



NAVI MUMBAI

MATLAB

Unit 7-Lecture 27

Debugging process , setting breakpoints

BTech (CSBS) -Semester VII

14 October 2022, 09:35AM



Debugging M-files

- 1) preparing for debugging,
- 2) examining values,
- 3) Debugging process
- 4) setting breakpoints
- 5) running with breakpoints
- 6) correcting and ending debugging
- 7) correcting an M- file



Build Process

- Before you can build an executable program or shared library for a model, choose and set up a compiler or IDE and configure the target environment.
- Several methods are available for initiating the build process.
- Tooling is available for reloading, rebuilding, and relocating generated code.
- If your system includes referenced models, reduce build time and control whether the code generator regenerates code for the top model.
- To improve the speed of code execution, consider using available profiling capabilities.

Functions

▼ Initiate Build Process	
<code>packNGo</code>	Package generated code in ZIP file for relocation
<code>rtw_precompile_libs</code>	Build model libraries without building model
<code>codebuild</code>	Compile and link generated code
<code>rtwrebuild</code>	Rebuild generated code from model
<code>slbuild</code>	Build standalone executable file or model reference target for model



Build Process

▼ Get or Modify Build Process Controls

<code>coder.buildstatus.close</code>	Close Build Status window
<code>coder.buildstatus.open</code>	Open Build Status window
<code>RTW.getBuildDir</code>	Get build folder information from model build information
<code>Simulink.fileGenControl</code>	Specify root folders for files generated by diagram updates and model builds
<code>switchTarget</code>	Select target for model configuration set



Debugging Approaches

There are probably a lot of ways to debug programs. These include:

- 1) editing the code and removing semicolons or adding a **keyboard** statement at judicious locations
- 2) **setting a breakpoint** at a particular line and stepping through code from there
- 3) using a variant of setting a breakpoint by using **dbstop if error**
- 4) seeing if the **mlint code analyser** can help (also reachable from the Tools menu)
- 5) comparing variants of the code using the **File and Folder Comparisons** tool or **visdiff** for command-line access



Set Breakpoint

Setting breakpoints pauses the execution of your MATLAB® program so that you can examine values where you think an issue might have occurred. You can set breakpoints interactively in the Editor or Live Editor, or by using functions in the Command Window.

There are three types of breakpoints:


- 1) Standard
- 2) Conditional
- 3) Error



Set Breakpoint

By default, when MATLAB reaches a breakpoint, it opens the file containing the breakpoint.

To disable this option:

1. From the **Home** tab, in the **Environment** section, click  **Preferences**.
2. In the Preferences window, select **MATLAB > Editor/Debugger**.
3. Clear the **Automatically open file when MATLAB reaches a breakpoint** option and click **OK**.



Standard Breakpoints

A standard breakpoint pauses at a specific line in a file. To set a standard breakpoint, click the gray area to the left of the executable line where you want to set the breakpoint. Alternatively, you can press the F12 key to set a breakpoint at the current line. If you attempt to set a breakpoint at a line that is not executable, such as a comment or a blank line, MATLAB sets it at the next executable line.

```
1  n = 50;  
2  r = rand(n,1);  
3  plot(r)  
4  
5  m = mean(r);  
6  hold on  
7  plot([0,n],[m,m])  
8  hold off  
9  title('Mean of Random Uniform Data')
```




Standard Breakpoints

To set a standard breakpoint programmatically, use the **dbstop** function.

```
dbstop in plotRand at 3
```

When debugging a file that contains a loop, set the breakpoint inside the loop to examine the values at each increment of the loop. Otherwise, if you set the breakpoint at the start of the loop, MATLAB pauses at the loop statement only once. For example, this code creates an array of ten ones and uses a for loop to perform a calculation on items two through six of the array:

```
x = ones(1:10);  
  
for n = 2:6  
    x(n) = 2 * x(n-1);  
end
```



Standard Breakpoints

For MATLAB to pause at each increment of the for loop (a total of five times), set a breakpoint at line four.

```
3  for n = 2:6  
4      x(n) = 2 * x(n-1);  
5  end
```



Conditional Breakpoints

- To set a conditional breakpoint, right-click the gray area to the left of the executable line where you want to set the breakpoint and select Set Conditional Breakpoint.
- If a breakpoint already exists on that line, select Set/Modify Condition. In the dialog box that opens, enter a condition and click OK.
- A condition is any valid MATLAB expression that returns a logical scalar value.
- **Example:** Set a conditional breakpoint at line four with the condition $n \geq 4$. When you run the code, MATLAB runs through the for loop twice and pauses on the third iteration at line four when n is 4. If you continue running the code, MATLAB pauses again at line four on the fourth iteration when n is 5, and then once more, when n is 6.



Conditional Breakpoints: Example

Code:

```
x = ones(1:10)

for n = 2:6
    x(n) = 2 * x(n-1);
end
```

Result:



1	x = ones(1:10);
2	
3	for n = 2:6
4	x(n) = 2 * x(n-1);
5	end

Stop the file:

```
dbstop in myprogram at 6 if n>=4
```



Error Breakpoints

To set an error breakpoint, on the **Editor** tab, click  **Run**  and select from these options:

- **Pause on Errors** to pause on all errors.
- **Pause on Warnings** to pause on all warnings.
- **Pause on NaN or Inf** to pause on NaN (not-a-number) or Inf (infinite) values.

```
dbstop if error
```

```
dbstop if caught error MATLAB:ls:InputsMustBeStrings
```