MTools IDE

Usage and Installation

# Installation

# Prerequisites

MTools IDE supports both Windows and Linux.

|  |  |  |
| --- | --- | --- |
| **Application** | **Version** | **Notes** |
| VistA / Fileman | Any | This is currently tested with VistA-FOI, available here: <http://www.osehra.org/page/osehra-code-repository> |
| Eclipse | Indigo |  |
| Java Runtime | 7 |  |

# Download the latest files

Download the latest version here: <https://github.com/JimDeanSpivey/M-Tools-Project/archive/master.zip> and unpack this file. It contains the plugin jar files needed, along with the KIDS packages.

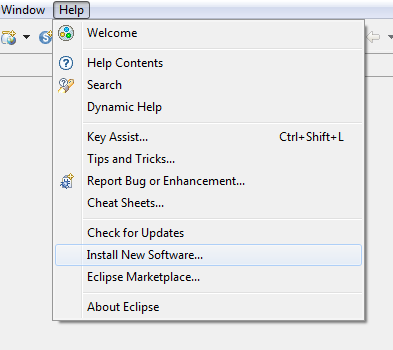
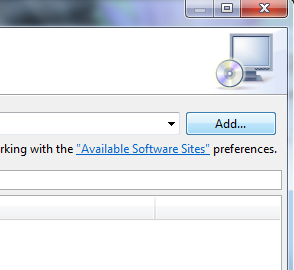
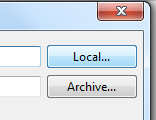
# Install the KIDS packages

From the unpacked file, the KIDS packages are located under /MiscDependencies/KIDS/. Install all three packages, M-Debugger, M-Editor and Utilities.

# Run the VistALink job

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

# Install the plugin

1. Open Eclipse.
2. Click Help 🡪 Install New Software  
   
3. Then click Add in the top right.  
   
4. Select the button named Local. This in towards the top right.  
   
5. Select the directory where the zip file was unpacked. Then choose the “MToolsUpdateSiteProject” directory.
6. After clicking OK, thenewly added update site will automatically be selected.
7. Click Next and follow the prompts to install the plugin.

# Configure the plugin

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.

# Usage

# Eclipse Projects

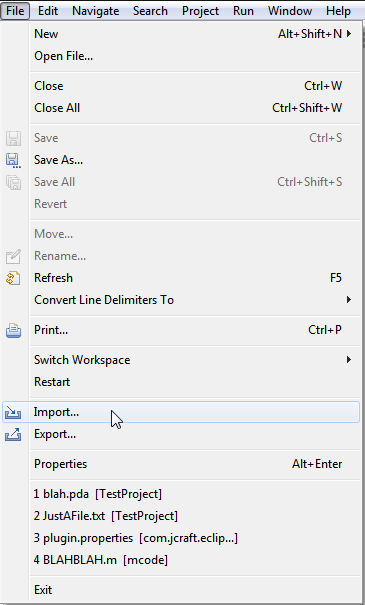
All files in eclipse must belong to a project, being that Eclipse can be described as project centric. To begin using MTools IDE, a new project must either be created, and files imported into it. Or an existing project can be opened, which already contains files and of course new files can be imported. Files can also of course be removed and/or deleted from eclipse. Additionally a file can be 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.

# Creating a new project

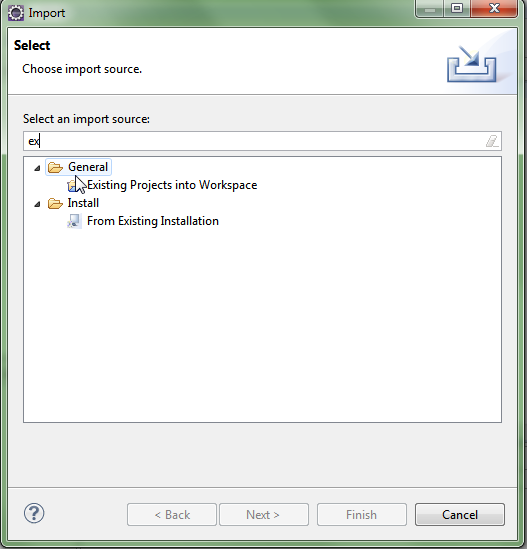
A standard eclipse project can be created via New -> New Project… This opens the new project wizard. The standard eclipse project can be chosen from General -> Project.

# Opening an existing project

To open an existing project, click on ‘New -> Import…‘ .



From the Import Wizard choose ‘Existing projects into Workspace’.



This can be filtered to easily by typing ‘ex’. Using the file dialog navigate to the directory which contains an existing project (a directory with a .project file in it). 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.

# Using the 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.

# Loading Routines

Routines can be loaded from the server. Either click the green M button in the eclipse Toolbar or, in 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 to. For example if you try 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 it, showing a diff if anything changes and prompting which file to choose.

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.

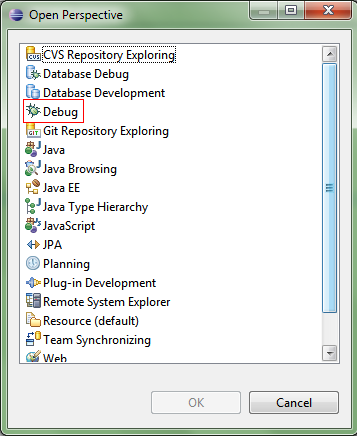
# Saving routines

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 compare the latest backup file. 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 of course, the files should be different as there should be changes to submit. It will actually show an informational dialog if the routine to be saved is the same as what is on the server.

