Automation test

Agenda

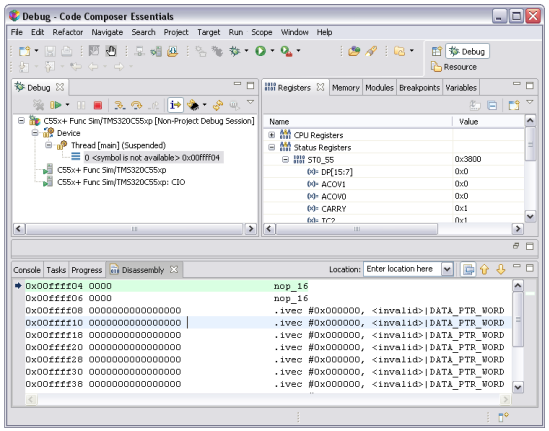
- Overview
- GEL(General Extension Language)
- DSS(Debug Server Script)
  - JavaScript
  - CCS scripting Console

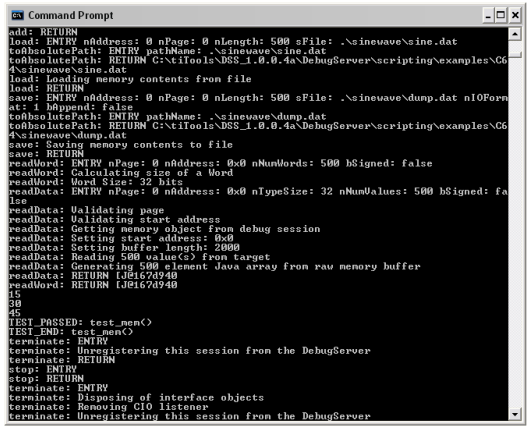
Why Automate?

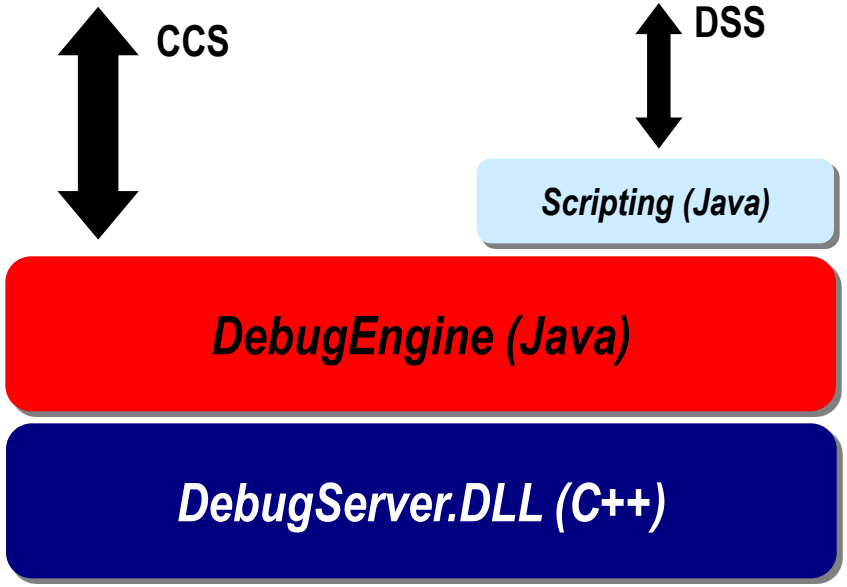
- Overnight testing and validation of applications
- Speed up start-up sequence
- Minimize user interaction for repetitive or tedious actions

CCStudio Tools to Automate

- GEL (General Extension Language)
  - Run within CCS
- DSS (Debug Server Script)
  - Installed with CCS by default, but actually independent of CCS.







GEL (General Extension Language)

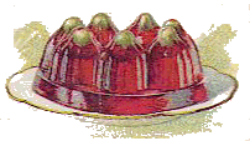
- ‘C’ like scripting language that extends CCStudio usefulness

- Many built-in GEL functions for common actions
  - All GEL functions can be called in DSS like below:

```
debugSession.expression.evaluate("GEL_AdvancedReset(\"System Reset\")");
```

- Create custom GEL functions for additional functionality
  - Automate several steps within one GEL function
  - Create GEL “hotmenu” items for easy access

Load GEL scripts(s)



- Load the GEL script(s) into CCStudio memory (File->Load Gel)
  - The script's **`StartUp()`** function will be executed upon a load (if one exists)
- Specify a default GEL file to be loaded upon target connection in target configuration.
  - The script's **`OnTargetConnect()`** function will be executed upon connection(if one exists)
- Automate loading several GEL scripts by using built-in GEL function: **`GEL_LoadGEL()`**

GEL Callback functions

- Built-in GEL callback functions provided
  - **`Startup()`:** Executed when the processor is initialized
  - **`OnTargetConnect()`:** Called when connection to the target has been established (only in versions of CCStudio 2.40 and greater)
  - **`OnFileLoaded()`:** Called after a program is loaded
  - **`OnPreFileLoaded()`:** Called before a program is loaded
  - **`OnReset()`:** Called after a target reset
  - **`OnRestart()`:** Called after a program restart
  - **`OnHalt()`:** Called after a user halt

- Use callback functions to execute a (series of) GEL function(s) upon an event

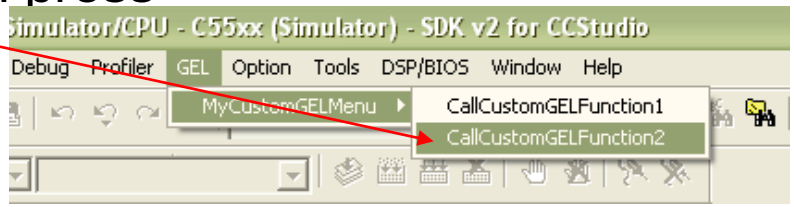
- Much of GEL's behavior is asynchronous
  - Built-in functions have asynchronous behavior
  - Use callback functions to ensure automatic execution of a GEL function *after* a certain event has occured

GEL Limitations for Automation

- Can only be interfaced from within CCStudio IDE
  - CCStudio must already be configured and IDE running
- Asynchronous behavior of built-in functions
  - Difficult to create long complex scripts
  - Built-in callback functions help but not enough
    - Cannot create custom callback functions
- GEL was never meant for full blown automation purposes like overnight testing or batch-runs

When to use GEL (for Automation)

- Use GEL to automate:
  - Startup actions
    - Utilize default GEL startup file function
      - Target initialization and sanity tests
      - Setting up Memory Map for CCStudio
  - Simple actions *within* the IDE
    - Automate a series of asynchronous IDE (built-in) actions
    - Create custom GEL ‘hotmenu’ items to call custom GEL functions with one button press



