

**Department of Veterans Affairs**

***Open Source Electronic Health Record Services***

**MTools Installation and Usage Guide**



**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

<b>1.</b>	<b>Installation .....</b>	<b>4</b>
1.1.	Prerequisites.....	4
1.2.	Download Latest Files.....	4
1.3.	Install KIDS Packages and Update VistALink .....	4
1.4.	Run VistALink Job .....	4
1.5.	Install the Plug-in .....	4
1.6.	Configure the Plug-in .....	6
<b>2.</b>	<b>Basic Configuration .....</b>	<b>6</b>
2.1.	Server Configuration .....	6
2.2.	Eclipse Projects .....	7
2.2.1.	Creating a New Project.....	7
2.2.2.	Opening an Existing Project.....	7
2.2.3.	Project VistA Properties.....	8
2.3.	VistA Perspective.....	9
2.4.	Connecting To and Disconnecting From Servers .....	10
<b>3.</b>	<b>Using MEditor.....</b>	<b>11</b>
3.1.	MEditor Preferences.....	11
3.2.	Loading Routines from Server.....	12
3.2.1.	Load M Routine Toolbar Button.....	12
3.2.2.	Routine Load Project Properties .....	15
3.2.3.	Backup Files .....	17
3.2.4.	MEditor Context Menu.....	17
3.2.5.	Project Explorer M Server Context Menu.....	18
3.2.6.	Load Routines.....	18
3.2.6.1.	Load Routine into Folder .....	18
3.2.6.2.	Load Namespace into Folder .....	19
3.3.	Saving Routines .....	19
3.3.1.	MEditor Save.....	19
3.3.2.	MEditor M Server Context Menu.....	20
3.3.3.	Save Routine.....	20
3.3.3.1.	Save Routine to Selected Server .....	20
3.3.4.	Project Explorer M Server Context Menu.....	20

3.3.5.	Save Routines.....	20
3.3.5.1.	Save Namespace .....	20
<b>4.</b>	<b>Using MDebug .....</b>	<b>21</b>
4.1.	Choosing Implementation .....	21
4.1.1.	Generic Implementation .....	21
4.1.2.	Caché Telnet Implementation.....	22
4.1.3.	GT.M Secure Shell Implementation .....	23
4.2.	Debug Perspective.....	24
4.3.	Adding Breakpoints.....	25
4.3.1.	Line Breakpoints .....	25
4.3.2.	Tag Breakpoints .....	26
4.3.3.	Watchpoints .....	27
4.4.	Starting a Debug Session .....	27
4.5.	Variables.....	28
4.6.	MUMPS Terminals.....	29
4.6.1.	Generic Implementation Interactive Console .....	29
4.6.2.	Caché Telnet and GT.M SSH Terminal View and Message Console ..	30
<b>5.</b>	<b>New VistA File Wizard.....</b>	<b>32</b>
<b>6.</b>	<b>MTools Context Menu Tools.....</b>	<b>33</b>
6.1.	Report Assumed Variables.....	35
6.2.	Report Errors.....	37
6.3.	Report Occurrences .....	37
6.4.	Report Quit Types.....	39
<b>7.</b>	<b>M Refactor Context Menu Items .....</b>	<b>39</b>
<b>8.</b>	<b>VistA Main Menu Utilities.....</b>	<b>40</b>
8.1.	Global Directory .....	41
8.2.	Global Listing .....	41
8.3.	Routine Listing .....	42

## 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 <a href="#">here</a> . VistA installation instructions are available on the Open Source Electronic Health Record Agent (OSEHRA) site <a href="#">here</a> .
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: MDebug, MEditor, and Utilities. KIDS installation instructions can be found [here](#).

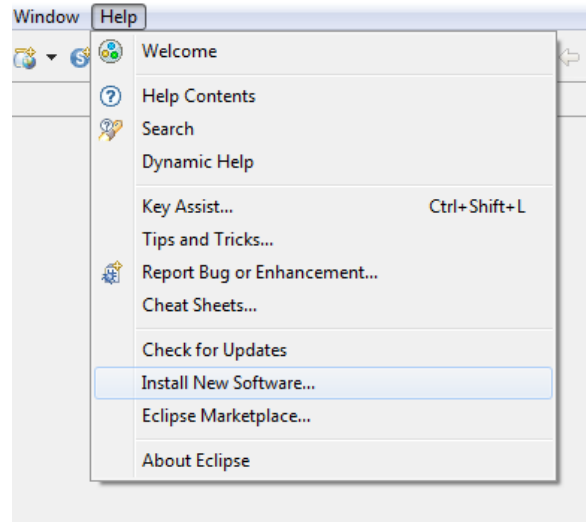
If a user is working in Linux, VistALink routines need to be updated with those found [here](#); these files can be loaded using *D ^%RI* utility from a MUMPS Terminal.

### 1.4. Run VistALink Job

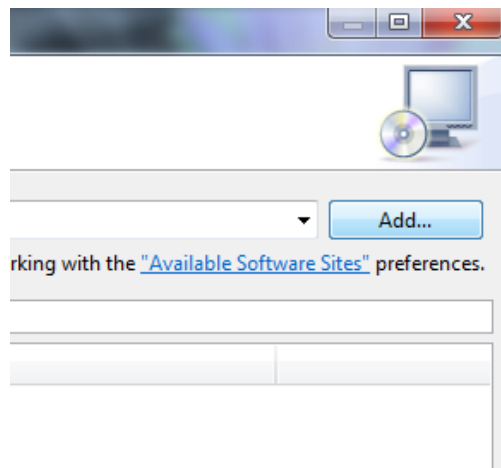
Enter the MUMPS command `JOB LISTENER^XOBVTCPL(8001)` to start VistALink. This action is required for the plug-in to connect with and talk to the server. Here 8001 refers to the port number and you can specify other ports as well.

### 1.5. Install the Plug-in

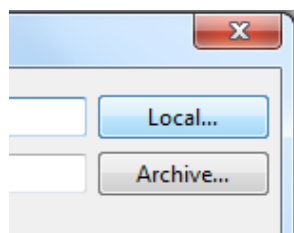
- 1) Open Eclipse
- 2) Click 'Help' → 'Install New Software'



3) Then click 'Add' in the top right



4) Select the 'Local' button in 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'

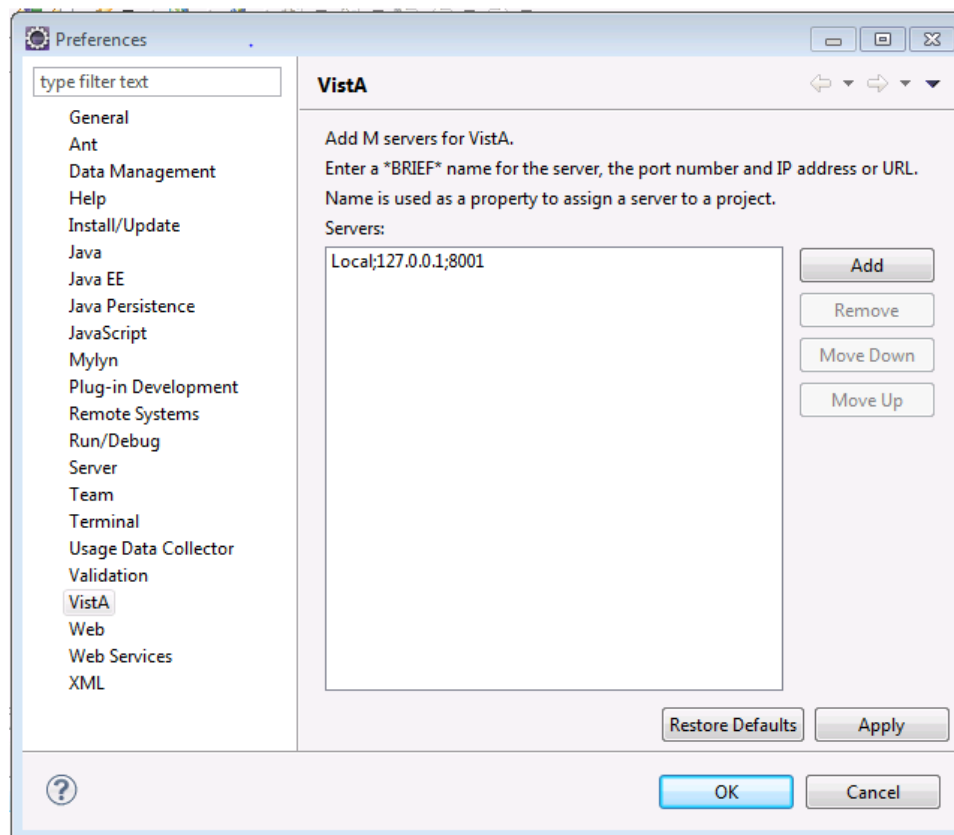
## 2. Basic Configuration

### 2.1. Server Configuration

M Server that will be used with MTools IDE is specified in Preferences.

- 1) From Eclipse's main menu (the top most bar) click Window → Preferences
- 2) Select VistA from the list on the left
- 3) Add a new server using add "Add" button. You need to enter a name, an IP address or hostname and a port for the server; a typical value would be `local;127.0.0.1;8001` for local development.
- 4) Click 'OK' to save the server

You can specify multiple servers, and unlike the previous version of MTools there is no concept of a primary server. How servers are used will be described in later sections.



## 2.2. Eclipse Projects

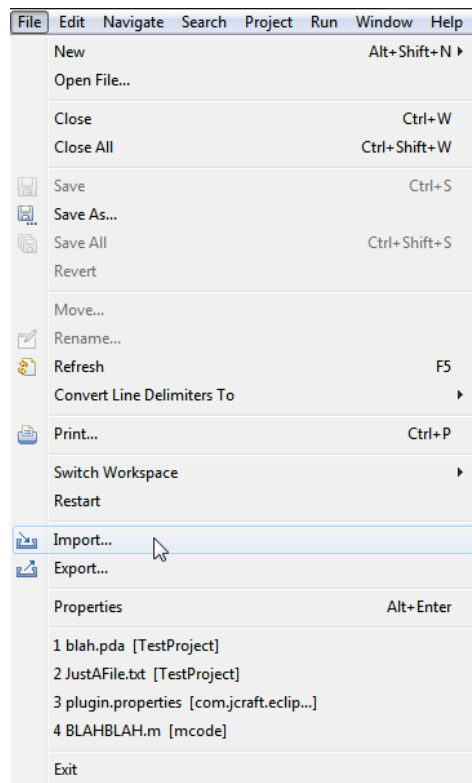
All files in Eclipse must belong to a project since Eclipse can be described as project centric. To begin using MTools IDE there are two options: (1) a new project can be created and files imported, or (2) an existing project can be opened containing existing files and the capability to import new files.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

### 2.2.1. Creating a New Project

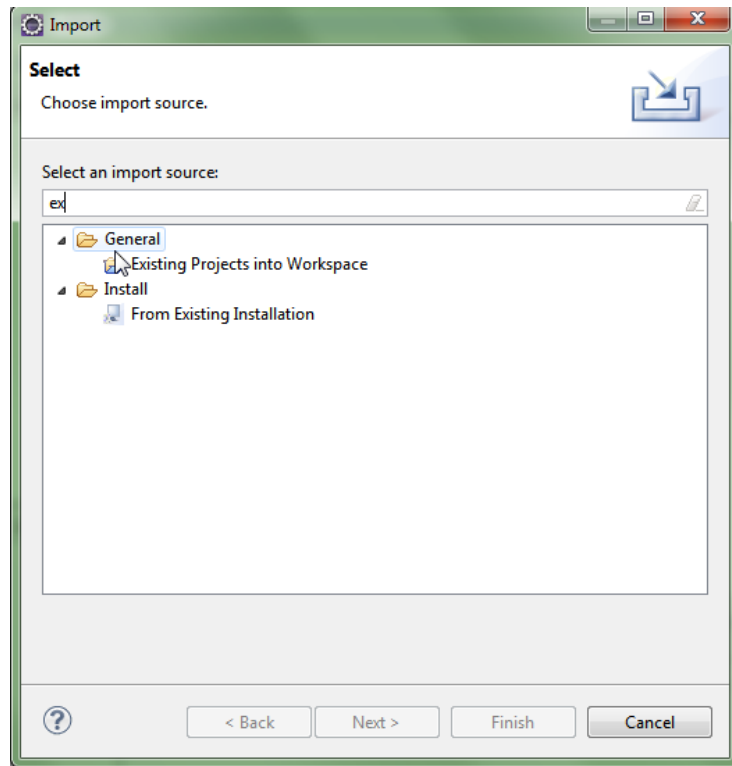
A standard Eclipse project can be created via 'New'→'New Project...'. This opens the new project wizard. The standard Eclipse project can be chosen from 'General'→'Project'.

### 2.2.2. 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.

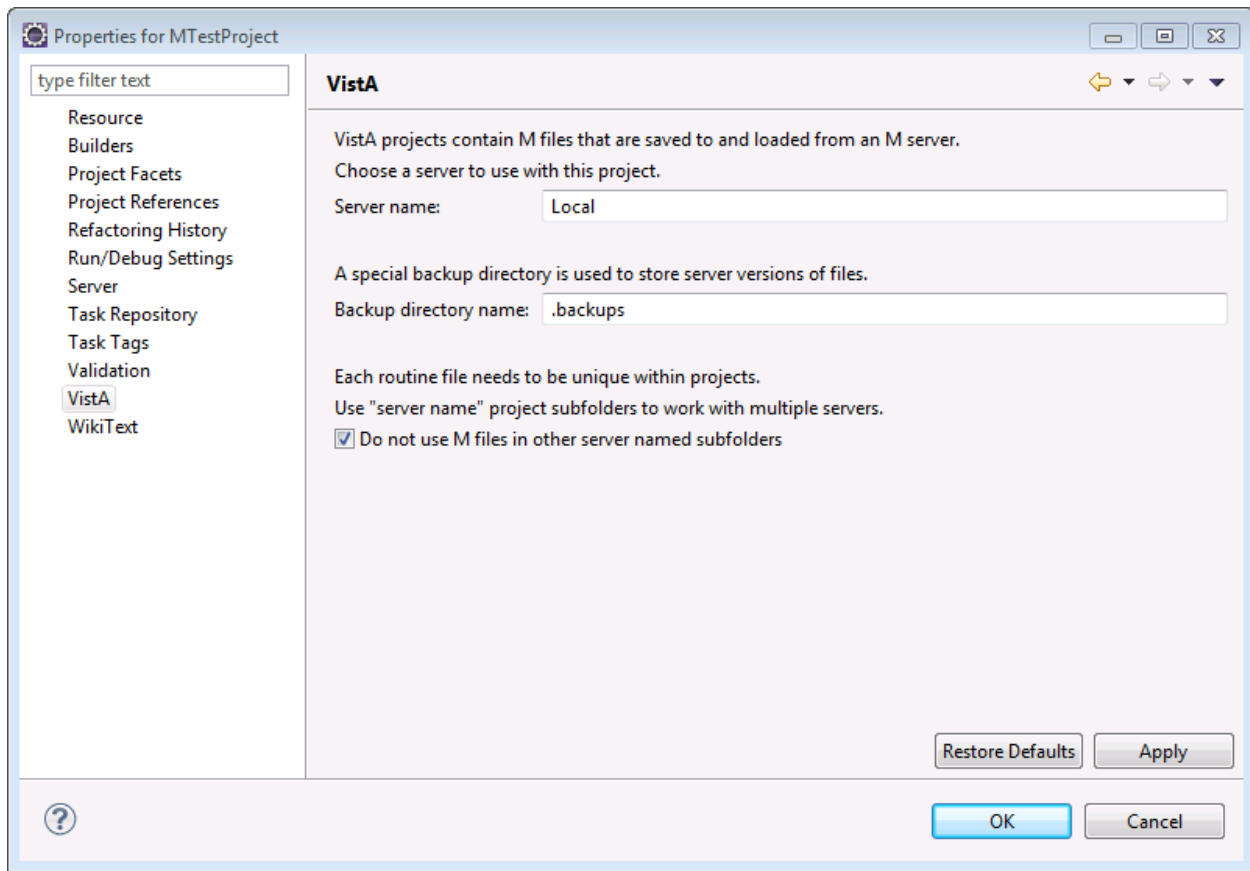


This can be filtered easily by typing 'ex'. Using the file dialog navigate to the directory which contains an existing project (a directory containing a .project file). This will typically be from a version controlled system such as Git, which will typically contain a parent root directory that has the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

### 2.2.3. Project VistA Properties

To be able to load and save files from a MUMPS server a project must be assigned a server. This assignment is done in VistA properties of a project. Project properties can be accessed from Eclipse's main menu (the top most bar, click 'Project' → 'Properties') or from the project context menu.





