Debugging Milestones Automation Applications on a Server (and Desktop)

Desktop Debugging

Debugging Milestones Automation programs on your desktop is fairly easy, using a combination of error messages from your scripting language that helps you identify the line of code rejected by Milestones, and the optional Milestones Automation Debug window.

For example, the following code:

```
Public Sub BadCall()
    Dim testresult As String
    Set objMilestones = GetObject("c:\users\delder\documents\test.mle")
    With objMilestones
        .Activate
        .setdebugmode "on"
        .AddSymbol 2, "28/15/2013", 2
        .Refresh
        .Close "NoSave"
    End With
    Exit Sub
End Sub
```

has an error in it in a call to Milestones.

When you run it under an Office program, such as Access or Excel, as a VBA script, you will get the following error screen:
which is somewhat limited.

If you click on the Debug button, at least the line that caused the problem is highlighted:
But the actual error is still not identified.

Luckily, this program also made a call to the Milestones method “SetDebugMode”, so if we look at the Milestones Automation debug window, more information is found:
Which will eventually lead us to conclude that “88” is not a proper month in a date string.

The automation interface allows only limited information to be sent back via an error return, just a failure notification and a very limited set of failure codes, thus most of the time, the Milestones interface will issue a “Type Mismatch” error for almost all errors.

This is why the Automation debug window was added, and which can be turned on or off via the SetDebugMode method or via the button in the Milestones | help | View System Info screen:

![System Information](image)

When you press this button, the Automation debug window appears, and any future automation calls will show up in the debug window. This can be useful for already deployed desktop applications that are failing, and have no way of turning on debug mode by themselves.

**Server Debugging**

Since, when you are running Milestones on a server, most likely under IIS, the Milestones GUI is not available, so there is no way to directly turn on debugging via the System Info screen, or even to see the debug window, since it is running under a different process and has no desktop window anyway.

While it is possible that you have a development environment that lets you remotely debug server based applications from within your development environment, in many cases that sort of remote debugging is not possible due to security or other issues. The rest of this paper discusses how to debug a server based application in the situation where the running code cannot be debugged from within the development environment.

So, if we have the following program on a Windows 2008 server that we are running under ASP.NET:
Note that the second PutCell has an error. However, if we simply run this from a browser, the default error page again tells us that we have a Type Mismatch error:
on a “Late Binding” call, but no line number or actual method that failed is shown in the Stack Trace. The page on this server does suggest though that we add `<compilation debug="true"/>` to the web.config file.

If you are running under ASP.NET, you can use the web.config file to at least get some line information. So if add the compilation debug=true to the web.config file we get a better error screen:
Now at least the failing line number is highlighted and we know the error was in a call to PutCell. If that still is not enough information to determine the problem, it is possible to get the same Milestones automation debug window information by using a Try/Catch sequence.

There is a method in Milestones Automation called “GetDebugText”, which when called, will dump the last 4000 characters in the debug window to a text file, so even though you cannot see the debug window, you can get its text.

So, if we enhance our sample application to have a Try/Catch sequence it will look like this:
Note that we have added the code boxed in yellow. Now when we run this we get the following error screen:

Test Type mismatch. [Exception from HRESULT: 0x80020005 (DISP_E_TYPEMISMATCH)]

which is not too useful since it just writes out the error code, but if we look at the debug.txt file that the call to GetDebugText in the Catch section created:
We can see that the second call to PutCell had an invalid column number (22).

**Dangling miles.exe processes**

If you have a situation where your Milestones automation application has terminated unexpectedly and the Milestones Close method was never called, you most likely have left some miles.exe processes running.

You should always check to see under Task Manager to see if any are running:

Be sure to click on the all users checkbox, since miles.exe will be running under a web site ID and not yours.
Normally a dangling process does not matter that much, unless you forgot to have the “Use Multiple Instances for COM/Automation checked in Milestones Tools | Program Options | Files and Automation screen:

If this option was not checked, then a possibly unstable copy of miles.exe will attempt to act as the automation server for any future Milestones objects. Also, if an open Milestones file is stuck in this dangling process, then other users may not be able to update it, or since it will be in the Running Object Table it may not be possible to open another copy of it via another automation session.
Issuing a .Close without the “NoSave” parameter can leave a Milestones file open, as the GUI may be displaying the “Do You Want to Save Your File” dialog in a hidden window with no way to get a response.

**Late Binding**

Earlier in this document a reference was made to “Late Binding”. The alternative to “Late Binding” is “Early Binding”. Binding has to do with how a program references the API of another program. Early binding allows the references to be “locked up” at compile time and no extra work is required to figure out API calls when the program actually runs. Late Binding leaves the API references open, and unchecked, until the program actually runs. Milestones uses “Late Binding” and that is why there is no reference file to aid you in typing in the calls to the various calls available via the Milestones API.

For more info on the technical details of early vs. late binding, please see this Microsoft document: [http://support.microsoft.com/kb/245115](http://support.microsoft.com/kb/245115)

Milestones is an OLE Automation (COM) server designed to run as an independent executable. Other applications can instantiate a Milestones object and then control it via calls to the Milestones API.

The nice thing about a “Late Bound” program is that it is version independent, i.e. even though the internals of Milestones get updated with each new version, all prior automation programs that use Milestones continue to function even though the underlying application has been updated, i.e. you are not forced to recompile your automation app with the latest reference file or worry that your app will no longer function just because you have updated Milestones to a new version.

Since Milestones is distributed as an EXE file and not a DLL you don’t have to worry about whether or not the Milestones exe is 32 or 64 bit or whether the calling app’s bitness matches.