GEL vs. DSS for automation

- GEL
  - Scripts run from within IDE
  - Asynchronous calls of built-in calls
  - Control active control window/core
- DSS
  - Scripts run in command window/shell
  - Synchronous support of built-in calls
  - Control multiple control windows/cores (Parallel Debugging) from one script

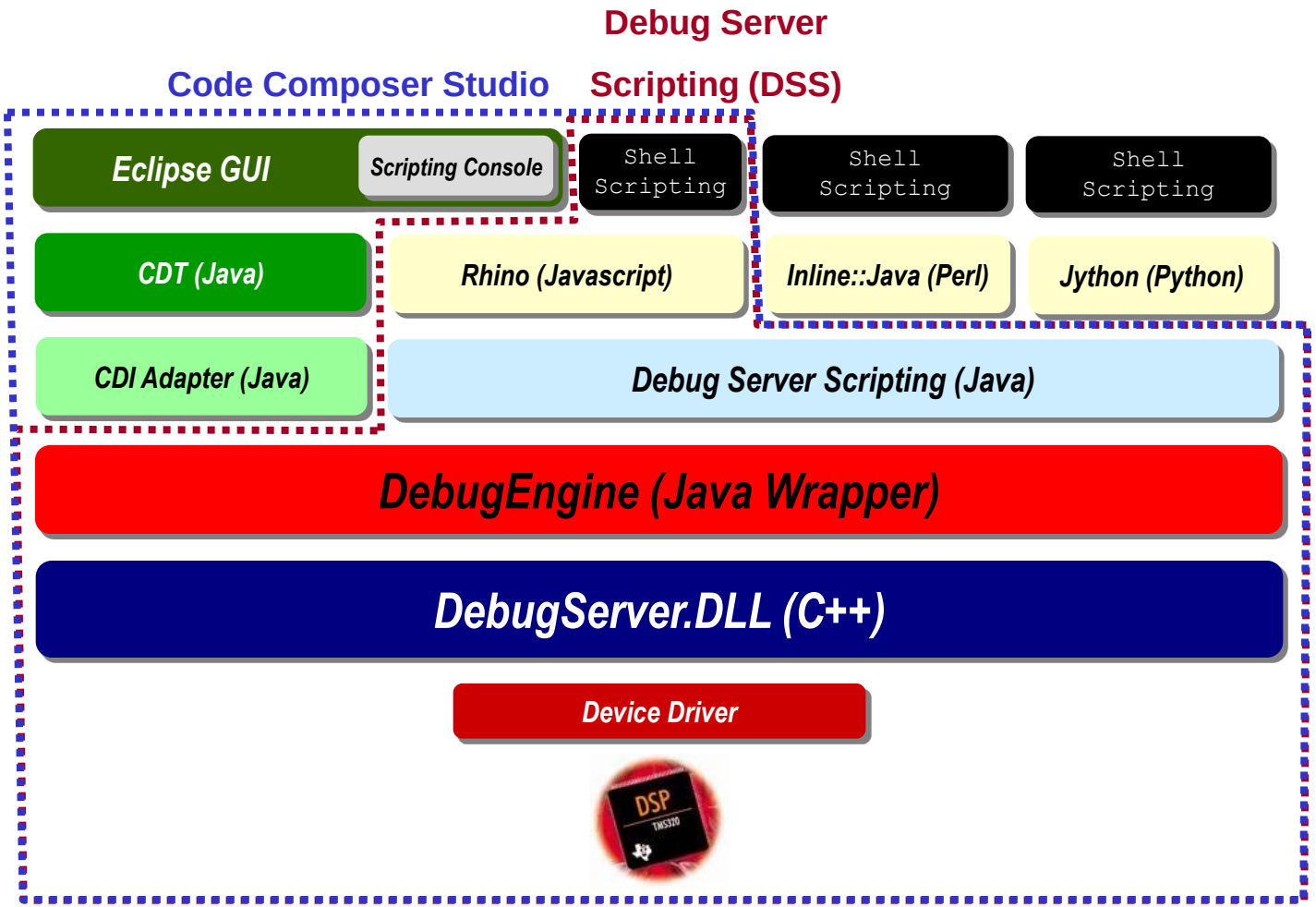
Agenda

- Overview
- GEL(General Extension Language)
- DSS(Debug Server Script)
  - JavaScript
  - CCS scripting Console