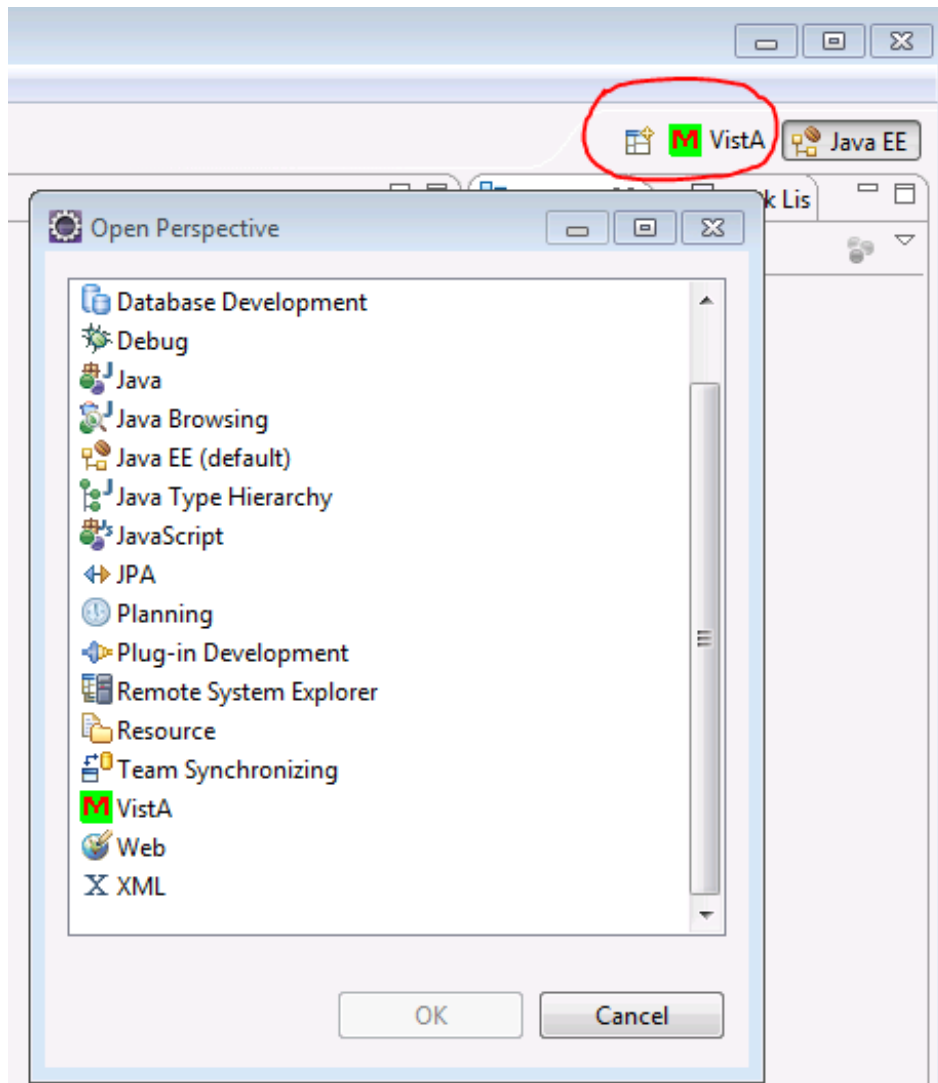
VistA projects are assigned to a server by entering the name in the properties. This server is the primary one used for loading and saving server routines. Loading and saving routines and the other two settings in this property are described later in this document.

## 2.3. VistA Perspective

MTools adds a new VistA perspective to Eclipse. When this perspective is selected on the main menu, toolbar and context menus are updated to have VistA related menu items. You can select VistA perspective from 'Window' → 'Open Perspective' → 'Other' on the main menu or via the 'Open Perspective' button on upper right corner. For editor functionality the rest of this document assumes that you are always in VistA perspective.

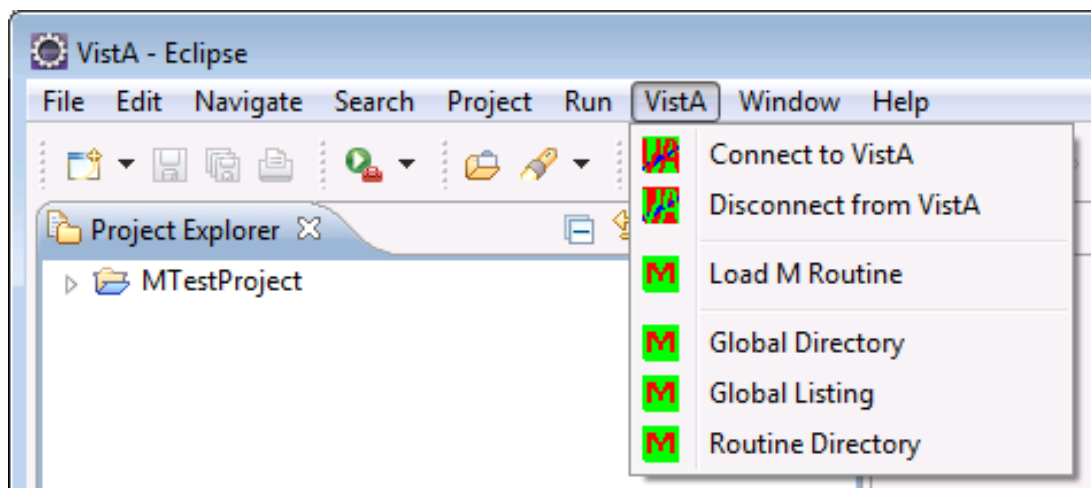
VistA Perspective adds a VistA menu to the main menu and a 'Load M Routine' button to the toolbar. All the other additions are in context menus, and most menu items have the M icon.

By default, VistA perspective uses Project Explorer and Outline view for context functionality and Console view for output.



## 2.4. Connecting To and Disconnecting From Servers

Under the VistA main menu you can find 'Connect to VistA' and 'Disconnect from VistA' menu items for servers specified in the VistA preferences. Note that most of the functionality described in this document prompts a user to connect to a server if the connection has not been established, so for most workflows you do not need to use these menu items directly. One notable exception is for MDebug where the 'Connect to VistA' menu item must be used.

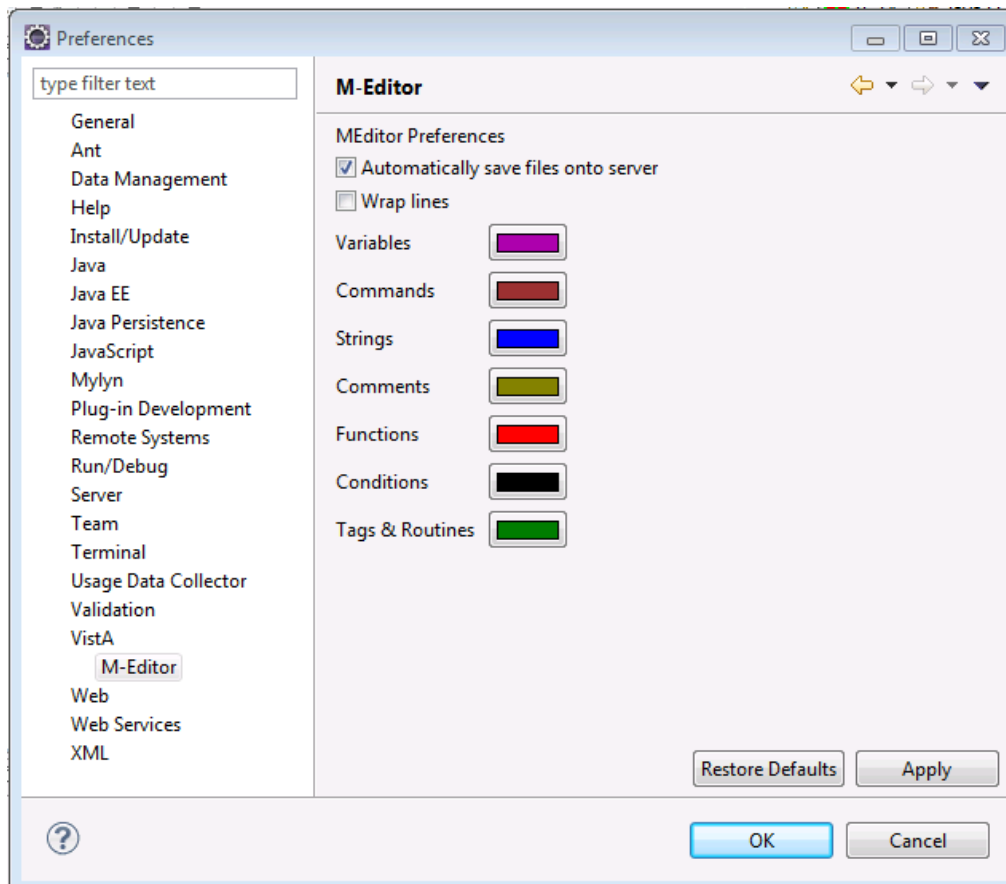


### 3. Using MEditor

MEditor is a custom editor for MUMPS language and part of MTools IDE. To begin using MEditor 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 lower left pane) will display tags in the current routine selected in the editor.

#### 3.1. MEditor Preferences

The colors that are used in MEditor can be configured in VistA preferences under 'Window' → 'Preferences' → 'VistA' → 'MEditor'.

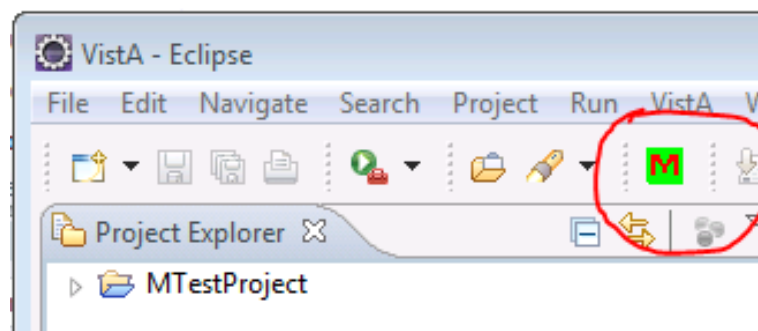


In addition to colors you can also select if lines in the editor is wrapped or not from the VistA preferences.

## 3.2. Loading Routines from Server

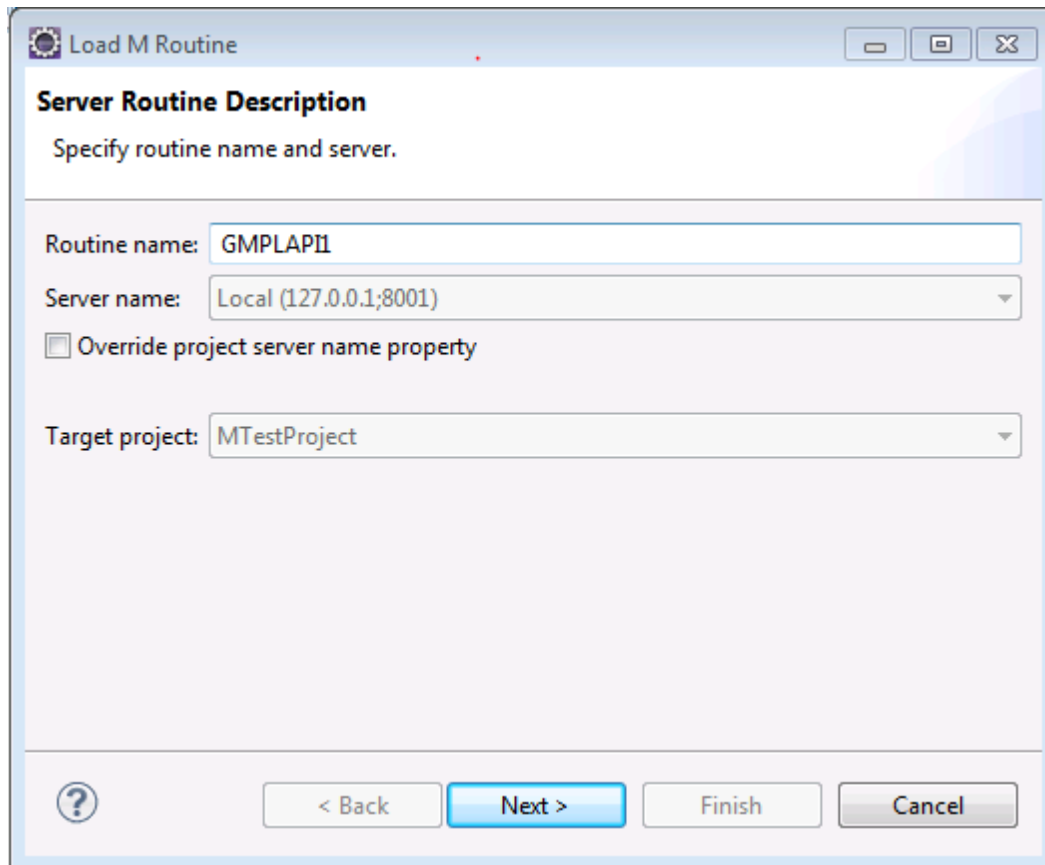
### 3.2.1. Load M Routine Toolbar Button

The most generic way routines can be loaded from a MUMPS server is the green 'M' button in the Eclipse toolbar. This button is not specific to any project and a wizard is launched when it is selected. 'VistA' → 'Load M Routine' menu item on the top VistA menu is identical in functionality to this button.

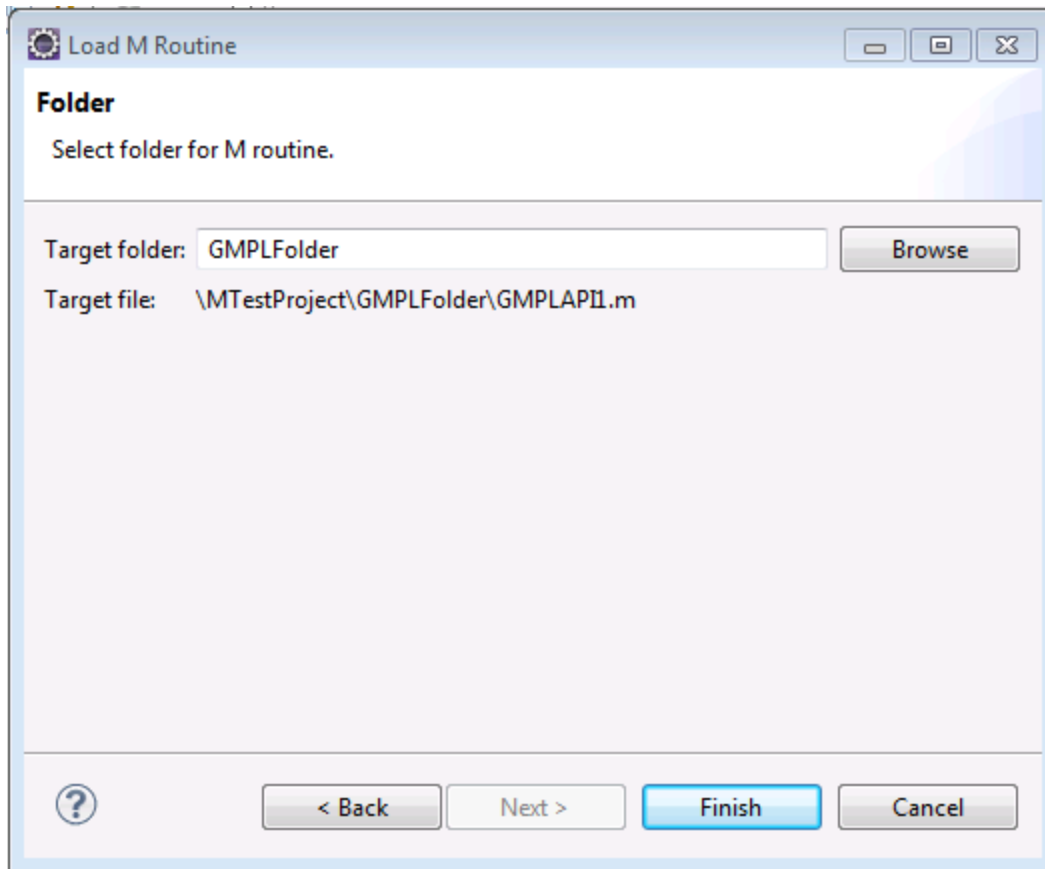


In the first page of the wizard you specify the routine name, the server name where the routine will be loaded from, and project name where the routine MUMPS file will be saved to on client.

