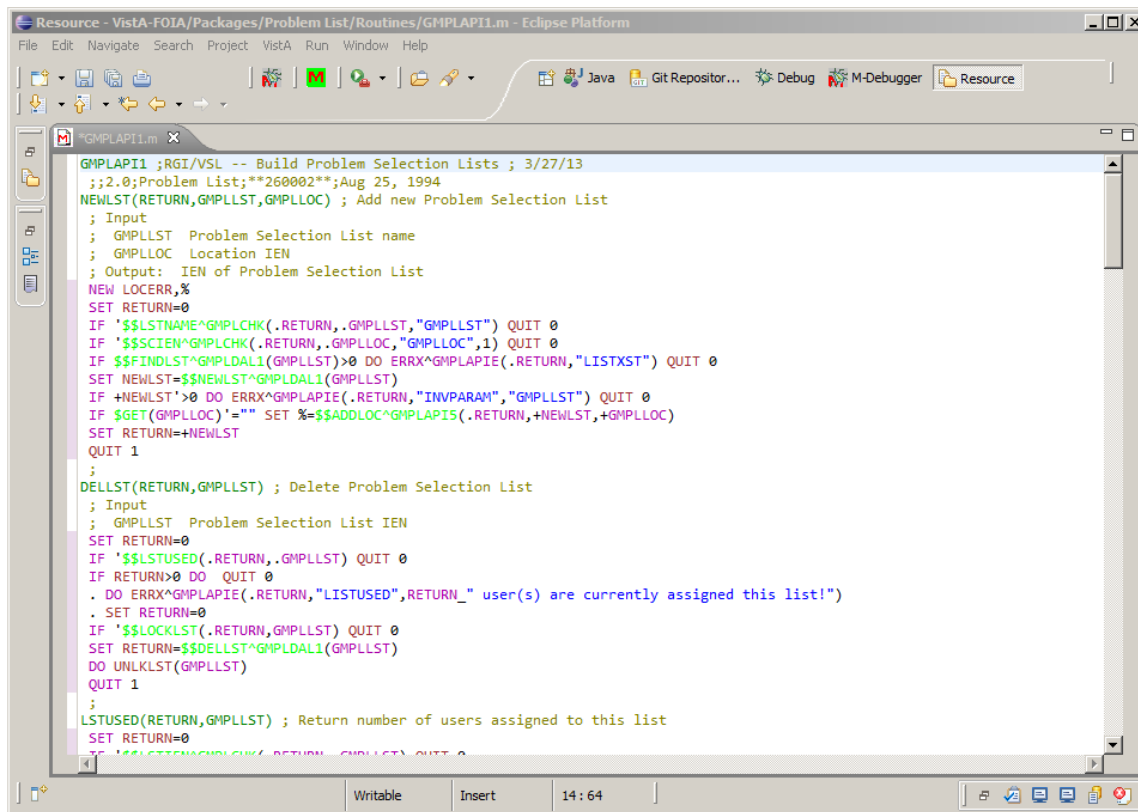
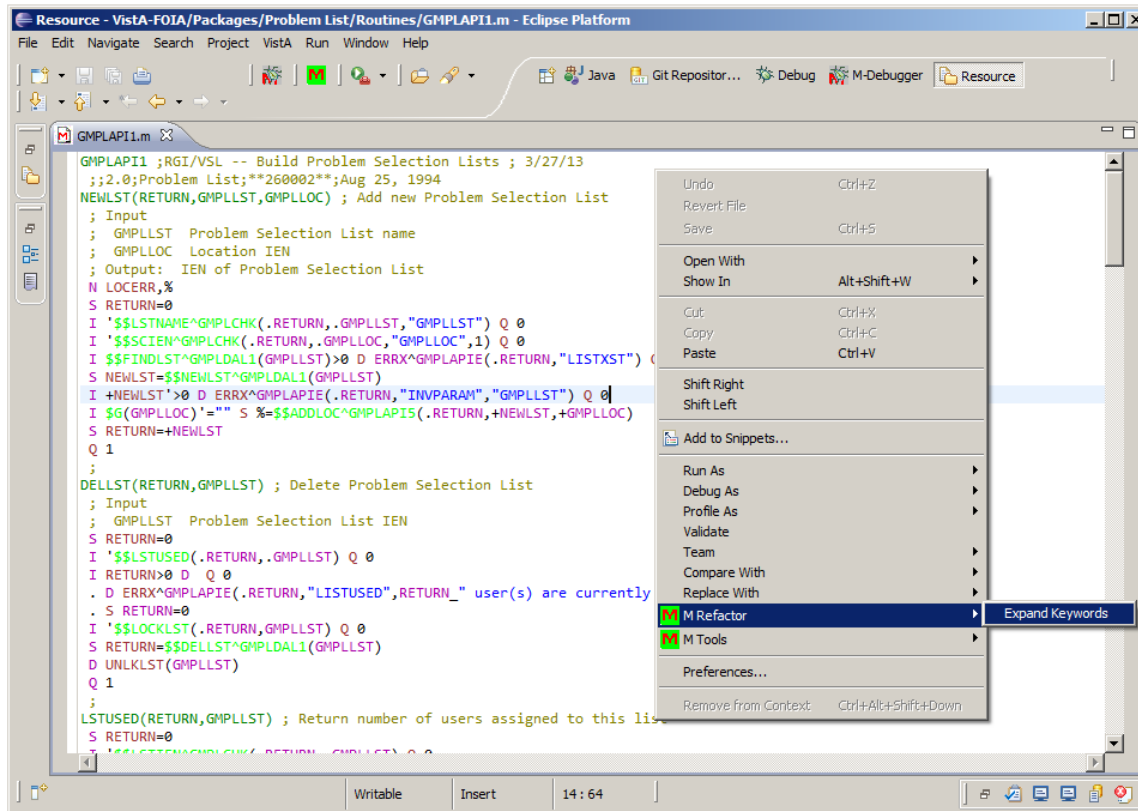
“Report Errors” writes out all the syntactical errors to the Console. The validation is independent from XINDEX validation and implemented on Java. It serves as confirmation of XINDEX results or can be used offline validation tool where users do not need to save the routine to the server to get validated.

2.6.3. Report Occurrences

Report Occurrences writes a number of MUMPS language constructs used in the routines to the Console. Most of these constructs should be avoided in VistA development and this report identifies the location which can be reviewed easily. Users can configure the constructs to report on from Project Properties’ M Tools section.



The following shows an example run; all the constructs include clickable locations.



Approval Signatures

This section is used to document the initial approval of the draft Open Source Electronic Health Record (EHR) Services MTools Installation and Usage Guide, and any subsequent updates and modifications; 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