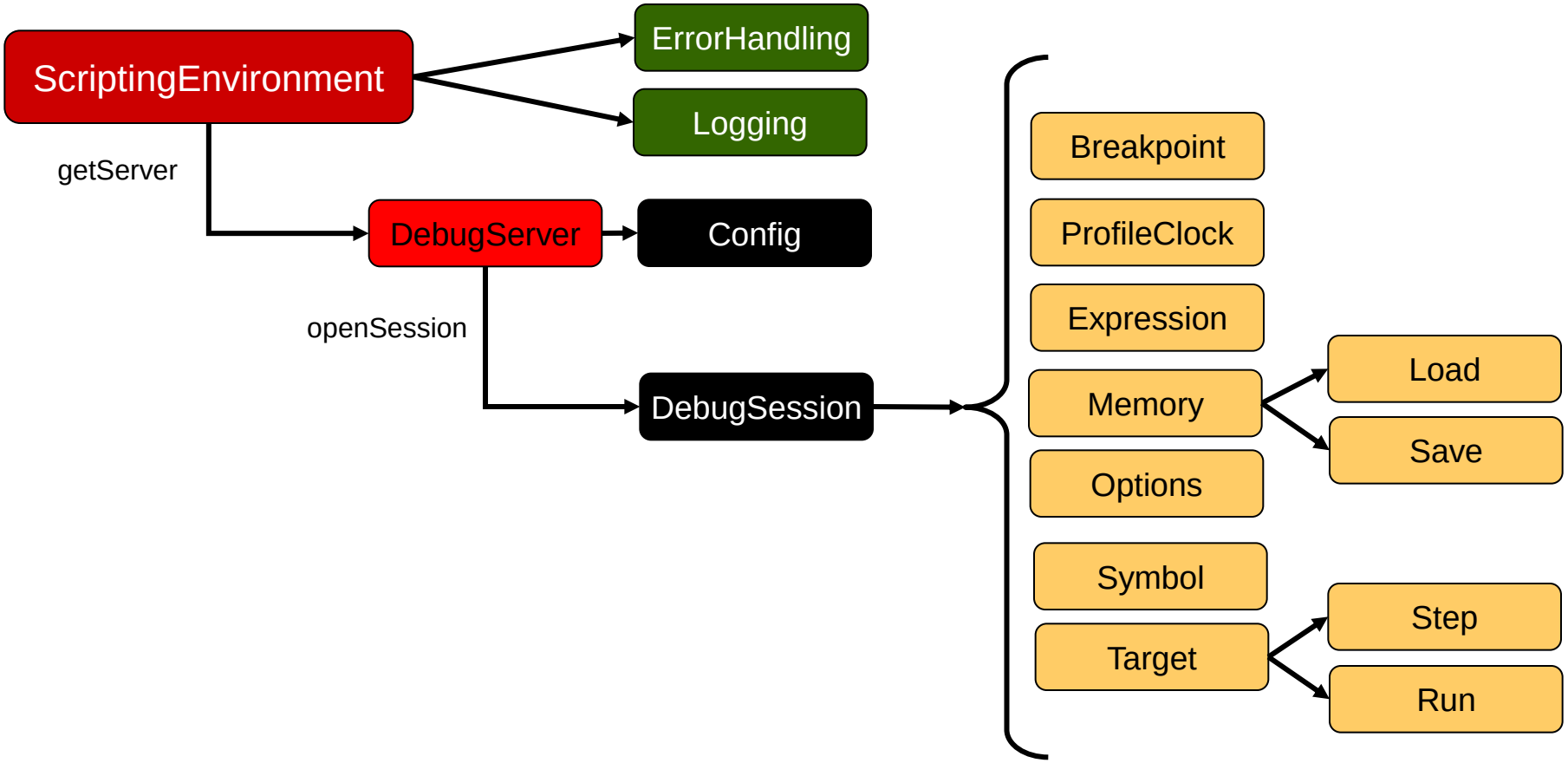
Debug Server Scripting (DSS): Intro

- What is it?
  - Set of Java APIs into the Debug Server (Debugger)
  - Can be used to automate any of the functionality of the debugger
- Scriptable through available 3P tools such as:
  - Javascript (via Rhino) or Java– This is the default language supported by DSS
  - Perl (via “inline::java” module)
  - Python (via Jython)
  - TCL (via Jacl/Tclblend)

DSS: Architecture

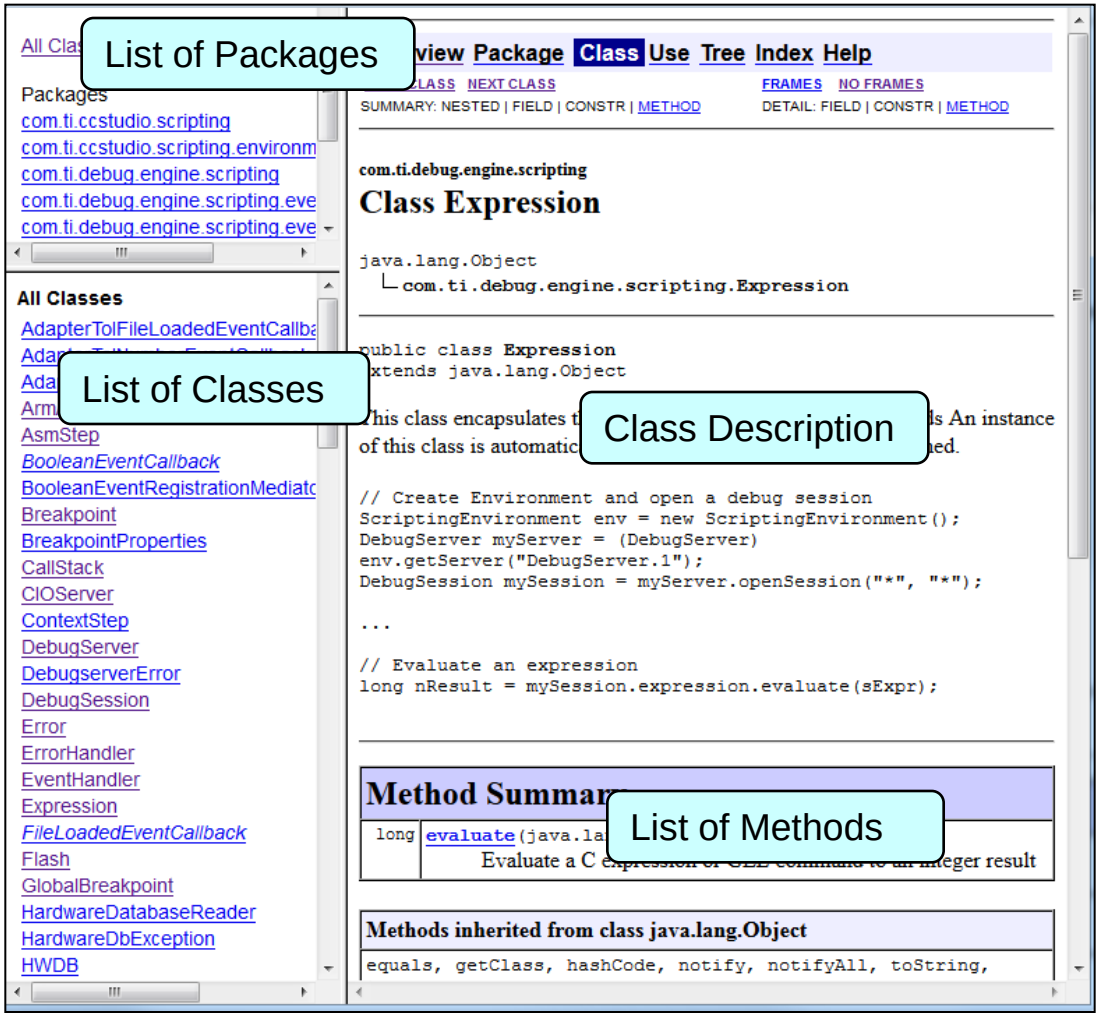


DSS: API Hierarchy (Subset)



DSS API Documentation

- *<CCSv5 INSTALL DIR>\ccsv5\ccs_base\scripting\docs\GettingStarted.htm*



Logging

- XML format (easy to parse)
  - Can specify XSTL file to be referenced in XML log file
    - Configure how log information is displayed when opened in a web browser
      - Compatibility varies with the browser type (**Internet Explorer** has the best compatibility)
    - XSTL Tutorial: http://www.w3schools.com/xsl/xsl_intro.asp
- More information
  - Sequence Numbers
  - Timing information (total, deltas, etc)
  - Status messages
- Log CIO output

DSS Log: Generated XML Log

```
<?xml version="1.0" encoding="windows-1252" standalone="no"?>
<?xml-stylesheet type="text/xsl" href="DefaultStylesheet.xsl"?>
<log>
<record>
  <date>2013-09-15T16:51:50</date>
  <millis>1379278310673</millis>
  <sequence>112</sequence>
  <logger>com.ti</logger>
  <level>FINER</level>
  <class>com.ti.ccstudio.scripting.environment.ScriptingEnvironment</class>
  <method>traceBegin</method>
  <thread>10</thread>
  <message>RETURN</message>
</record>
<record>
  <date>2013-09-15T16:51:50</date>
  <millis>1379278310673</millis>
  <sequence>113</sequence>
  <logger>com.ti</logger>
  <level>FINER</level>
  <class>com.ti.ccstudio.scripting.environment.ScriptingEnvironment</class>
  <method>traceSetFileLevel</method>
  <thread>10</thread>
  <message>ENTRY sLevel: ALL</message>
</record>
...
```