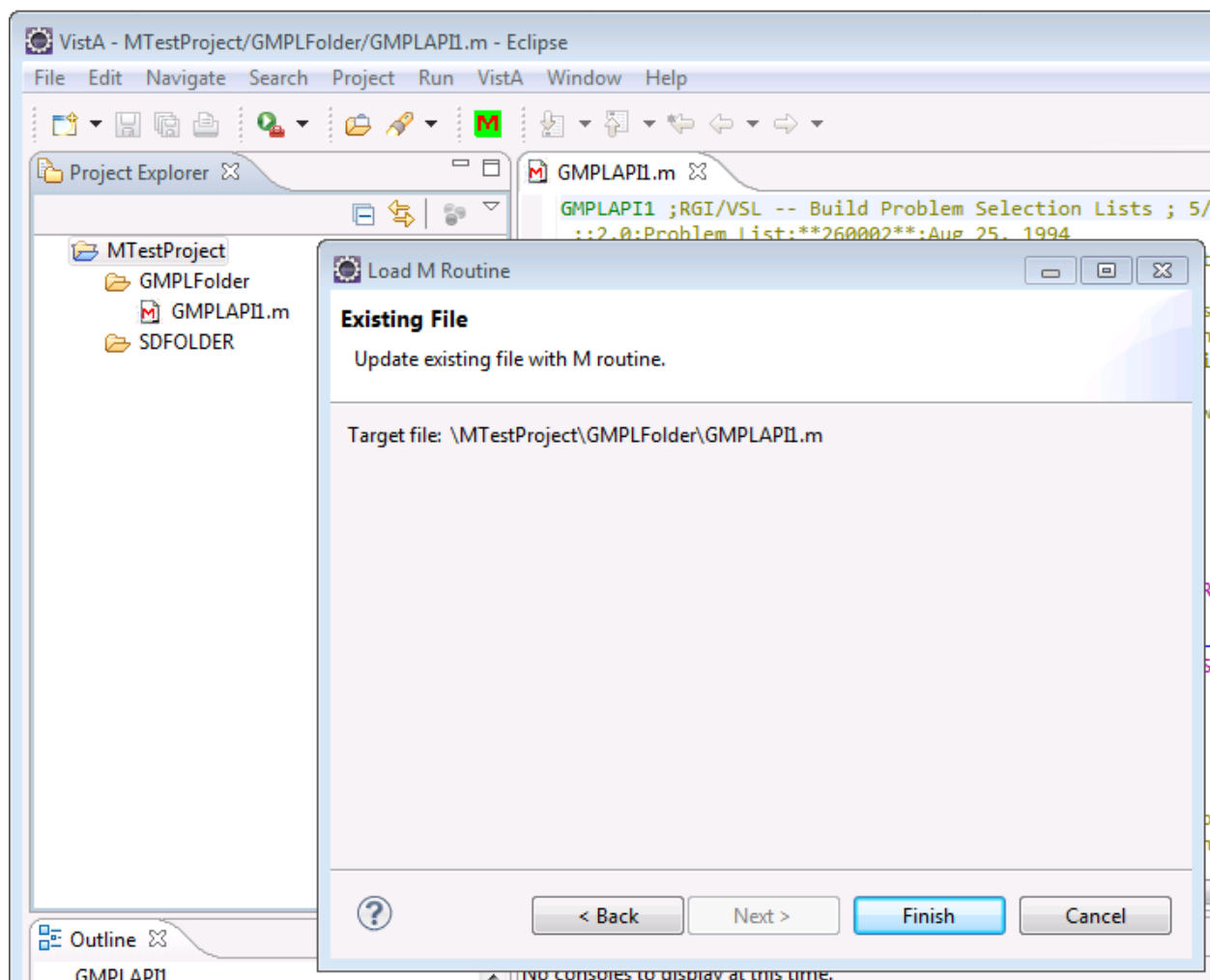
When possible both the server name and the project name are defaulted. By default when a server is selected only the projects that are configured to use the selected server are shown in drop down selection list. You can override the filtering by checking “Override project server name property”. In this case all projects are shown; this allows you to load routines into the project from secondary servers that are not assigned to the project.



Unlike the previous version of MTools IDE, this version supports arbitrary hierarchical directory structures in a project. Thus, by default, users can choose the target folder on the next page of the wizard. An existing folder under the selected project’s root directory must be selected; you can also type in the folder you want but the folder name must be relative. Currently only existing folders are supported. An empty folder is allowed in which case routine is loaded to the root project directory. Full target file path is always displayed in the second page of the Load M Routine wizard. When you hit ‘Finish’ the routine is loaded from the server to the file shown and the file is opened in MEditor.

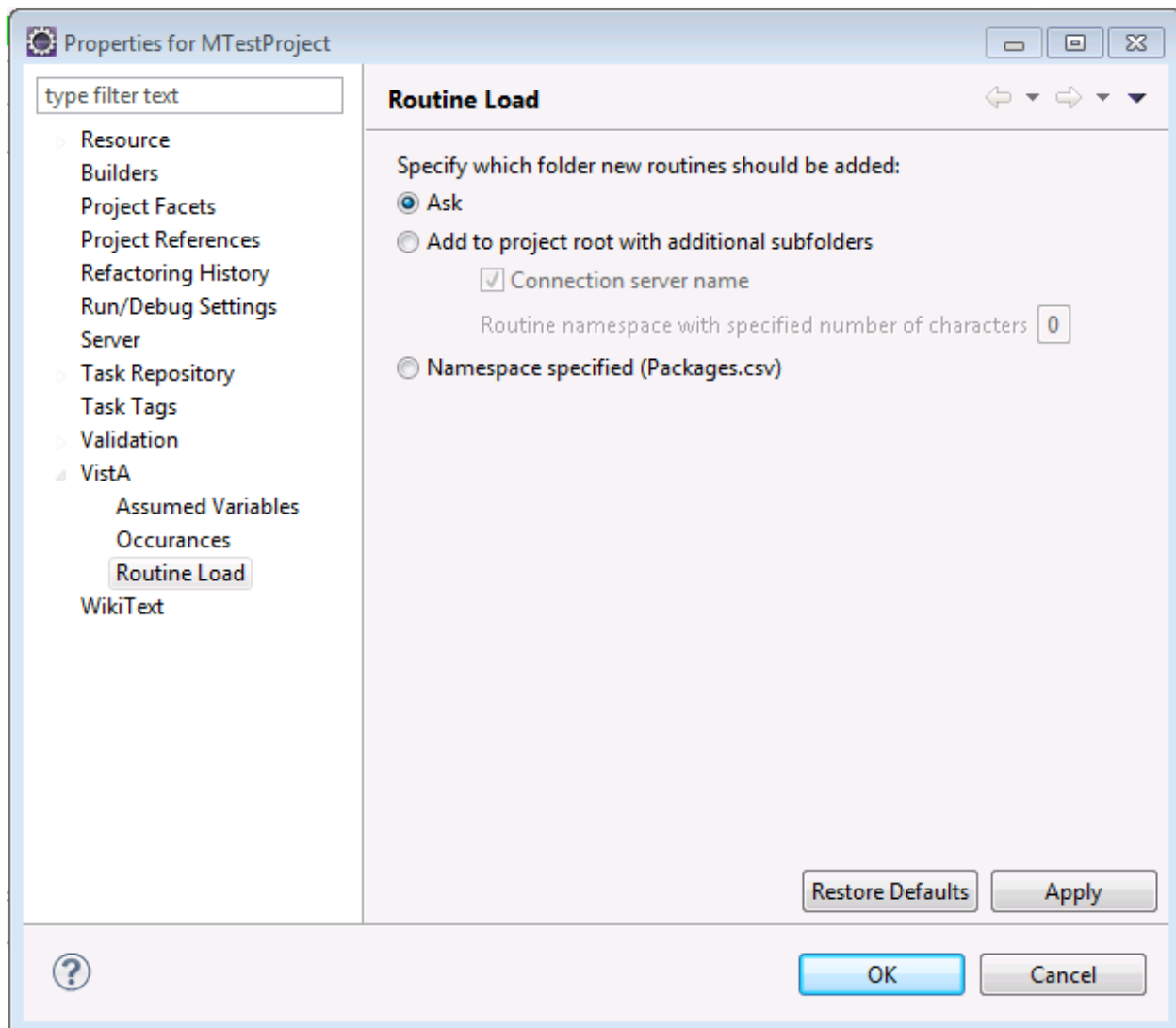


Once the routine is loaded to a particular folder all the subsequent loads from server updates the existing file and user will not be shown any folder selection widget. Various MTools functionality assumes each M file uniquely exists in the project. Load M Routine wizard to enforce this requirement. Currently uniqueness is not enforced on manual file creation. If the selected routine already exists, the second page of the wizard shows the path of the existing file.



### 3.2.2. Routine Load Project Properties

As discussed in the previous section, to support arbitrary hierarchical directories users are asked to enter a folder whenever a new routine is loaded from the server. You can, however, configure MTools to load new routines to particular folders in project properties 'Project' → 'Properties' → 'Vista' → 'Routine Load'.



By default “Ask” is the choice for a folder when a new routine is loaded from the server. You can choose “Add to project root with additional subfolders” to store all routines in a flat directory structure as in the previous version of MTools. The flat directory can be the project root directory or you can add additional subfolders to the project root. Connection server name by default is added as a subfolder under project root. Additionally you can chose to add an additional subfolder based on the namespace of the routines. The number of characters to use for the namespace folders can be configured. Zero means no namespace folder and the maximum is three characters if namespaces are to be used as folders.

Note that MTools assumes each routine uniquely exists in a project. If your workflow requires that you need to work with multiple versions of a routine from multiple servers in the same project you have to choose “Add to project root with additional subfolders” here with “Connection server name” checked. For this case all the server named folders other than the primary server can be ignored for functionality such as Load M File. The “Do not use M file in other server named subfolders” configuration is located in project VistA properties page and is



not checked by default. This configuration is the only support MTools provides to work with multiple servers in the same project.

The final folder specification for new routine location is “Namespace specified”. This setting supports OSEHRA VistA-FOIA directory structure where routine directories are organized according to the packages. With this setting it is assumed that there is a file named Packages.csv in the root directory of the project similar to OSEHRA VistA-FOIA. Packages.csv specifies location of each routine based on its namespace. Note that if Packages.csv does not exist, this setting causes the routines to always be loaded to the project root directory.

### **3.2.3. Backup Files**

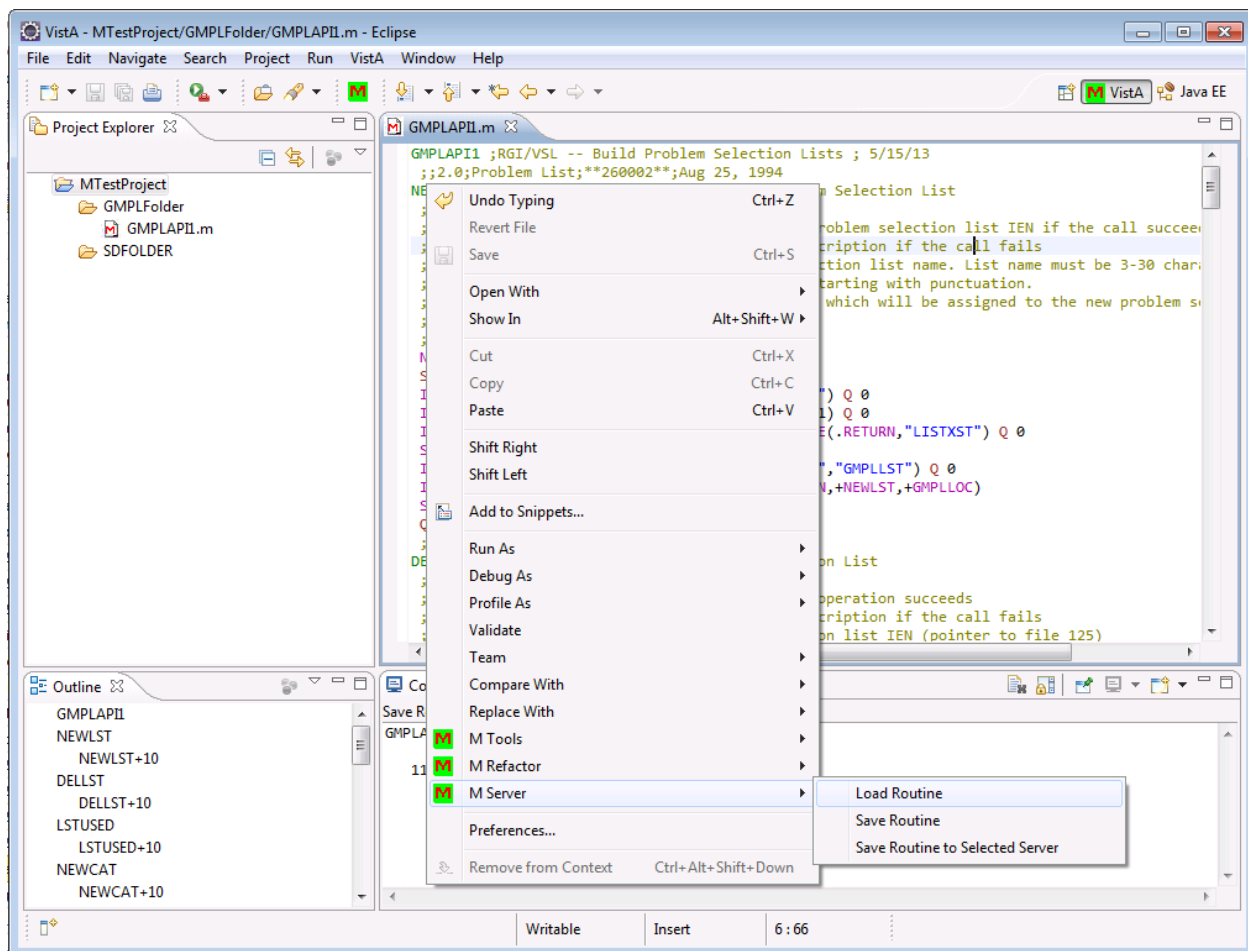
Whenever a file is successfully loaded, a backup file is either created or updated in the backup folder of the project. Each project backup folder is specified in Project VistA properties (‘Project’ → ‘Properties’ → ‘VistA’) and defaults to “.backup”. MTools always use the Server Name that the file is loaded from as an additional subfolder under the backup folder. You can turn off backup feature by specifying an empty backup folder in project VistA properties.

The backup file contains the latest server version locally. The backup file’s relative path with respect to backup folder and server named subfolder is always identical to its original relative path with respect to the project root. Backup files will always have the same name as the original file but with the extension “.mbk”. Users are free to browse the contents of the backup files and MEditor is configured to open “.mbk” files as well. However, it is recommended that these files are not changed manually.

Backup files are primarily used for comparison purposes when a file is loaded from the server and backup/comparison purposes when a file is saved to the server. If MTools detects the server file is changed outside Eclipse the user is alerted to examine the changes. You may choose to include the backup folder in your version control system so that all versions of the backup files are available. Otherwise Eclipse local history can be used to compare versions of backup files. You can configure how long to keep the backups in the local history; please reference Eclipse ‘Help’ for more information on this.

### **3.2.4. MEditor Context Menu**

MEditor’s M Server context menu contains a “Load Routine” submenu item which can be used directly to update the file in the editor from the server. In this case no wizard is used since the file is already selected. Otherwise the functionality is identical to the Load M Routine toolbar button.