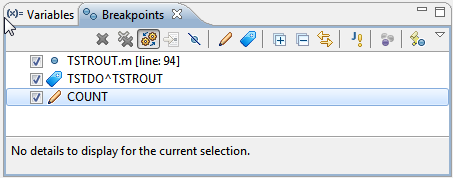
# Debugging

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



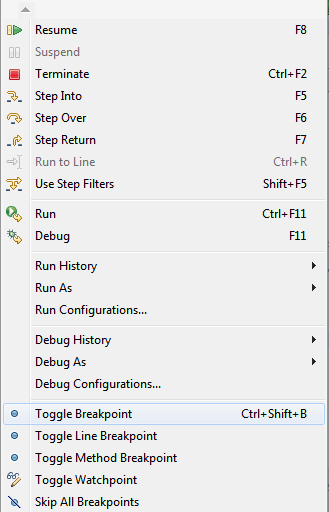
# Adding breakpoints

The MDebug plugin has 3 types of breakpoints, line breakpoints, tag breakpoints and variable watchpoints. All breakpoints can be seen from the breakpoints view, under the debug perspective.

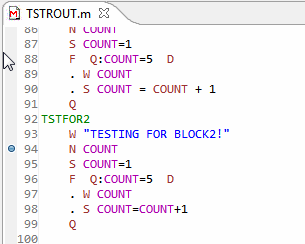


# Line breakpoints

Open the M Code in an editor that you want to debug. Either press ‘ctrl + shift + B’ or go to the Run Menu at the top of the Eclipse application and select toggle Line Breakpoint.



Third, you can also double click on 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.



# Tag breakpoints

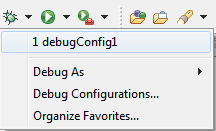
Arbitrary tags can be entered (eg: TAG^ROUTINE) too. From the breakpoints view, click on the blue tag icon.

# Watchpoints

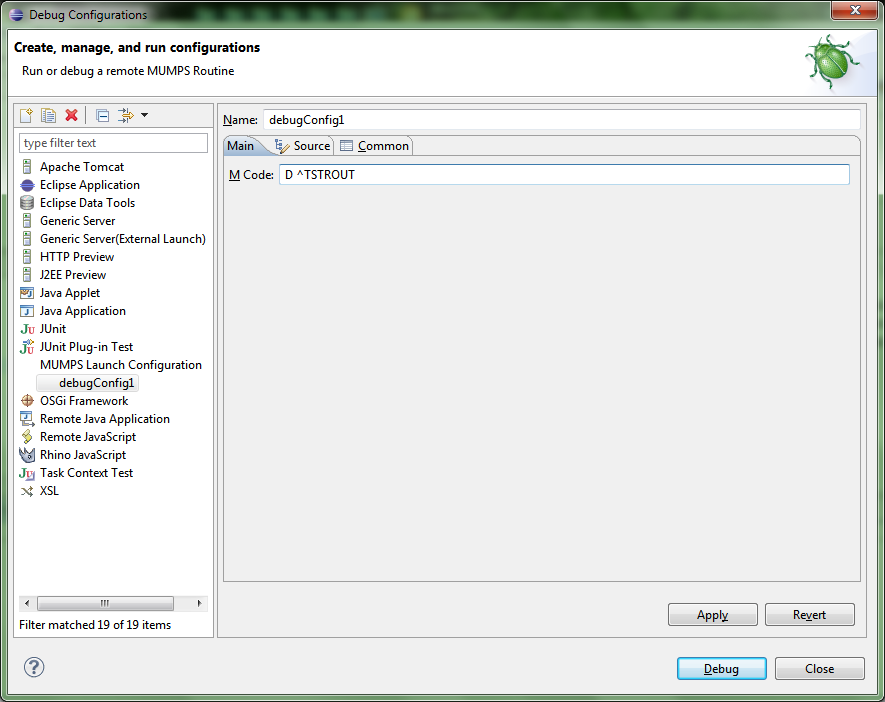
Variable watchpoints can be added. They are triggered when a variable value changes. They too are added from the breakpoints view, by clicking the pencil icon.

# Starting a debug session

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

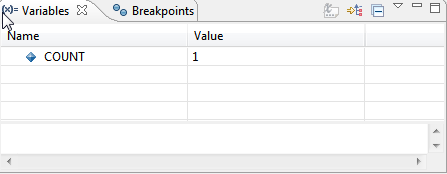


Enter the M code which you want to debug, and which ideally will encounter your breakpoints.

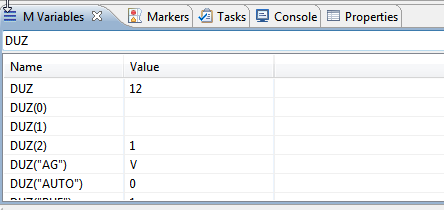


# Variables

Variables will be displayed in the default Eclipse Debug Variables View. But only those variables created after the debug session has 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 atop the variable viewer.



# Interactive Console

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

# 

# MTools Right Click Tools

# TODO: cover installation regarding environment variable to setup MRA location