XSTL file to use

DSS Log: Open in Web Browser

Debug Server Log

Total Execution Time: 10489 ms

Sequence	Time (ms)	Delta (ms)	Level	Method	Message
3	0		FINER	traceSetFileLevel	RETURN
4	1	1	INFO	traceWrite	Begin scripting session
5	1	0	FINER	getServer	ENTRY sServerName: DebugServer.1
6	1	0	FINER	getServer	Getting definition for: DebugServer.1
7	9	8	FINER	getServer	Constructing server
8	17	8	FINER	getServer	RETURN com.ti.debug.engine.scripting.DebugServer@1a9dc0e
9	17	0	FINER	setConfig	ENTRY sConfigurationFile: C:/CCSWorkshop/DSS/lab1/StellarisLaunchPad.ccxml
10	18	1	FINER	setConfig	RETURN
11	18	0	FINER	openSession	ENTRY sPattern: .*
12	19	1	FINER	start	ENTRY
13	19	0	FINER	start	Firing: onServerStarting()
14	19	0	FINER	start	Connecting to XPCOM DebugServer
15	205	186	FINER	start	Initializing DebugServer using specified configuration: "C:/CCSWorkshop/DSS/lab1/StellarisLaunchPad.ccxml"
16	206	1	FINER	waitUntil	ENTRY com.ti.ccstudio.scripting.environment.ScriptingEnvironment@2dcc31 timeout: infinite
17	656	450	FINER	<init>	CPU Name: CORTEX_M4_0

JavaScript Syntax Basics (1)

- Operators are ‘C’ like
  - +, -, *, /, %, ++, --, >, <, >=, <=, ==, !=…
- Strings can be concatenated using '+'
  - `print('He' + 'llo'); // displays “Hello”`
- Variables:
  - Have no type attached and any value can be stored in any variable
  - Declared with a **`var`** statement
    - `var x;`
  - Any type of variable can be convert to string automatically
    - `var x=15;`
    - `print(x); // displays “15”`
- Control statements similar to 'C'
  - if, if … else, switch, for, while, etc
- Comments are 'C' like
  - `// This is a JavaScript comment`
  - `/* So is this */`

JavaScript Syntax Basics (2)

- Array

```
var cars=new Array();
cars[0]="Audi";
cars[1]="BMW";
cars[2]="Volvo";
var good_cars=["Audi","BMW","Volvo"];
```

- Structrue or Object

```
var person={firstname:"Bill", lastname:"Gates", id:5566};
```

- Structure array

```
var test_cases = [
     {program: "AIF2_LTE_FDD.out", timeOut:   20000},
     {program: "HyperLink.out"   , timeOut:  300000},
     {program: "Memory_Test.out" , timeOut: 1000000},
     {program: "UART.out"        , timeOut:   20000}];
var this_program= test_cases[0].program;
var its_timeOut = test_cases[0].timeOut;
```

JavaScript Syntax Basics (3)

- Functions can be defined with the **function** keyword
  - `function doThis(x) { y=x; }`
- Call methods via <object>.<method>
  - Ex: `debugServer.openSession(".*");`
- Send messages to the console with the `print()` method
  - `print("Hello!");`
- Does not support:

  #define DEBUG 1

  #if DEBUG…#endif

  Should use:

  var DEBUG =1;

  if (DEBUG) {…};
- File path separator can be “/” or “\\” instead of “\”
  - var target_board_cfg= "C:/Users/a0389163/ti/CCSTargetConfigurations/K2K_EVM.ccxml“
  - var target_board_cfg= "C:\\Users\\a0389163\\ti\\CCSTargetConfigurations\\K2K_EVM.ccxml"

Creating a DSS script (Basic Debug)

1. Import DSS and Java Packages
2. Create the Scripting Environment Object
3. (Optional) Enable and configure logging
4. Get a handle to the debug server (Start the Debugger)
5. Configure the debugger for desired target
6. Open a debug session
7. Connect to the target
8. Execute any desired debug actions (load program, run, halt, set breakpoints, etc)
9. Terminate Debugger
10. (Optional) Disable logging

1. Import DSS and Java Packages

- Import the relevant packages
  - DSS
    - **`importPackage(Packages.com.ti.debug.engine.scripting)`**
    - **`importPackage(Packages.com.ti.ccstudio.scripting.environment)`**
  - Java (if needed)
    - **`importPackage(Packages.java.lang)`**
    - **`importPackage(Packages.java.io)`**
    - etc

```
importPackage(Packages.com.ti.debug.engine.scripting);
importPackage(Packages.com.ti.ccstudio.scripting.environment);
importPackage(Packages.java.lang);
importPackage(Packages.java.io)
```

2. Create the Scripting Environment Object

- Create the Scripting Environment Object
  - **`ScriptingEnvironment.instance()`**

```
var script = ScriptingEnvironment.instance();
```

3. Enable and Configure Logging

- Enable Logging to a file
  - **`traceBegin(<output file name>, <stylesheet>)`**
    - Example stylesheet provided in: *<CCS INSTALL DR>\ccsv5\ccs_base\scripting\examples\DebugServerExamples\DefaultStylesheet.xsl*
- Configure Logging
  - For the log file
    - **`traceSetFileLevel(<TRACE LEVEL>)`**
  - For the console messages
    - **`traceSetConsoleLevel(<TRACE LEVEL>)`**

```
script.traceBegin("C:/mypath/log_file.xml", "C:/mypath/DefaultStylesheet.xsl");

script.traceSetConsoleLevel(TraceLevel.INFO);
script.traceSetFileLevel(TraceLevel.ALL);
```

Message Verbosity Levels (Trace Level)

- In order from least verbose to most
  - **`TraceLevel.OFF`** (turn off logging)
  - **`TraceLevel.SEVERE`** (only very severe messages from the debugger)
  - **`TraceLevel.WARNING`** (**`TraceLevel.SEVERE`** + warning messages from the debugger)
  - **`TraceLevel.INFO`** (**`TraceLevel.WARNING +`** basic and C I/O)
    - script_env.traceWrite() output is INFO level
  - **`TraceLevel.ALL`** (log everything)

traceWrite() and *print()*

- There is a DSS API called **`traceWrite(<string>)`** that allows you to send "printf" type messages to both the log and console
  - **`script.traceWrite("Hello");`**
  - JavaScript **`print()`** API would only go to the console

4. Get a Handle to the Debug Server

- Get a handle to the debug server
  - **`getServer("DebugServer.1")`**

```
var debugServer = script.getServer("DebugServer.1");
```