### 3.2.5. Project Explorer M Server Context Menu

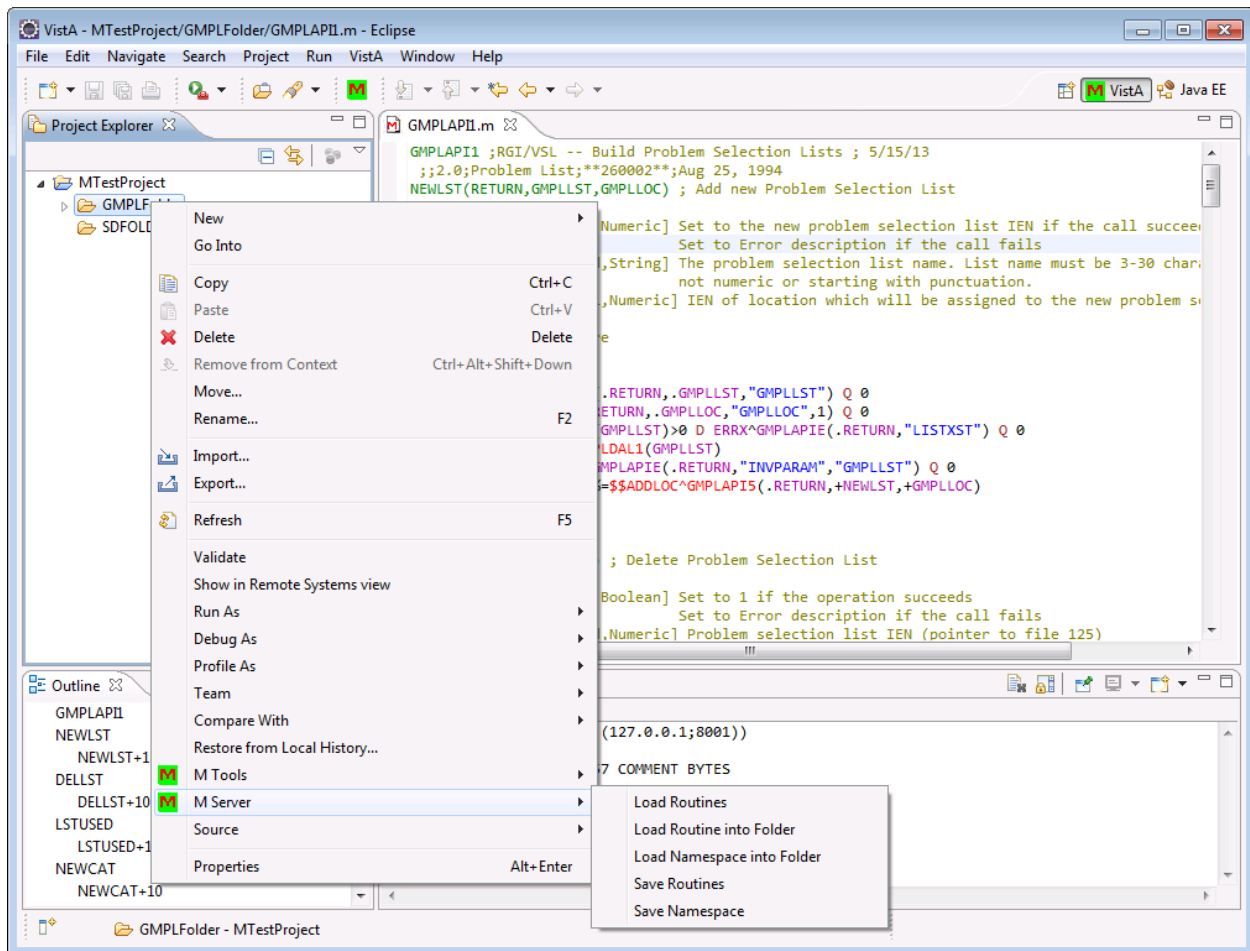
The Project Explorer M Server context menu item can also be used to load a single selected routine from the server; the functionality is also identical to the MEditor Load Routine Context Menu item. In addition, from the Project Explorer M Server Context Menu multiple routines can be loaded at once.

### 3.2.6. Load Routines

The M Server → Load Routines context menu loads the selected routine(s) from the server. This menu is shown only when one or more M routines and/or folders are selected in the explorer. When a folder is selected, all the files under the folder are recursively included in the list to be loaded. Save for how the routine is selected, the functionality is identical to the 'Load M Routine' toolbar button; this essentially repeats the load process for all the selected files.

#### 3.2.6.1. Load Routine into Folder

If a single folder is selected on the Project Explorer, additional load options are available in the Project Explorer M Server context menu item. The first of this loads a single routine into the folder; functionally this is equivalent to the case in Load M Routine wizard when the user can select a folder. Once this menu item is selected, the user is presented with an Input Dialog where they can enter the routine name to be loaded. Please note that if the routine already exists in the project, the existing file is updated even if it is not in the selected folder.



### 3.2.6.2. Load Namespace into Folder

Load Namespace into Folder enables multiple routines to be loaded to the same folder. When this menu item is selected, the user is presented with an Input Dialog where they can specify a namespace. All the routines in the specified namespace are loaded into the project; new routines are saved in the selected folder and existing routines are updated.

## 3.3. Saving Routines

### 3.3.1. MEditor Save

In MEditor routines are saved, just like all Eclipse files, to the disk; additionally, when the 'Save' button is clicked, MEditor will also attempt to save the routine to the server. You can choose to disable this behavior by unchecking 'Automatically save files onto server settings' in VistA preferences ('Window' → 'Preferences' → 'VistA' → 'MEditor'). If automatic save is disabled, users can use context menu Save Routine to save the routine which is discussed shortly.

This version of MEditor always saves the files on client when the user chooses even if the file cannot be saved to the server (for example: when a connection cannot be established). In addition, tabs, control characters, and empty lines are automatically removed upon saving a file.

Control characters and empty lines are invalid in VistA routines and currently MEditor does not support tabs as line-start indicator.

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 to alert users to examine the differences as the difference means that the server file is changed outside MTools. In this case the file will not be immediately saved and users will have to save it again after examining the differences. Unlike the previous version of MTools, no third party compare tool is used in this version. Users can use the Local History or version control system compare facilities. If the file to be saved is identical to the server, no saving is done and an informational dialog will be shown.

Once the routines are saved, XINDEX is also run on the server side and the results are shown in the Console view.

### **3.3.2. MEditor M Server Context Menu**

MEditor M Server context menu item contains ‘Save Routine” and “Save Routine to Selected Server” that can be used to save files to server. Both can be seen in the screenshots shown for Load Routine.

#### **3.3.3. Save Routine**

This saves the routine in MEditor to the server and is functionally equivalent to saving it from MEditor.

##### **3.3.3.1. Save Routine to Selected Server**

This is similar to MEditor, but users can select the server to save the routine to and thus overriding the primary server specified in project properties. This functionality is provided to replace the secondary server saving in the previous version of MTools. This is the only way MTools provides to override the server specified in project properties.

#### **3.3.4. Project Explorer M Server Context Menu**

The Project Explorer M Server context menu item can also be used to save a single selected routine to the server; the functionality is identical to the MEditor Save Routine context menu item. In addition, multiple routines can be saved at once from the Project Explorer context menu item.

#### **3.3.5. Save Routines**

This capability saves all the selected files to the server. If a folder is selected, all the files that are hierarchically under the folder are saved; this functionality is identical to individually selecting files.

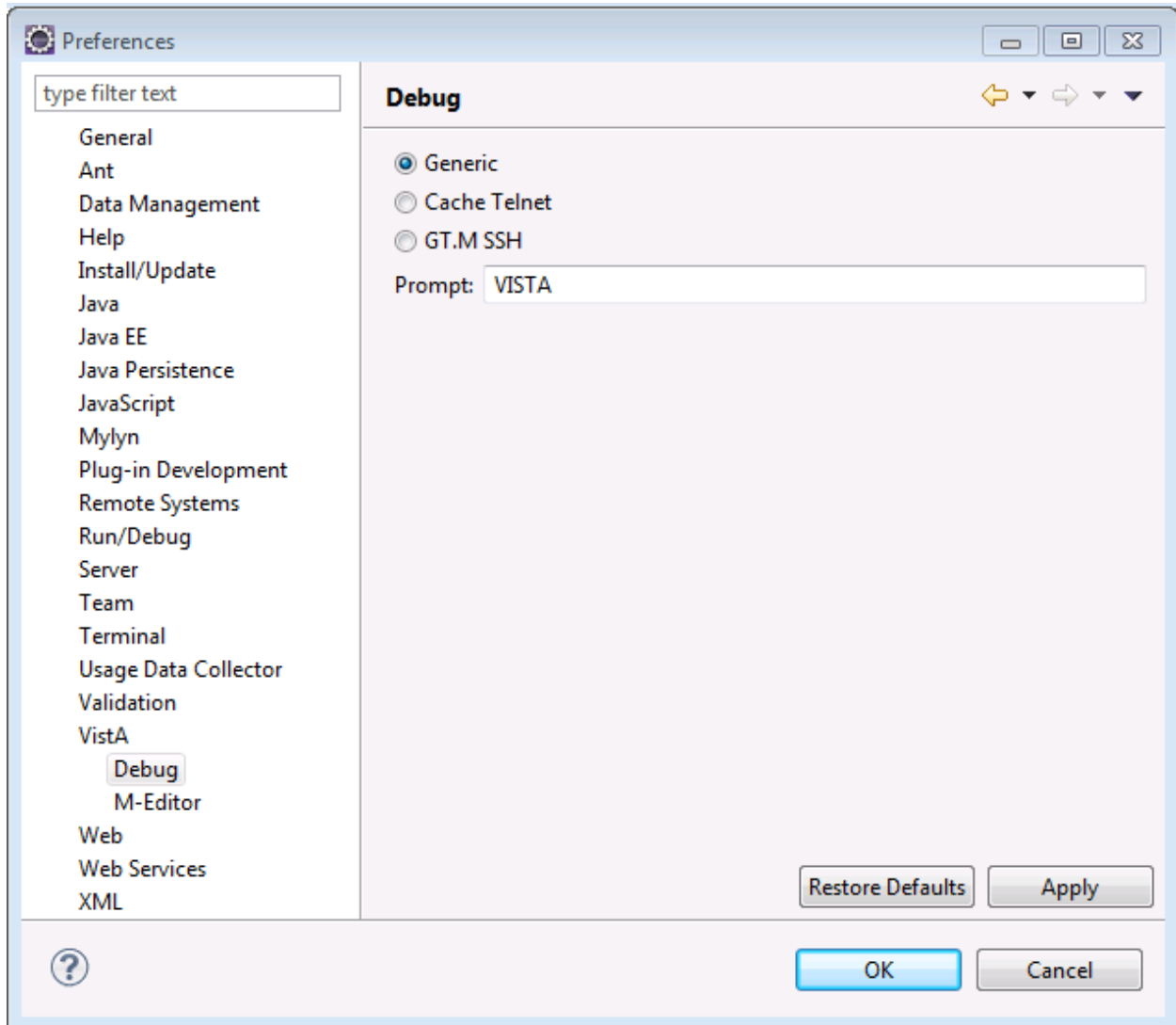
##### **3.3.5.1. Save Namespace**

If there are folders selected in the Project Explorer, then Save Namespace is shown in the context menu. If this item is selected a user can then specify a namespace and all the files that are hierarchically under the selected folders with the specified namespace will be saved; all others will be ignored.

## 4. Using MDebug

### 4.1. Choosing Implementation

MDebug implements three separate debuggers. The implementation to use can be chosen from 'Window' → 'Preferences' → 'VistA' → 'Debug'.



#### 4.1.1. Generic Implementation

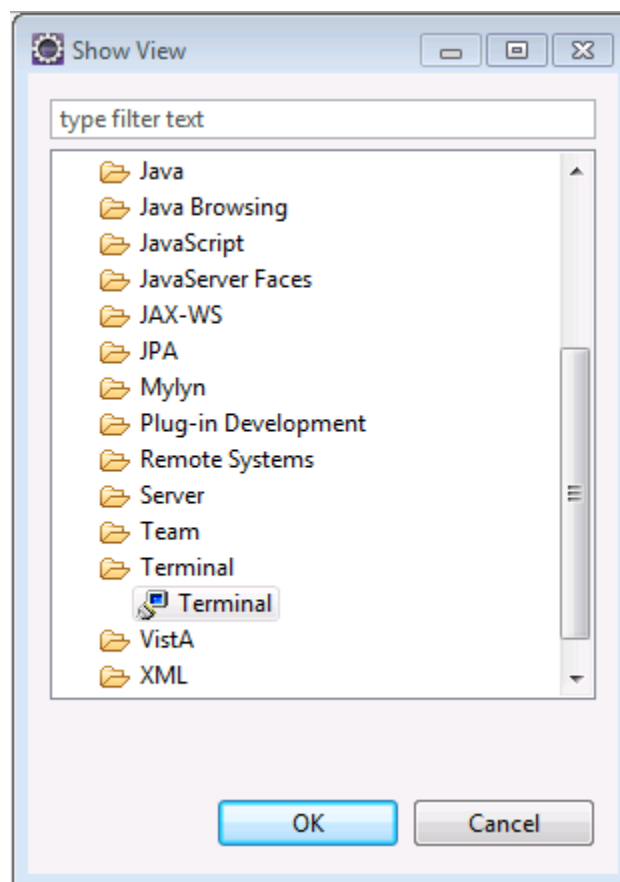
Generic implementation is the improved version of the original MDebugger. This version implements a custom MUMPS interpreter in the back end and is independent from the specific MUMPS implementation. Similar to MEditor it uses VistA Link to communicate with the server. Although this version has the advantage of being the same for GT.M and InterSystems Caché, currently the interpreter does not implement all the MUMPS language constructs. It should be noted that especially in complicated MUMPS code, such as Fileman routines, problems may occur.

Issues may also arise in UI-heavy MUMPS programs like Roll-and-Scroll Interface programmer entry point (^XUP) since terminal reads and writes need special handling; this did not yet mature in the code. Generic Implementation is useful for newer, “silent” API code which does not typically use complicated MUMPS constructs that have not yet been implemented by the back end interpreter.

#### 4.1.2. Caché Telnet Implementation

If you are using InterSystems Caché then you can use the Caché Telnet implementation. This implementation runs an InterSystems Caché terminal in Eclipse and users transparently use ZBREAK, BREAK, and argument less GOTO commands to implement the debugger. Please see Caché documentation for more information and to learn about these commands if you are not familiar with them.

The terminal is based on Eclipse Terminal view available from ‘Window’ → ‘Show View’ → ‘Other’ → ‘Terminal’.

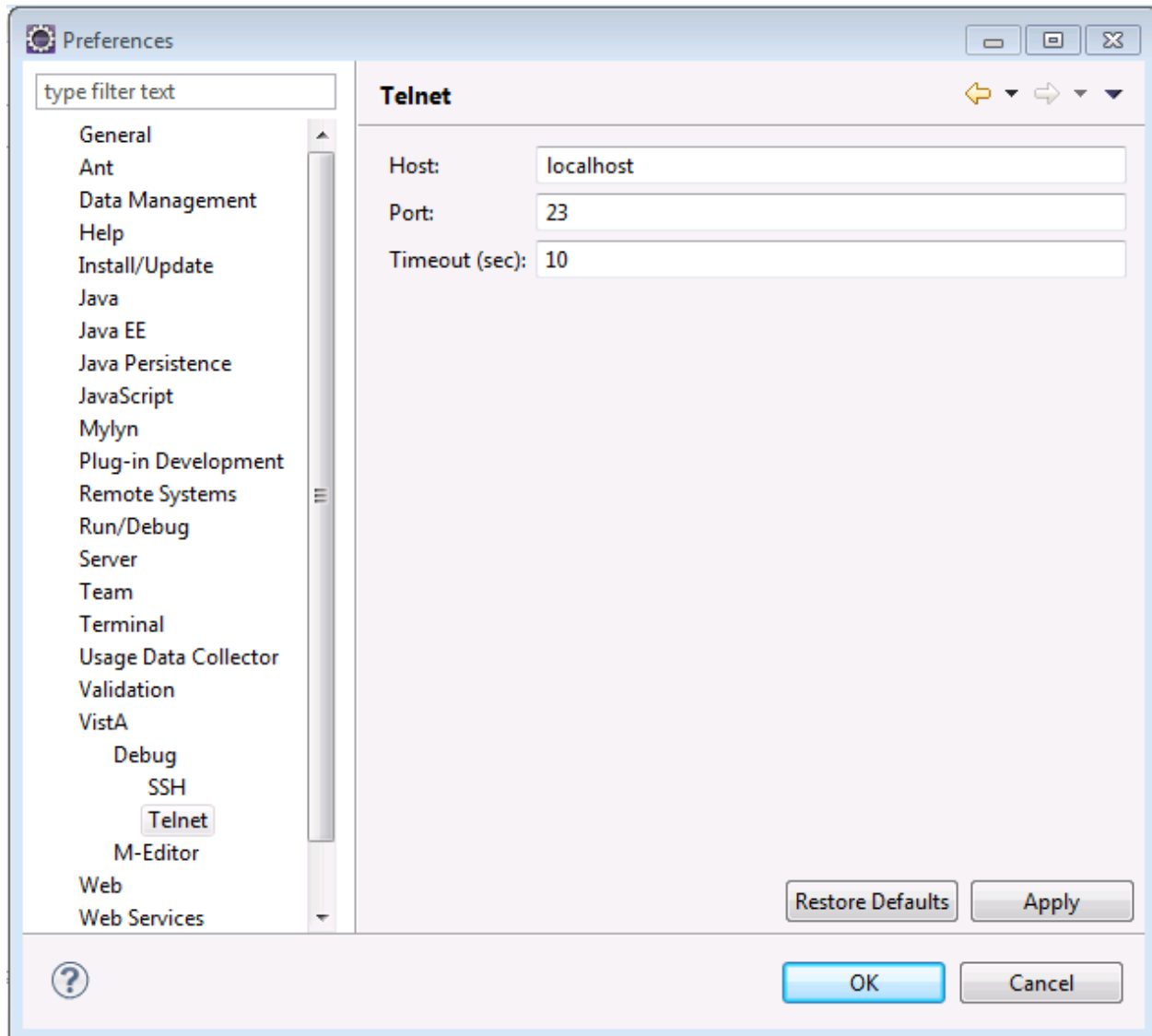


Although users do not utilize this view directly to use MDebug Caché Telnet Implementation, users are advised to launch this view to ensure that it is possible to connect to InterSystems Caché using Telnet within Eclipse or debug any connection issues. If a user cannot connect to InterSystems Caché using this Telnet Terminal, then MDebug Caché Telnet Implementation will not work.

MDebug Caché Telnet Implementation assumes that the user is set up to launch a particular namespace automatically on Telnet connection (please see Caché documentation for more

information on this topic). The implementation manipulates input and output streams of Terminal/Caché communications; specifying the namespace in the “Prompt” field is particularly important as this is how MDebug recognizes Caché is connected, a break occurred, or a ZBREAK command is successfully executed.

Users can specify host or port information in ‘Window’ → ‘Preferences’ → ‘VistA’ → ‘Debug’ → ‘Telnet.’ Default host is the localhost and port is the default port that Caché uses.



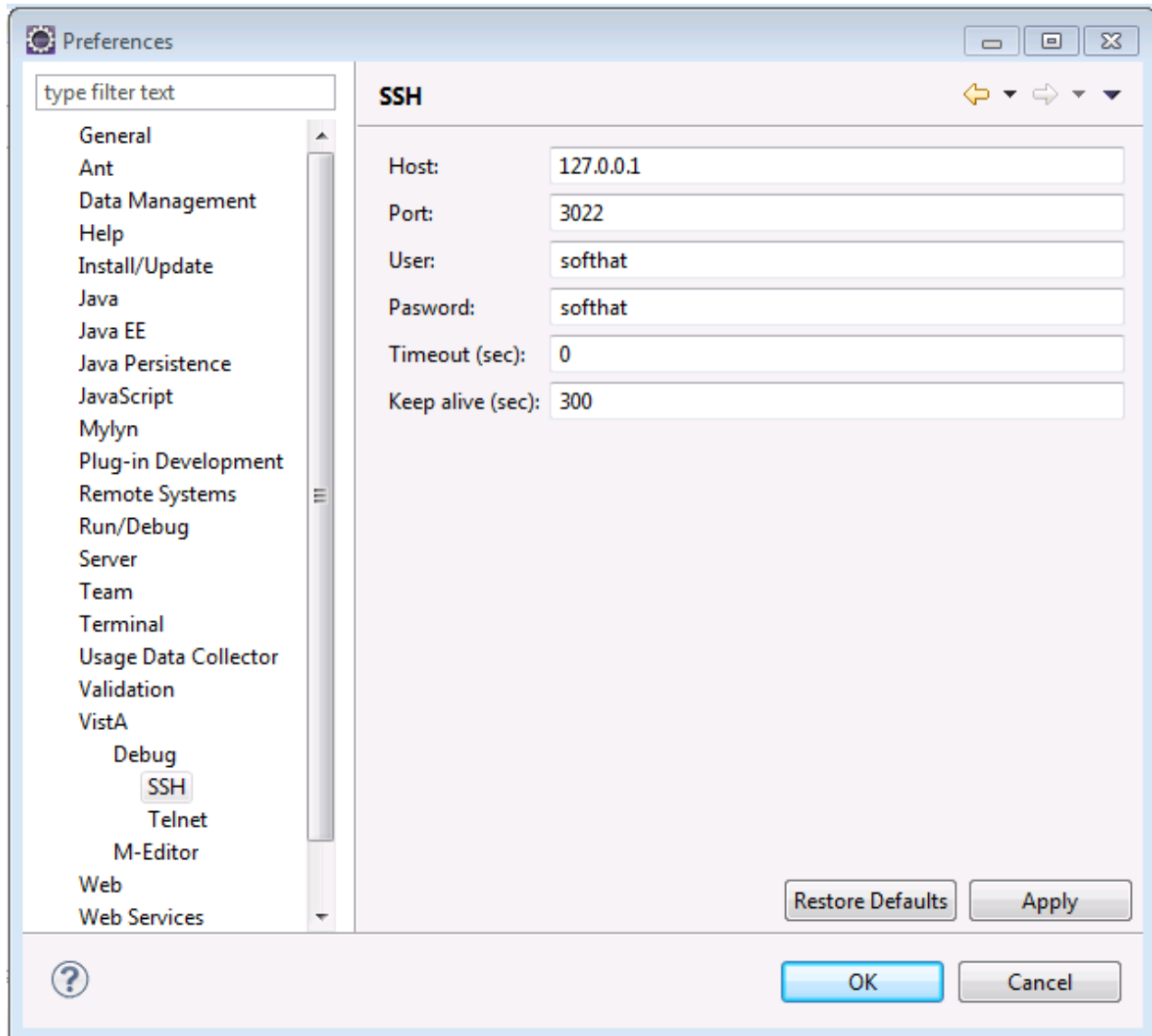
#### 4.1.3. GT.M Secure Shell Implementation

If you are using GT.M then you can use GT.M secure shell (SSH) implementation. This implementation runs a GT.M terminal in Eclipse and users transparently use ZBREAK, ZSTEP, and ZCONTINUE commands to implement the debugger. Please see GT.M documentation for more information and to learn about these commands if you are not familiar with them.

Similar to Caché Telnet terminal, the GT.M SSH terminal is based on Eclipse Terminal view available from ‘Window’ → ‘Show View’ → ‘Other’ → ‘Terminal.’ It is recommended that users

verify they can connect to GT.M using SSH from this terminal; if a user cannot connect to GT.M using this terminal MDebug GT.M SSH implementation will not work. The current version of MDebug assumes Linux shell prompt is "\$" and GTM prompt is "GTM". The prompt that is specified in 'VistA' → 'Debug' → 'Prompt' is not respected. MDebug uses these prompts to recognize debugging event such as connection has been succeeded or a break has occurred; if these prompts are setup to be different, MDebug will not work.

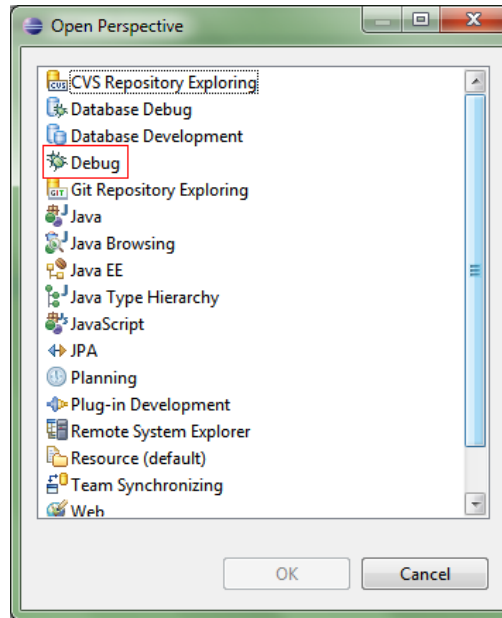
Users can identify the specifics of the SSH connection in 'Window' → 'Preferences' → 'VistA' → 'Debug' → 'SSH'. Defaults are based on the connection from a Windows machine to the OSEHRA VistA virtual machine where guest machine port 22 is forwarded to port 3022.



## 4.2. Debug Perspective

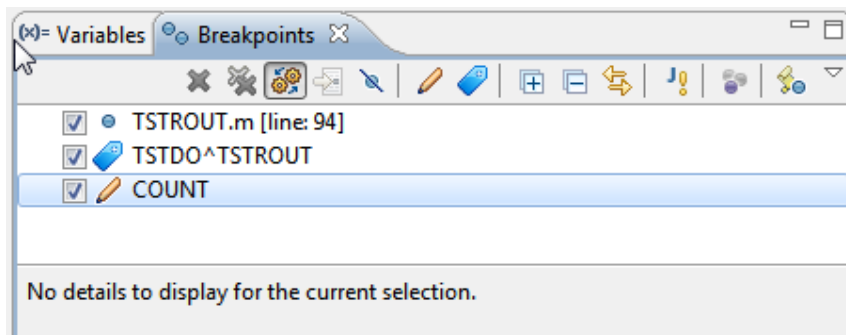
To use the debug features, enter the Eclipse Debug Perspective. Click the Change Perspective menu in the top right of the Eclipse IDE; then choose `Debug`.





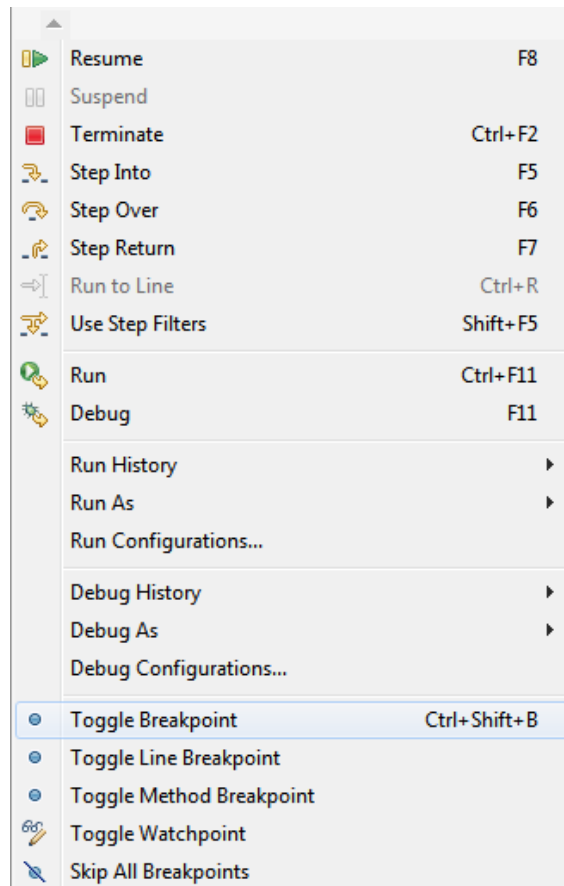
### 4.3. Adding Breakpoints

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.

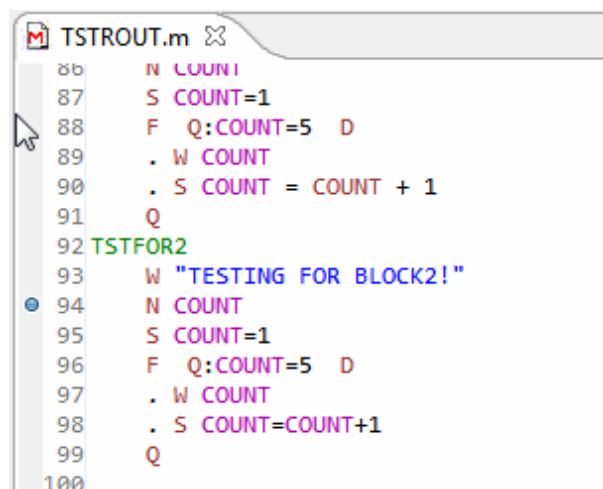


#### 4.3.1. Line Breakpoints

Open the MUMPS 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.



#### 4.3.2. Tag Breakpoints

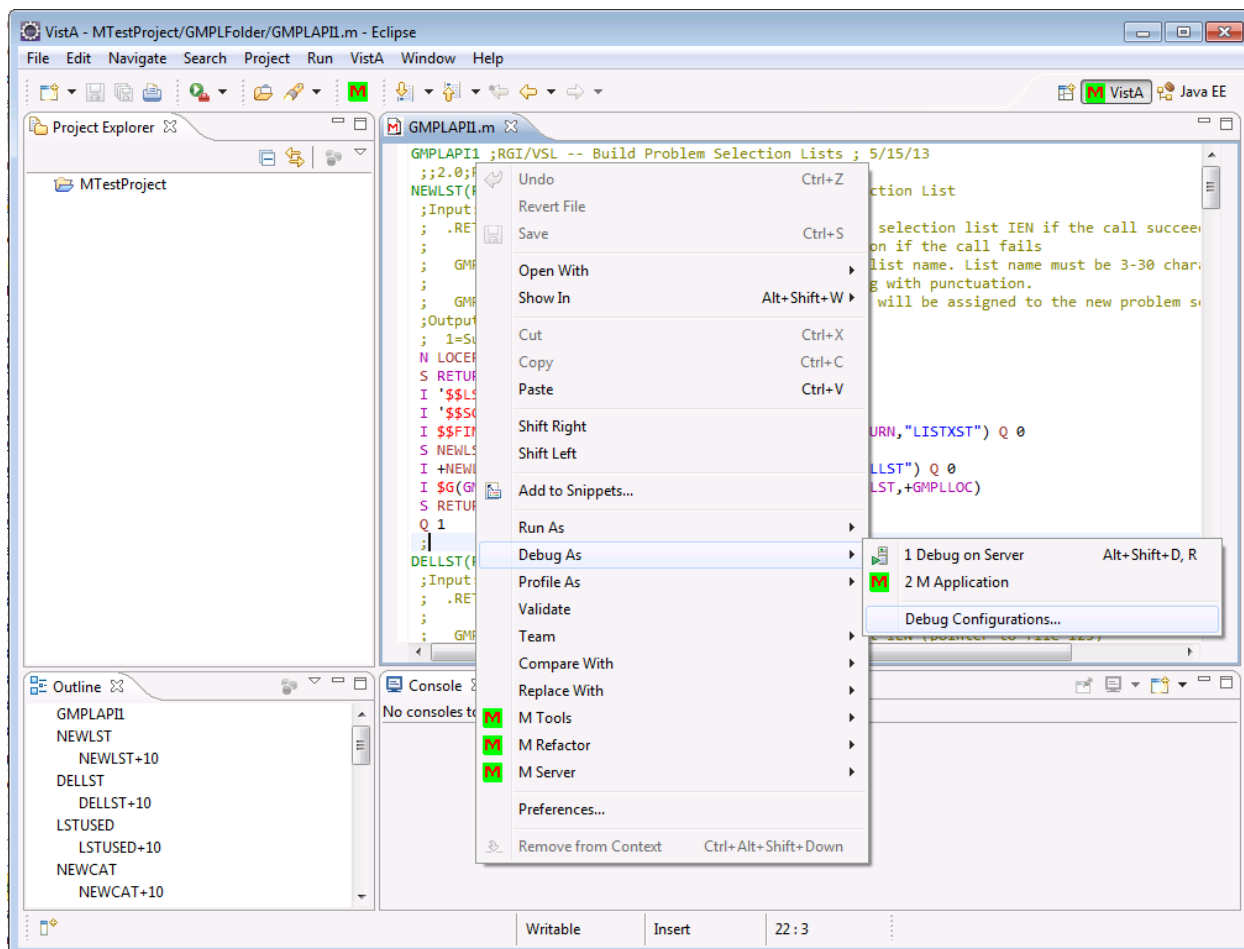
Arbitrary tags can also be entered (e.g.: *TAG+offset^ROUTINE*) by clicking on the blue tag icon from the breakpoints view.