5. Configure the Debugger for the Target

- Configure the debugger for desired target by passing in a target configuration *.ccxml file to the debug server
  - **setConfig(<ccxml file>)**

```
debugServer.setConfig(" C:/Users/ti/CCSTargetConfigurations/target.ccxml");
```

  - **Normally, *.ccxml file is created in CCS target configuration dialogs.**

6. Open a Debug Session

- Open a debug session to a CPU
  - **`openSession()`**: Assumes a single board/CPU configuration and opens a connection to it
  - **`openSession(<regular expression>)`**: Uses a regular expression to match a board/CPU configuration
  - **`openSession(<board name>,<CPU name>)`**: Specify the board name and the CPU name to open a connection for

```
var debugSession = debugServer.openSession("*", "*");
```

  - A wildcard "*" will match the first found target

- NOTE: Multiple debug sessions can be opened for multi-core targets (up to one debug session for each CPU)

7. Connect to the Target

- Connect to the target
  - **`target.connect()`**

```
debugSession.target.connect();
```

8. Execute Desired Debug Actions

- The “main” section of the script – execute all debug actions desired
  - Load program: **`memory.loadProgram(<program>)`**
  - Run program: **`target.run()`**
  - Set breakpoints on symbol: **`breakpoint.add(<symbol>)`**

```
debugSession.memory.loadProgram("C:/mypath/myfile.out");
debugSession.target.run();
```

  - Debug Server simultaneous functions can run-control actions on multiple targets simultaneously.

```
debugServer.simultaneous.run();
```

9. Terminate Debugger

- Shutdown the debugger after all debug actions are complete
  - **`debugServer.stop()`** – this call will terminate both the debug session and shut down the debug server

```
debugServer.stop();
```

10. Disable Logging

- Stop and disable logging (this is needed to properly close the log file)
  - **`script.traceEnd()`**

```
script.traceEnd();
```

Exception Handling

- DSS APIs throw Java exceptions
- If you do not catch exception, the script will abort on any exception.
- You can handle the exception in the script to fail gracefully or continue on (ex. Javascript try-catch)

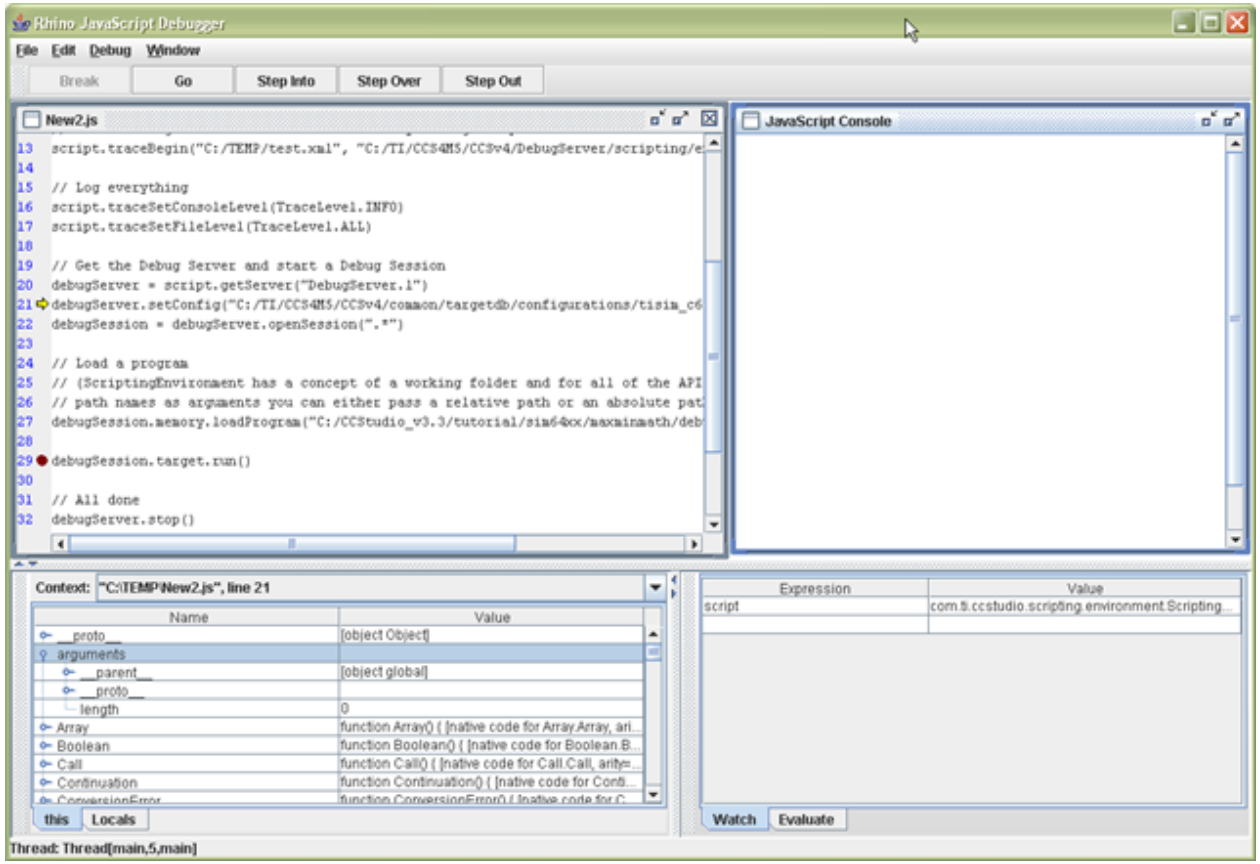
```
try {
    debugSession.memory.loadProgram(testProgFile);
} catch (ex){
    dssScriptEnv.traceWrite("Program load of " +
        testProgFile + " failed!");
}
```

Running a DSS JavaScript

- Run your DSS script using the supplied DSS shell script
  - **dss.bat** (DOS)
  - **dss.sh** (Linux)
- DSS shell script will setup the environment needed to run DSS and then execute the DSS JavaScript
- Located in <*CCSv5 INSTALL DIR*>*ccsv5\ccs_base\scripting\bin*
- Usage:
  - **`> dss.bat [-dss.debug] <dss javascript>`**

Debug Scripts (Rhino Debugger)