### 4.3.3. Watchpoints

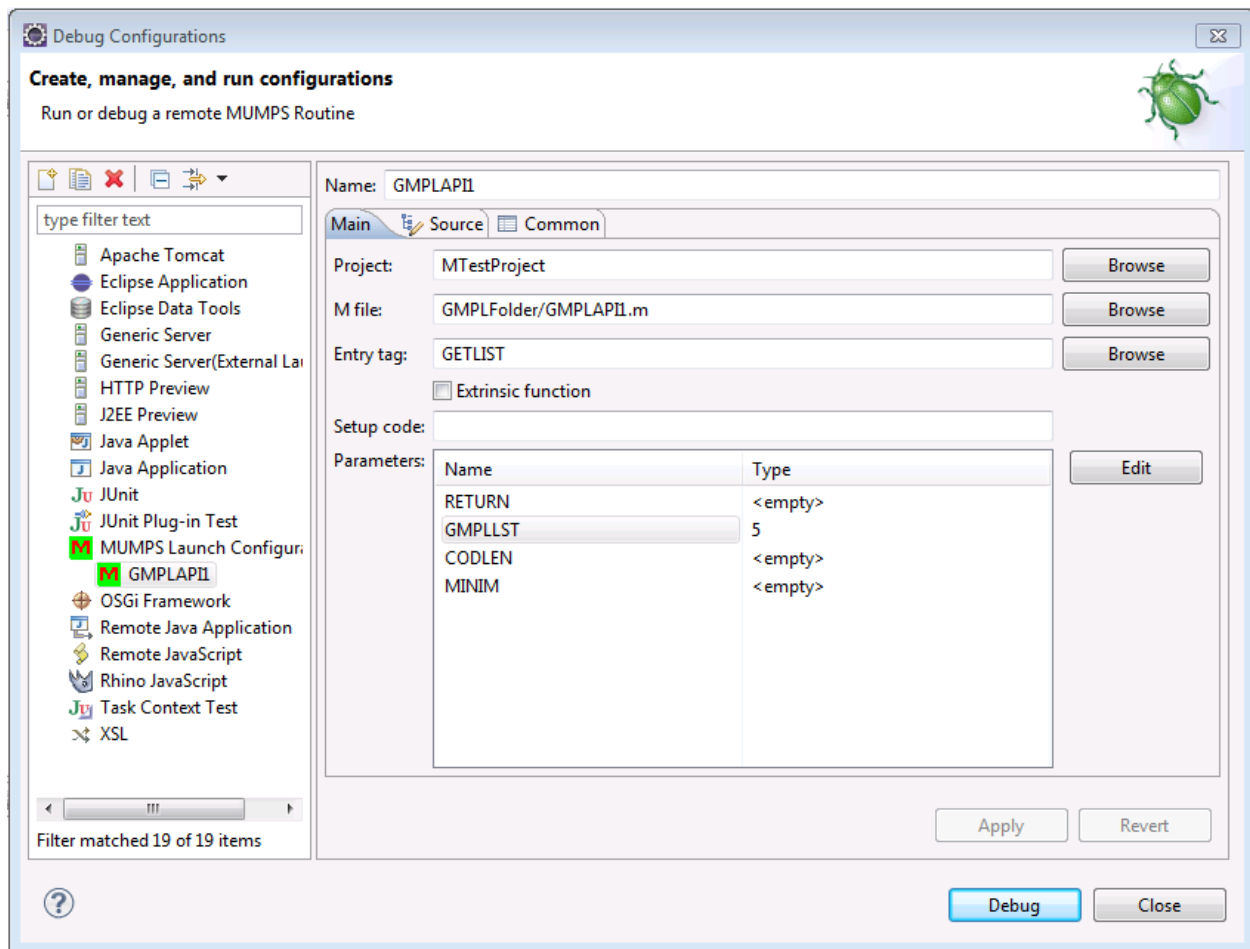
Variable watchpoints are triggered when a variable value changes, and can also be added from the breakpoints view by clicking the pencil icon. GT.M implementation does not support variable watchpoints.

## 4.4. Starting a Debug Session

To start the debugger, a new MUMPS launch configuration must be created. Launch configurations are organized with respect to routines, thus it is recommended they are created from the file context menu; doing so will populate most fields in the Debug configuration wizard.



Selecting Debug configurations brings up the Debug Configuration wizard where users can create a new launch configuration

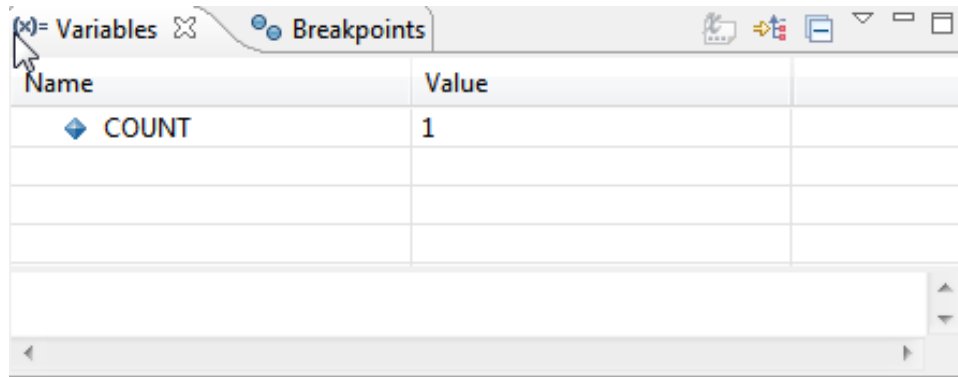


For a valid configuration, the following files must be specified: Project, M File, and Entry tag. When launched from a context menu, MTools auto populates the Project and M File fields. Once an Entry tag is selected, and if there are parameters for the entry tag, those values can be specified in the parameters table. The name of the parameters is automatically populated based on the Entry Tag. If the Entry Tag is an extrinsic function the check box for the same must be checked. Currently MTools do not automatically fill or validate this check box. Users can specify additional code that needs to be run before the selected tag is called. For example, this is useful to set the user (S DUZ=1) for user sensitive launches such as ^XUP.

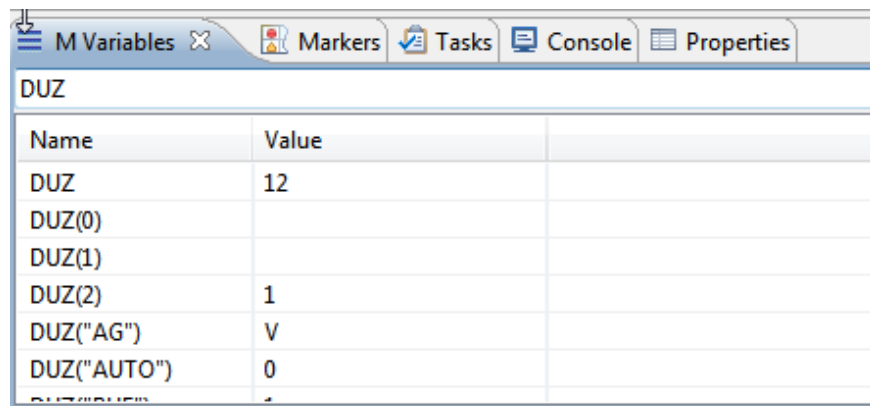
Once all the information is entered, you can then click 'Debug' to start the session. Note that once a debug launch configuration is created, you can use the M Application shortcut to launch the debug session for a particular file. If there are multiple configurations, a selection dialog will be presented to select one.

#### 4.5. Variables

Variables will be displayed in the default Eclipse Debug Variables view. For Generic Implementation, only those created after the debug session started are shown.



Since Generic Implementation uses VistA Link, there are additional VistA Link and RPC variables defined on the server that the default view does not show. There is a new custom view created for that. Users may open it by clicking on 'Window' → 'Show View' → 'Other...' and choosing M Variables. This view can also be filtered by using the text box on top of the variable viewer.

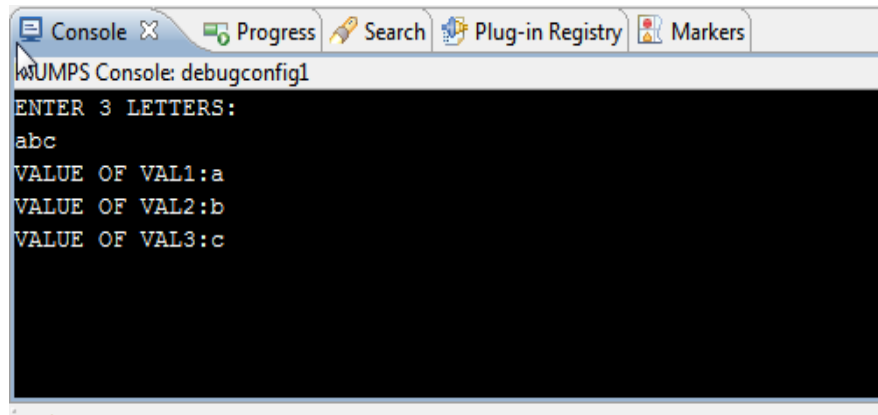


For GT.M SSH and Caché Telnet Implementations, there are no additional variables on the server other than those shown in Variables view; M Variables view is not populated.

## 4.6. MUMPS Terminals

### 4.6.1. Generic Implementation Interactive Console

MDebug Generic Implementation 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 the input.



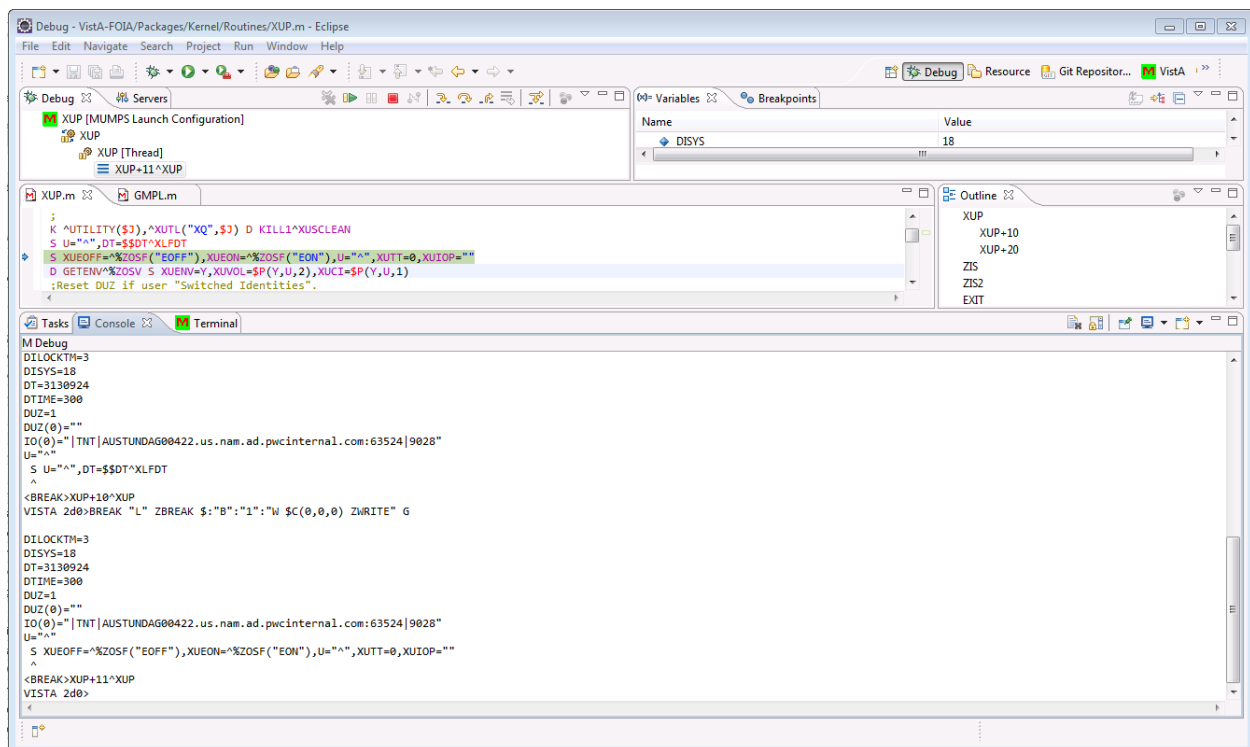
#### 4.6.2. Caché Telnet and GT.M SSH Terminal View and Message Console

MDebug Caché Telnet and GT.M SSH implementations use two views: MDebug Terminal and Console Views.

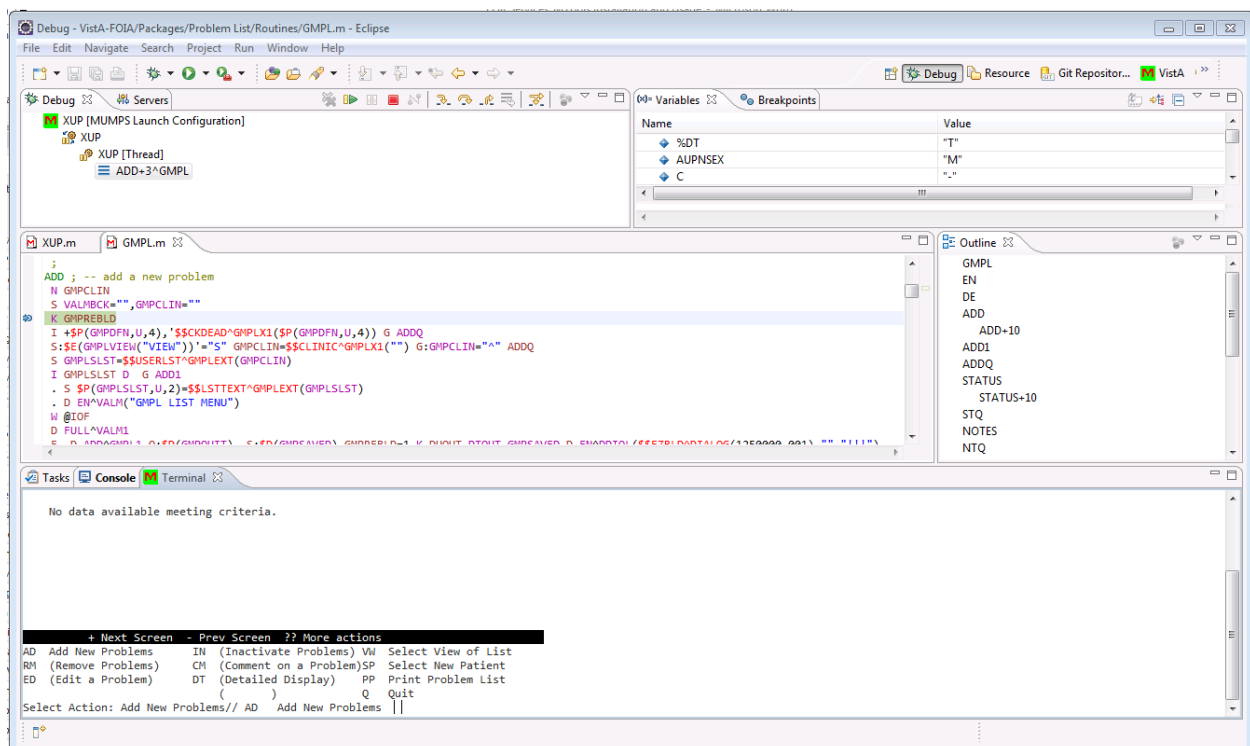
MDebug Terminal view is in fact the stripped down version of the Eclipse Terminal view. Eclipse Terminal view implements Telnet and SSH communications with the server and VT100 (ANSI) emulation. MDebug simply interrupts streams between the Terminal and the server; it also:

- Sends ZBREAK related commands to the server to define break points and stepping using the outbound stream
- Filters inbound stream to recognize break messages

All the actual MUMPS program reads and writes are written to the MDebug Terminal, and all launch and debug related commands and prompts are written to the MDebug Message Console. The following figure illustrates the MDebug Message Console with some underlying ZBREAK commands:

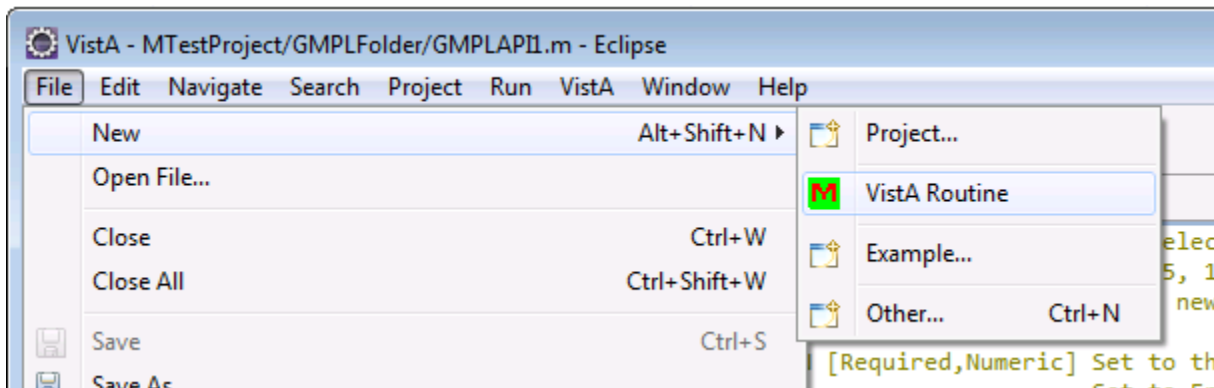


The figure below shows the MDebug Terminal View for a break in XUP when a user adds a new Problem:

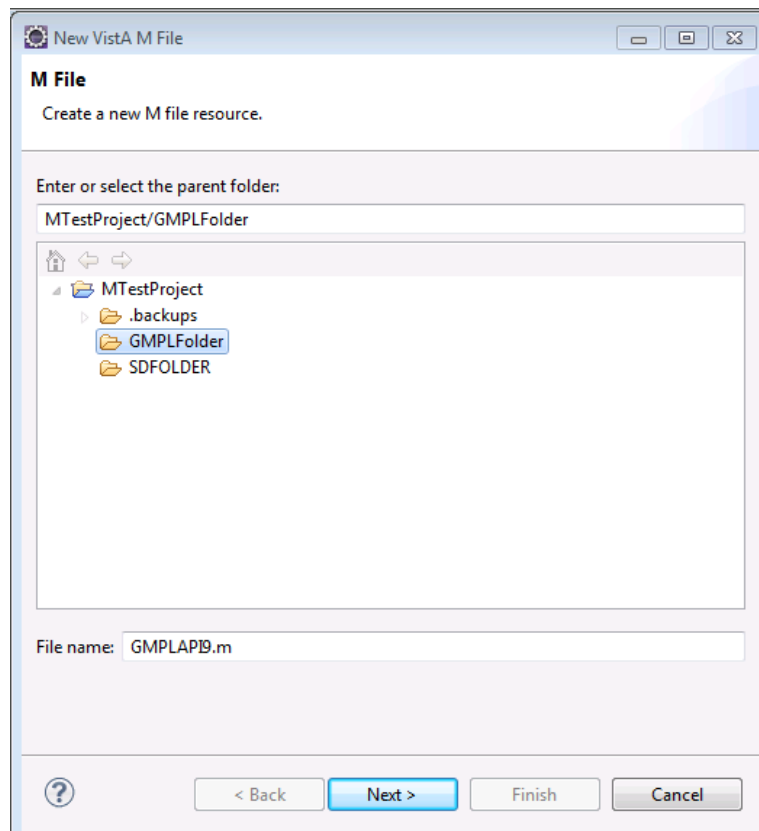


## 5. New VistA File Wizard

MTools provides a new New VistA File wizard from the File → New menu.



In the first page of the wizard, the name and folder of the new VistA file is specified; the file is restricted to be a MUMPS file.



The second page of the folder helps to create the first two lines of a VistA routine which must follow VA Standards and Conventions (SAC).



**New VistA M File**

**Routine Description**  
Create first two lines.

Site: PWC

Developer: AU/CB

Brief description: Test

Date: 7/21/2013

Major version number: 3

Minor version number: 2

Package name: Problem List

Patch number: 26003

Version date: 7/30/2013

Build number: 3

First line: ;PWC/AU/CB - Test;7/21/2013

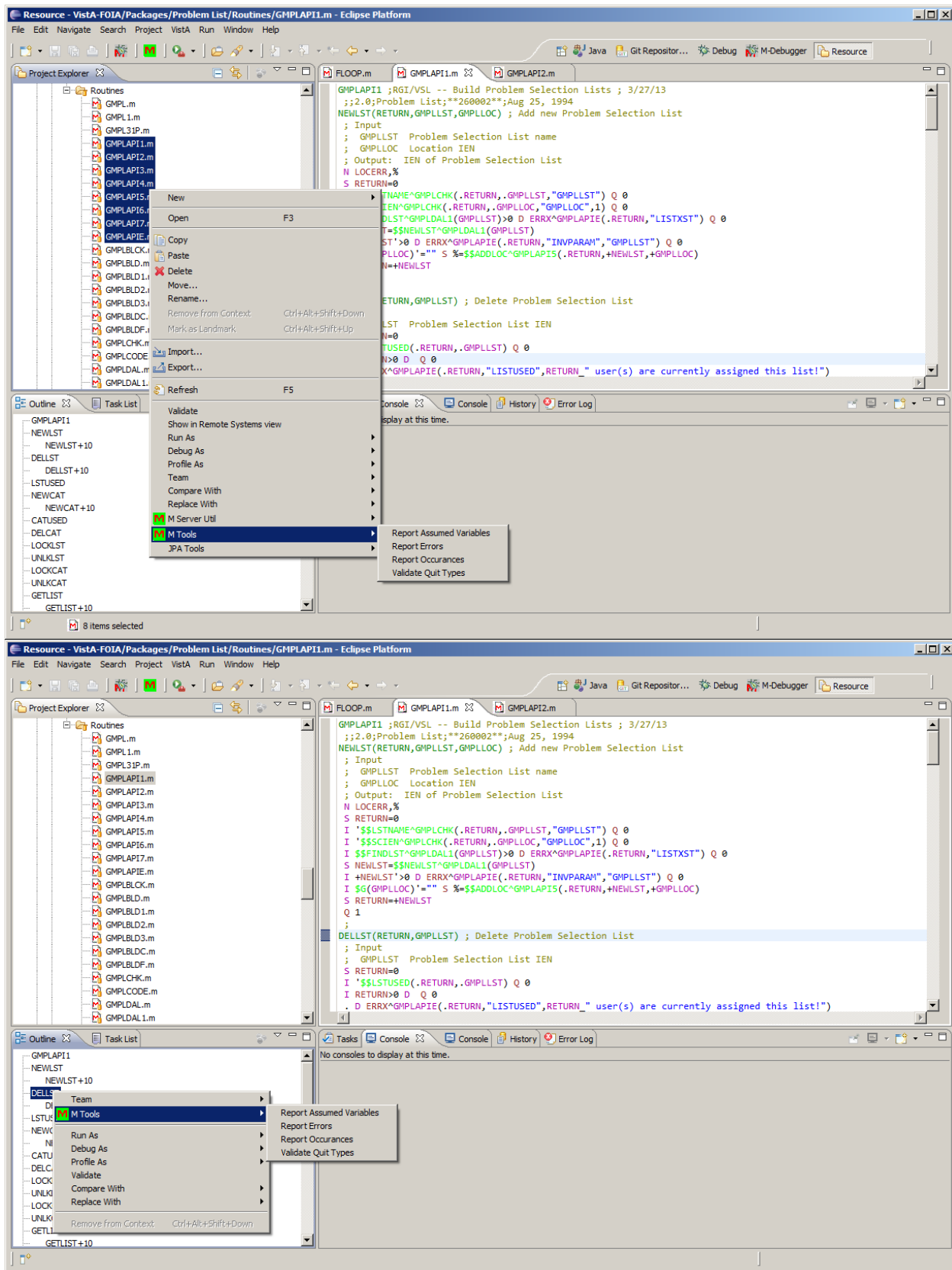
Second line: ;;3.2;Problem List;\*\*26003\*\*;7/30/2013;Build 3

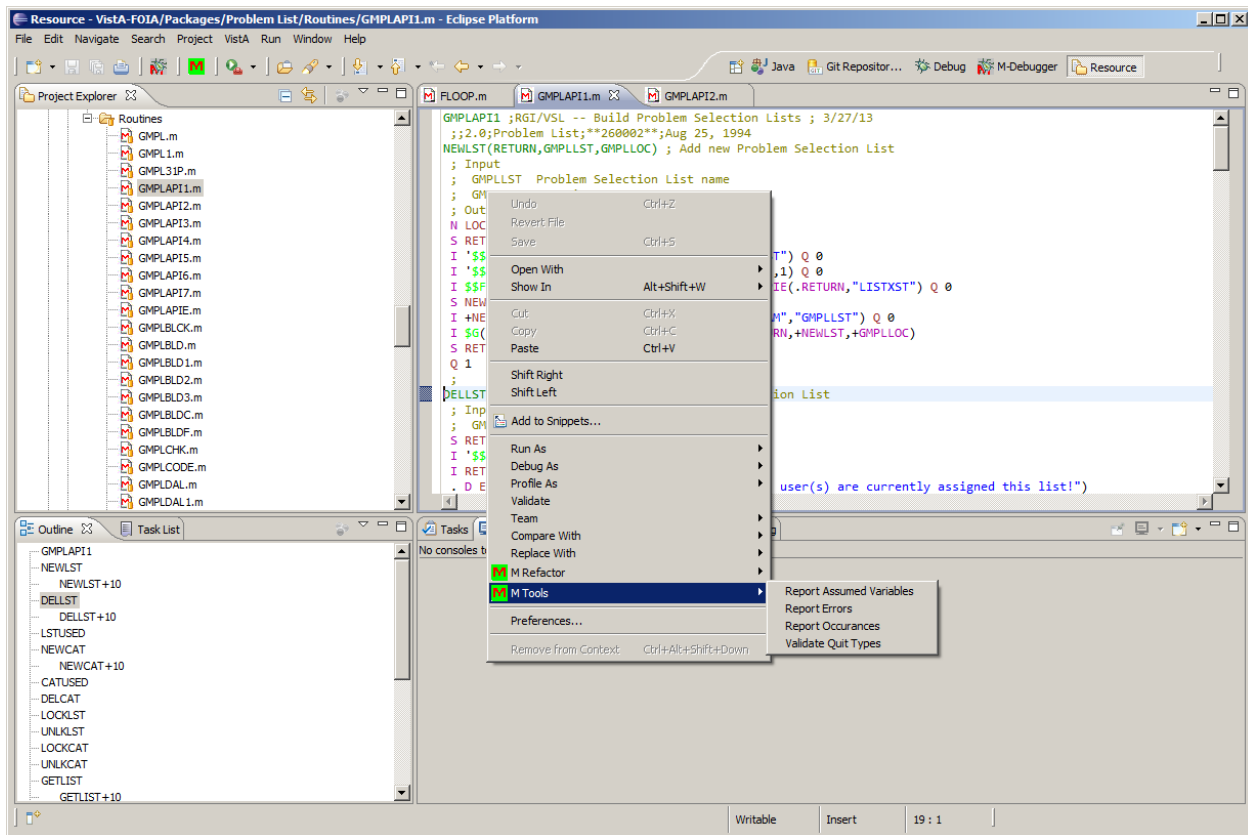
? < Back Next > Finish Cancel

Note that new VistA files are not automatically saved to the server. When you hit finish in the wizard, the file is opened in MEditor and users can save the routine to the server as described in the previous sections. The New VistA M File wizard replaces functionality in the previous version of MTools where files were previously updated on the server side.

## 6. MTools Context Menu Tools

A number of MUMPS code validation and analysis tools are available in the MTools IDE. These tools can be found under the M Tools menu item in the context menus of the Project Explorer view, Outline view, and MEditor. Users should note that these tools are client based and do not use a MUMPS server.



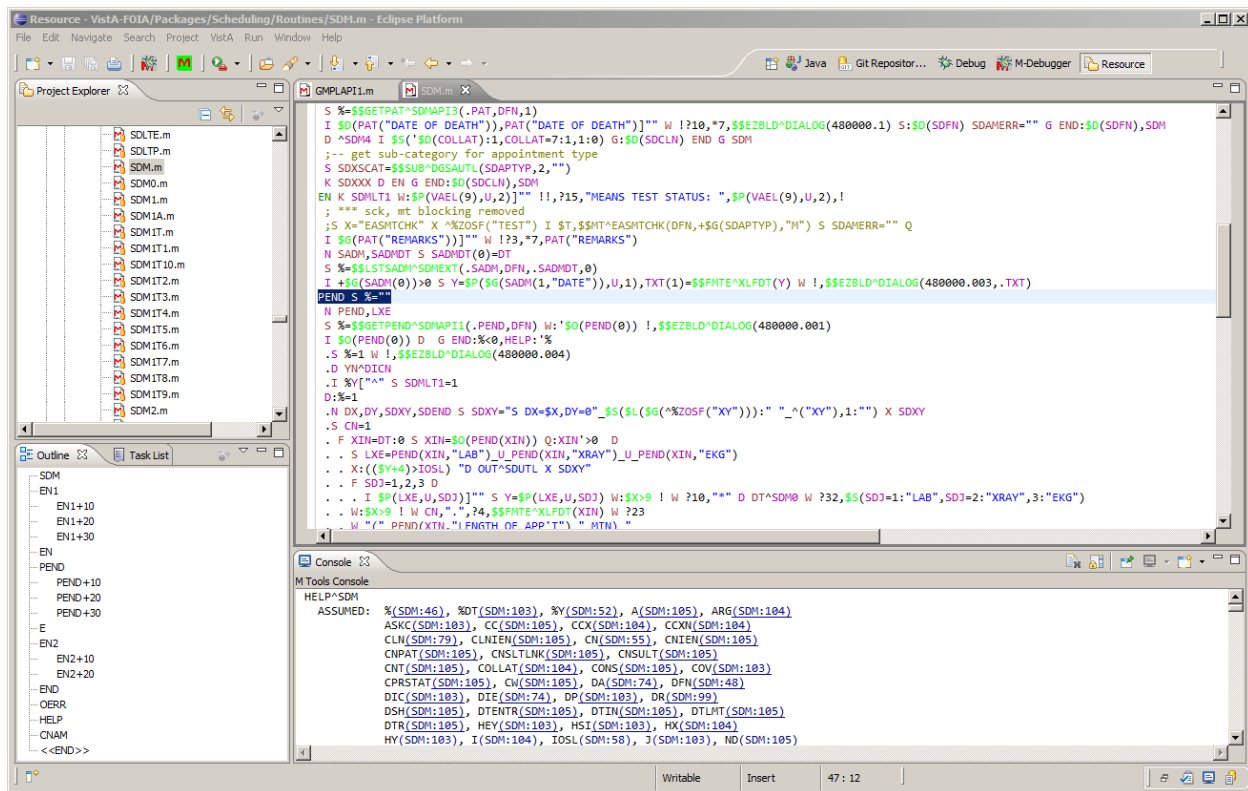


Currently there are four tools: Report Assumed Variables, Report Errors, Report Occurrences, and Validate Quit Types. Each of these tools is run for each entry point tag in routines. When selected from the Outline view, they are run for a specific tag; when they are run from MEditor, they are run for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

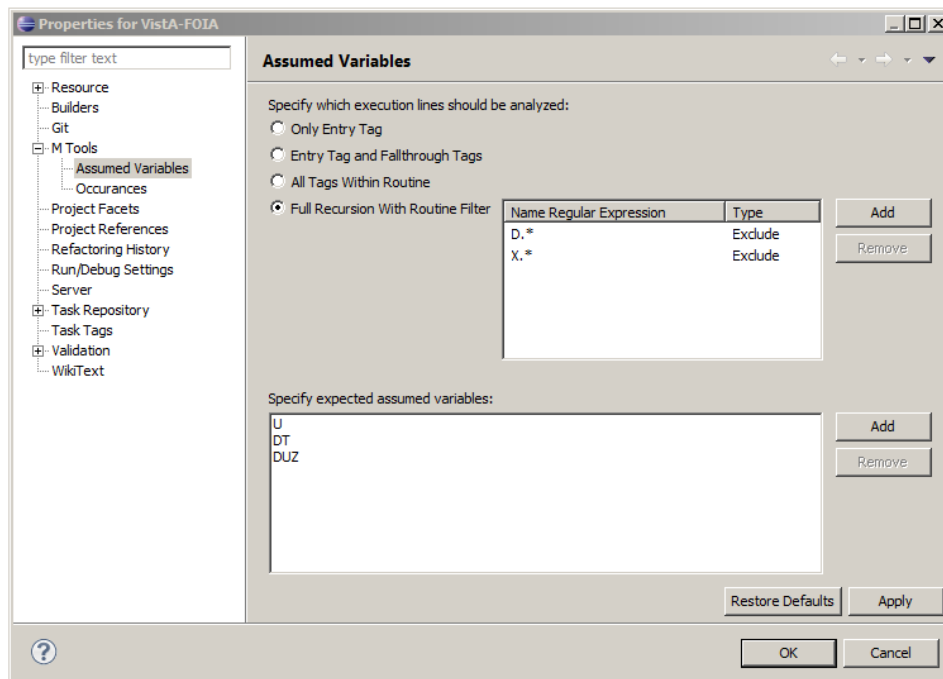
## 6.1. Report Assumed Variables

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 the clickable location of the assumed variable; 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 only, 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 the 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 GOTO commands 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 assumed by design.