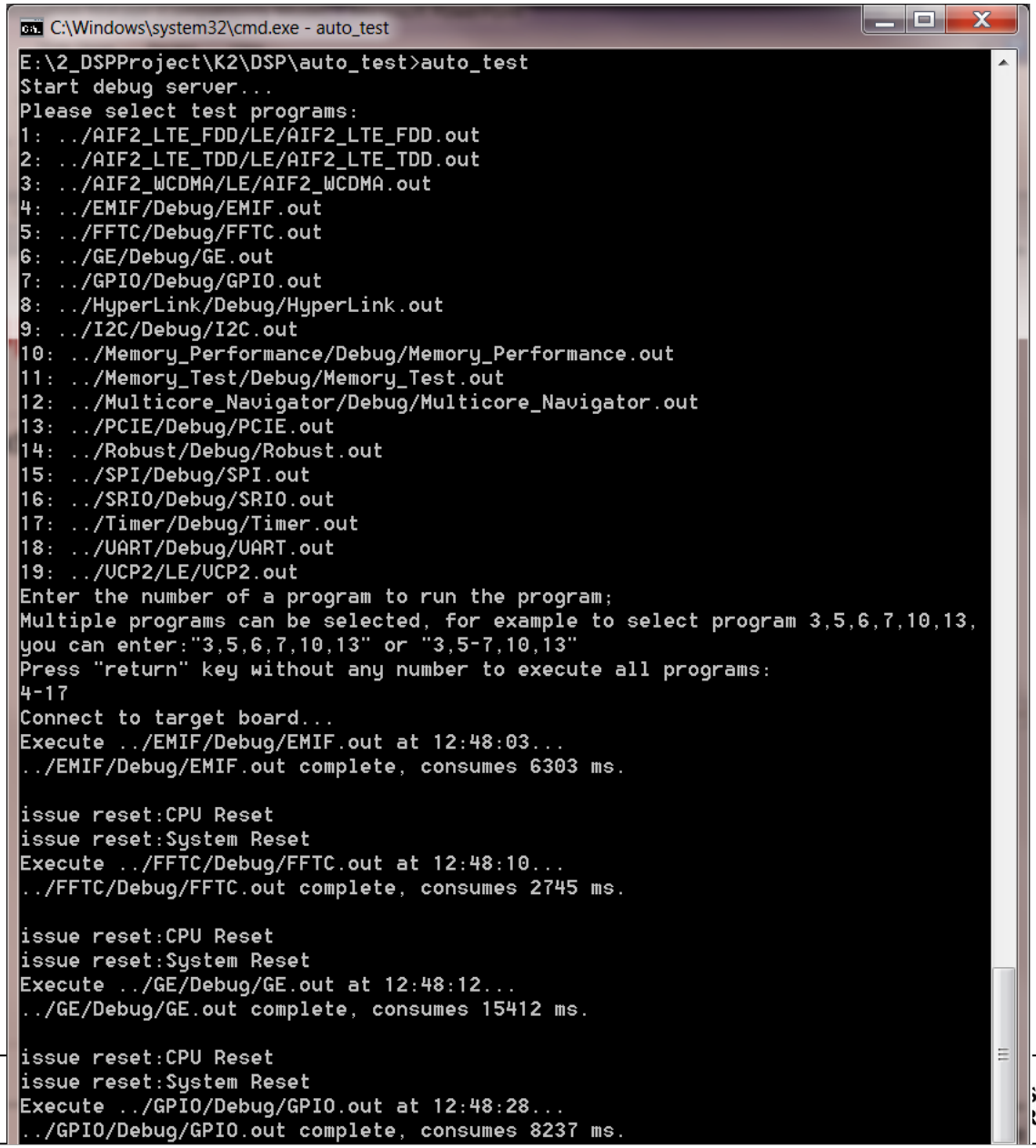
- Rhino Debugger, which comes with DSS, supports:
  - Single stepping, Breakpoints, viewing script variables…



Using the Rhino Debugger

- Pass in the **`-dss.debug`** parameter when calling the dss script via **dss.bat**
  - **`> dss.bat -dss.debug myScript.js`**

DSS Example: K2 STK automation test



DSS Example JavaScript (1)

```
// Modify these variables to match your environment. Use forward slashes
//the configuration file for the target board to run the test programs
var target_board_cfg= "C:/Users/a0389163/ti/CCSTargetConfigurations/K2K_EVM.ccxml"
//list of test programs and timeout value in ms for execute the program
var test_cases = [
    {program: "../AIF2_LTE_FDD/LE/AIF2_LTE_FDD.out"   , timeOut:   30000},
    {program: "../GE/Debug/GE.out"                    , timeOut:   40000},
    {program: "../Memory_Test/Debug/Memory_Test.out" , timeOut: 1000000},
    {program: "../PCIE/Debug/PCIE.out"                , timeOut:  150000},
    {program: "../Robust/Debug/Robust.out"            , timeOut:  150000},
    {program: "../SPI/Debug/SPI.out"                  , timeOut:  500000},
    {program: "../SRIO/Debug/SRIO.out"                , timeOut:  700000},
    {program: "../UART/Debug/UART.out"                , timeOut:   20000},
    ……
    {program: "../VCP2/LE/VCP2.out"                   , timeOut:   30000}];

// Import the DSS packages into our namespace to save on typing
importPackage(Packages.com.ti.debug.engine.scripting);
importPackage(Packages.com.ti.ccstudio.scripting.environment);
importPackage(Packages.java.lang);
importPackage(Packages.java.io);
```

DSS Example JavaScript (2)

```
// Create our scripting environment object - which is the main entry point
// into any script for creating other Scriptable Servers and Sessions
var script = ScriptingEnvironment.instance();

// Create a log file in the current directory to log script execution
script.traceBegin(logFile, "DefaultStylesheet.xsl");

// Set trace levels for console and logs
script.traceSetConsoleLevel(TraceLevel.INFO);
script.traceSetFileLevel(TraceLevel.ALL);

script.traceWrite("Begin scripting session");

// Get the Debug Server and start a Debug Session
var debugServer = script.getServer("DebugServer.1");
debugServer.setConfig(target_board_cfg);
var debugSession = debugServer.openSession("*", "*");

// Connect to the CPU
debugSession.target.connect();
```

DSS Example JavaScript (3)

```
for(var i = 0; i < num_test_case; i++)
{
        ……
    // Change timeout from the default (infinite)
    script.setScriptTimeout(timeOut);
    //save CIO of the program to a text file.
    debugSession.beginCIOLogging(CIO_log_file_name);
    try{
        // Load a program
        debugSession.memory.loadProgram(program);
        // Run the target
        debugSession.target.run();
    }catch(exception)
    {//catch exception and continue to execute the next program
        exceptionCount++;
    }
    debugSession.endCIOLogging();
    //advanced reset to clean the device for running next test case
    AdvancedReset();
        ……
}
```

DSS Example JavaScript (4)

```
// All done
debugServer.stop();

script.traceWrite("End scripting session");

// Stop logging and exit.
script.traceEnd();
```

Run DSS with Java

- Because DSS is implemented as a set of Java APIs, it works quite nicely out of the box with Java.
- JavaScript is executed with Rhino, which is installed with CCS by default.
- Java code for DSS need be compiled and executed with JDK Version 7.4 or above, which need be installed separately from the Sun Java SE download site