## 6.2. Report Errors

This function 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.

## 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.



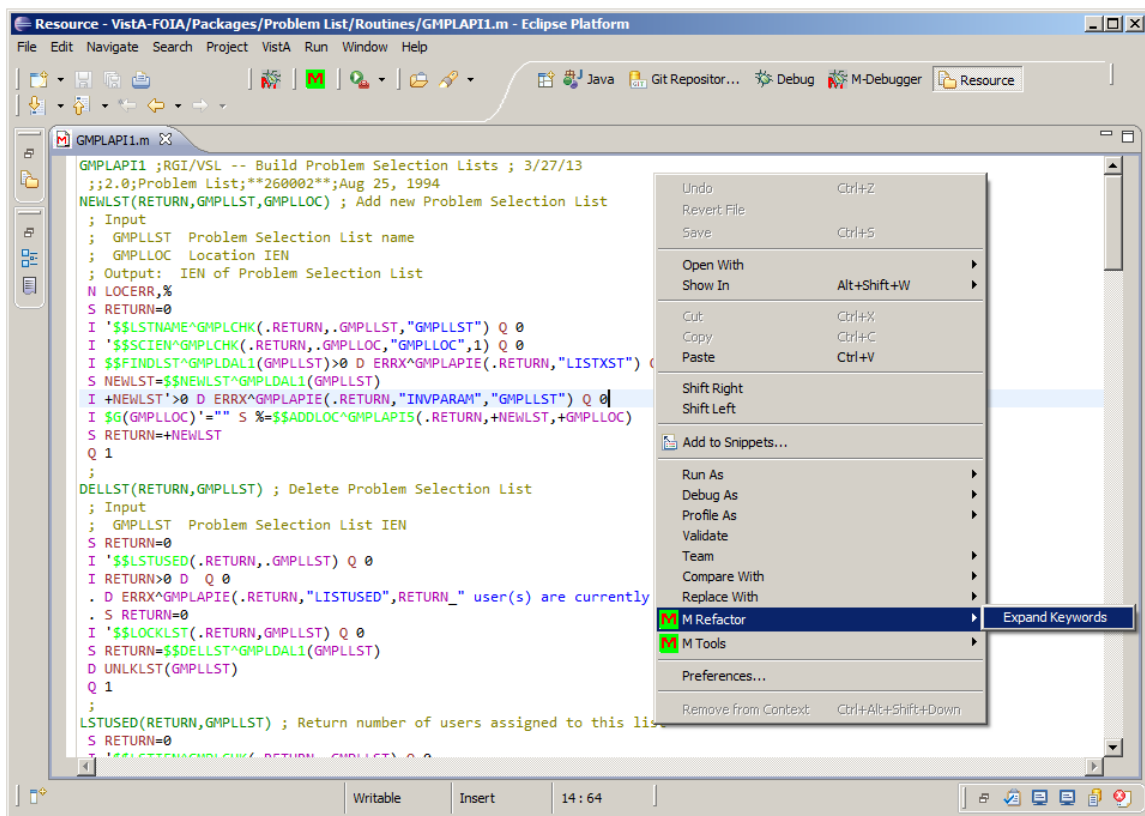
## 6.4. Report Quit Types

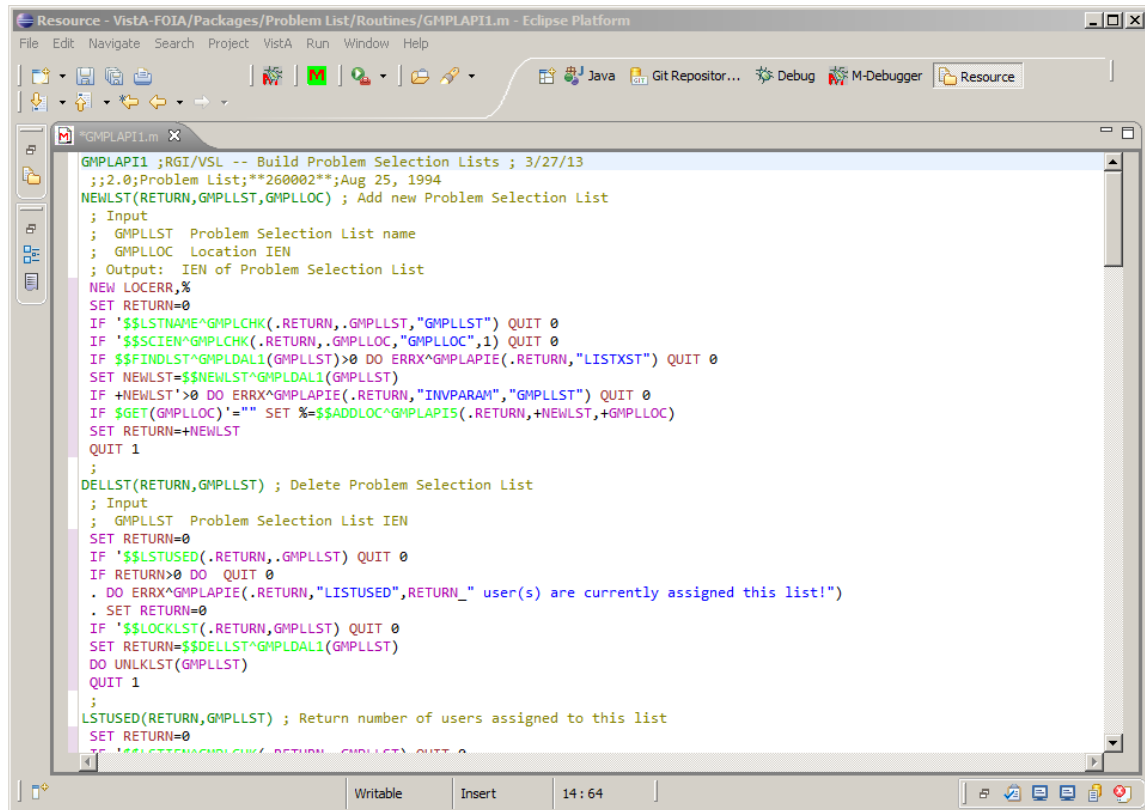
Report Quit Type performs two analyses: the first ensures that each entry tag has consistent QUIT commands; all QUIT commands in an entry tag must either all return values or all not return values. The analysis is recursive with respect to GOTO commands.

The second analysis looks for DO/GOTO commands that are called for extrinsic functions and SET commands that are called for tags that do not return value.

## 7. M Refactor Context Menu Items

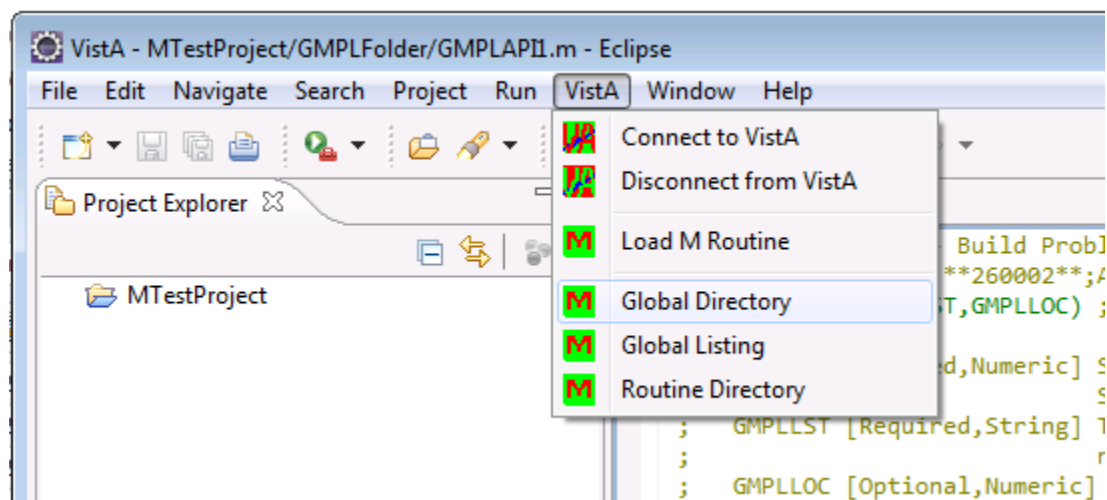
The M Refactor context menu item is available only from MEditor and it currently has a single action to Expand Keywords. This item changes all the command and intrinsic function mnemonics to their full form. The following screenshots illustrate before and after Expand Keywords state of a routine:





## 8. Vista Main Menu Utilities

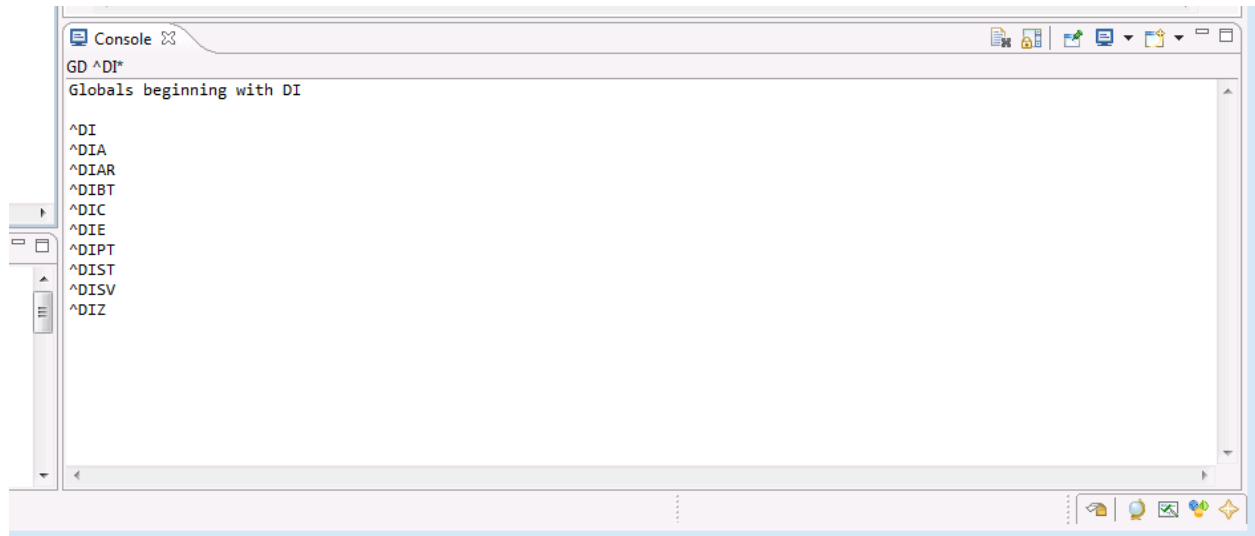
MTools provides three utilities that are not specific to any project, but give information on routines and globals on the server. These utilities are available from the Vista main menu.





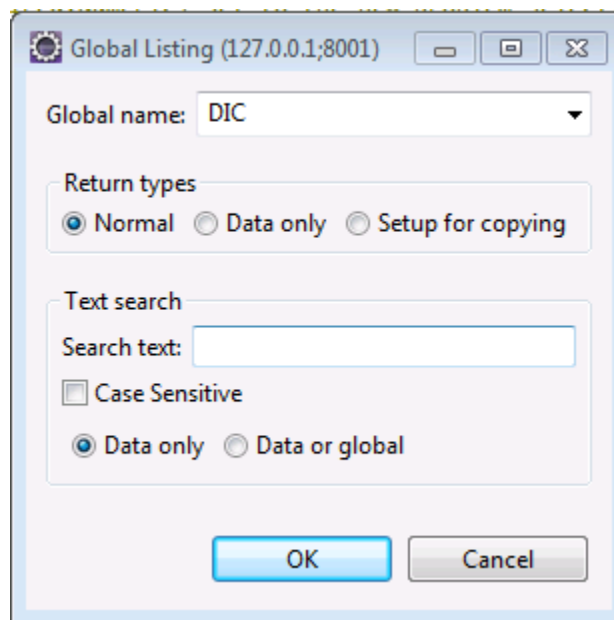
## 8.1. Global Directory

This utility lists all the globals with a specified namespace. When this menu item is selected, the user is prompted to enter the namespace and the results are displayed in the Console view. The following shows the output when the namespace is specified to be DI.



## 8.2. Global Listing

This utility lists the content of a specified global. The global information to be listed is entered in a dialog which is displayed when Global Listing is selected.

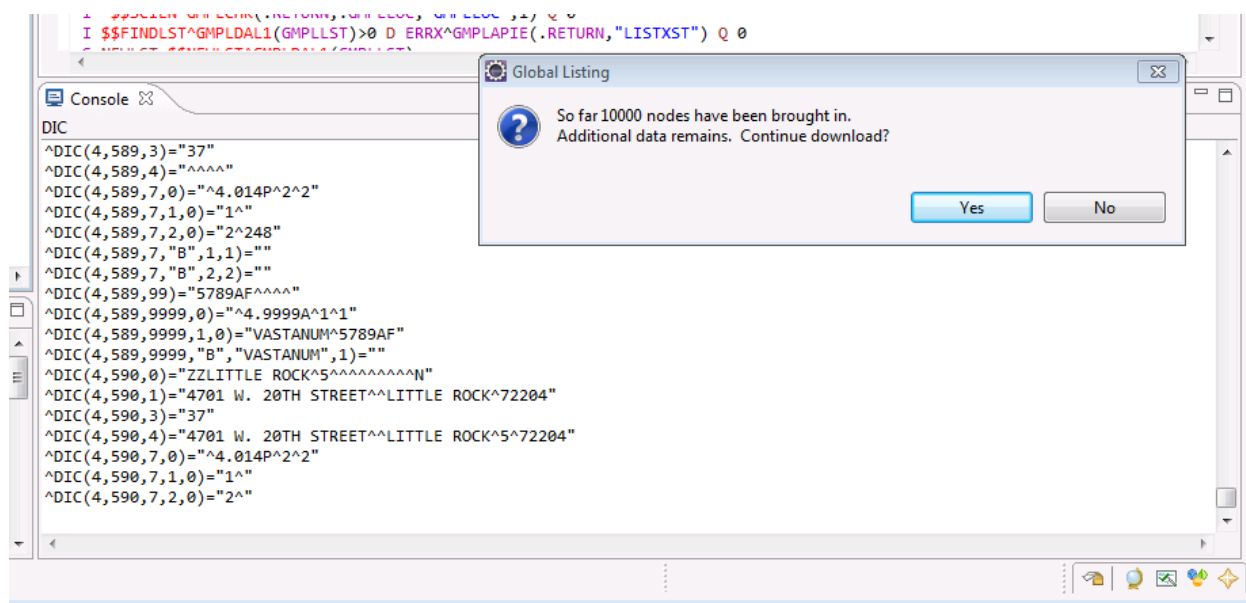


If Normal return type is selected, the global, indices, and data are displayed. For Data Only, the global and indices are not displayed. If Setup for Copying is selected, the output is a legal

MUMPS script with SET commands that can be copied and pasted in to a routine. In all cases, the output is written to the Console view.

Users can further filter the results by specifying a search text. For this case only, those entries that contain the search text are displayed.

The result below shows the results for global DIC as specified in the dialog above; note that the results are shown in 1000 line increments.



### 8.3. Routine Listing

This is similar to Global Listing, but shows all the routines in a specified namespace. The result below shows when user specifies namespace GMPL. The namespace input is shown immediately after user selects Routine Listing.