DSS example with Java

```
import com.ti.ccstudio.scripting.environment.*;
import com.ti.debug.engine.scripting.*;
public static void main(String[] args)
{
    ScriptingEnvironment env = ScriptingEnvironment.instance();
    DebugServer debugServer = null;
    DebugSession debugSession = null;
    try { // Get the Debug Server and start a Debug Session
        debugServer = (DebugServer) env.getServer("DebugServer.1");
        debugServer.setConfig(tisim_c64xple.ccxml");
        debugSession = debugServer.openSession(".*");
         // Load a program
        debugSession.memory.loadProgram("modem.out");
    } catch (Exception e)
    {
    …
```

Steps to run DSS based on java code

1. Set path

   > set PATH=%PATH%;"C:\ti\ccsv5\ccs_base\common\bin;"C:\ti\ccsv5\ccs_base\common\uscif

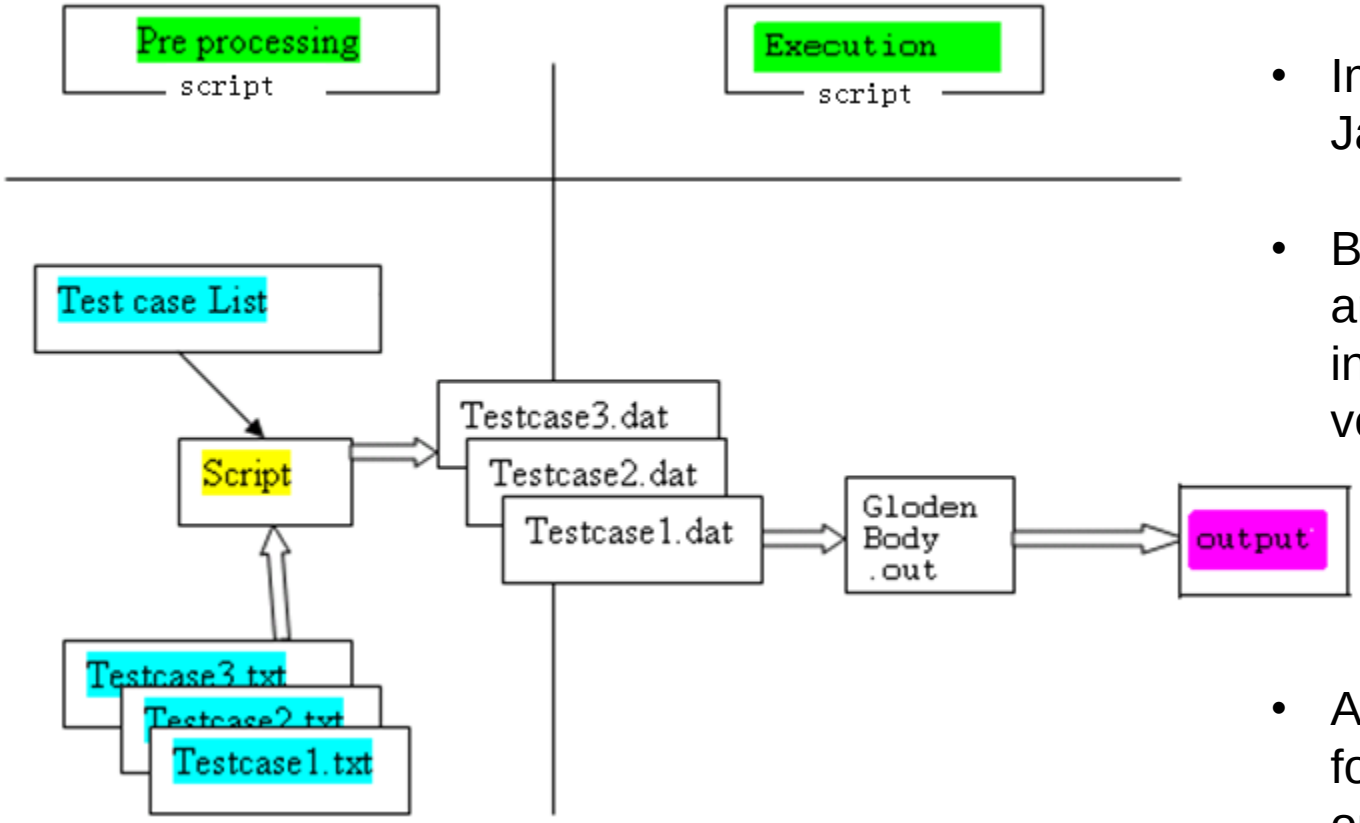
2. Compile the *.java code

   > javac -cp "C:\ti\ccsv5\ccs_base\DebugServer\packages\ti\dss\java\com.ti.ccstudio.scripting.environment_3.1.0.jar";C:\ti\ccsv5\ccs_base\DebugServer\packages\ti\dss\java\com.ti.debug.engine_1.0.0.jar";"C:\ti\ccsv5\ccs_base\DebugServer\packages\ti\dss\java\dss.jar" DssExample.java

3. Execute the Java code

   > java -cp C:\DSS_Java_Example;"C:\ti\ccsv5\ccs_base\DebugServer\packages\ti\dss\java\dss.jar";"C:\TI\CCSv5\ccsv5\ccs_base\DebugServer\packages\ti\dss\java" DssExample

DSS example: K2 AVV automation Test



- Implemented with Java code.
- Besides program load and run, the script inject input data, and verify output data.
- All test cases must follow same input and output rules.

Agenda

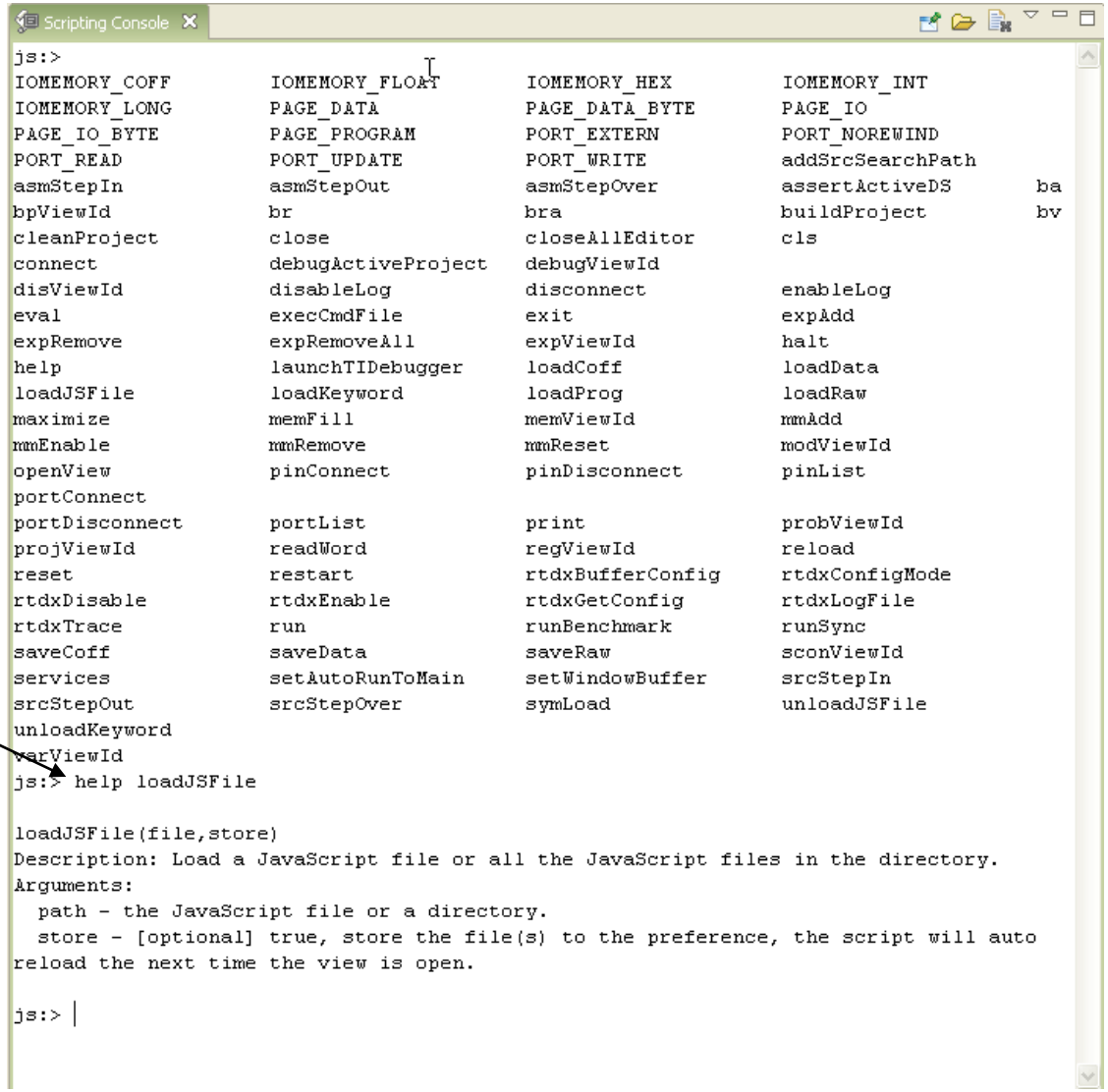
- Overview
- GEL(General Extension Language)
- DSS(Debug Server Script)
  - JavaScript
  - CCS scripting Console

Scripting Console in CCS

- Command line operation of CCS
- *View->Scripting Console*
- Press TAB for a list of commands
  - Press TAB for partially typed commands for auto-complete feature
- To get documentation for a command
  - **`js:> help <command>`**
- JavaScript shell and has access to all DSS APIs
- Run DSS scripts from the console
- Create your own custom commands
  - Create a JavaScript function in a *.js file
  - Load the custom Javascript file
    - loadJSFile <full path>/myCustomConsoleCmd.js
    - Optional boolean second parameter that will auto-load the script

Scripting Console

- *View- > Scripting Console*
- Press TAB for a list of commands
  - Press TAB for partially typed commands for auto-complete feature
- To get documentation for a command
  - **`js:> help <command>`**



Scripting Console

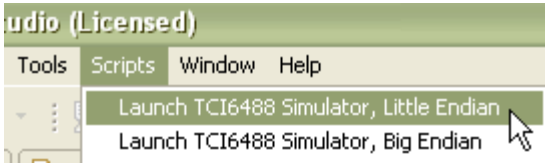
- Both the Scripting Console and GEL can be used for automation
- GEL can be used only within an active debug session and (mostly) apply to a debug context
- The Scripting Console can be used anytime (though certain commands will not work without a debug session)
- Scripting Console and GEL can both add menus to the custom ‘Scripts’
  - GEL: **`hotmenu <function>`**
  - Scripting Console: **`hotmenu.addJSFunction`**

Scripting Console – Example Script

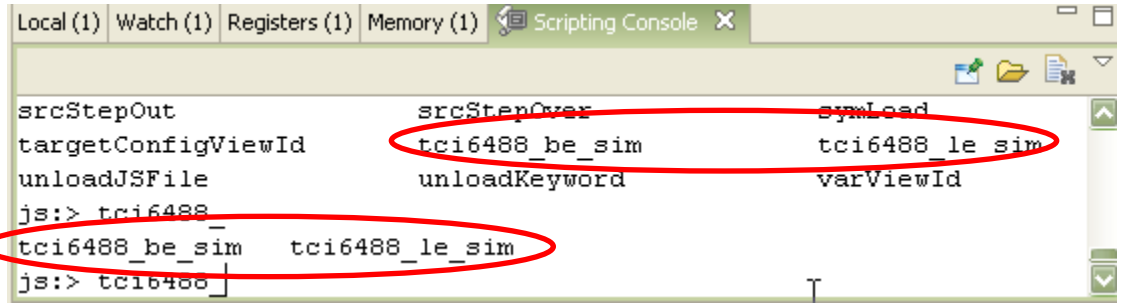
```
// Add entries to the 'Scripts' menu
hotmenu.addJSFunction("Launch TCI6488 Simulator, Little Endian",
    "tci6488_le_sim()");
hotmenu.addJSFunction("Launch TCI6488 Simulator, Big Endian",
    "tci6488_be_sim()");

// Path to folder with target setup ccxml files
var setupConfigFileFolder = "C:/Documents and
    Settings/login/user/CCSTargetConfigurations";

// configure for a TCI6488 Symmetric Simulator, Little Endian
function tci6488_le_sim()
{
    ds.setConfig(setupConfigFileFolder + "/tci6488_le_sim.ccxml");

    debugSessionCPU1 = ds.openSession("*", "C64+_0");
    debugSessionCPU2 = ds.openSession("*", "C64+_1");
    debugSessionCPU3 = ds.openSession("*", "C64+_2");
}
```

Scripting Console

- Scripts menu with added console commands



- Console commands added to list of supported console



DSS tips

- CCS can be opened when running DSS as long as CCS does not connect target.
- On PC's with multiple CCStudio installs, the scripting utility will invoke the debug server of CCStudio that was last launched.
- A running script can be terminated with Ctrl+C in command window.
- DSS can not automate build CCS projects. However, there are other utilities that comes with CCS for command-lines builds and these utilities can be called from a script to automate builds: http://processors.wiki.ti.com/index.php/Projects_-_Command_Line_Build/Create

DSS: Resources

- DSS MediaWiki documentation (recommended reading):
  - http://processors.wiki.ti.com/index.php/Debug_Server_Scripting
- DSS API documentation:
  - *<CCSv5 INSTALL DIR>\ccsv5\ccs_base\scripting\docs\GettingStarted.htm*
- Java API on-line help
  - *http://docs.oracle.com/javase/6/docs/api/overview-summary.html*

Q&A