Advanced Debug Capabilities of CCE

Darian Sale

Software Development Organization

Objectives

• Introduction to advanced features of CCE
  – Debug Server Scripting (DSS)
  – 3rd party Eclipse plug-ins
  – Other advanced Eclipse features
    • Perspectives and workspaces
    • Local history
    • Refactoring
    • Advanced breakpoint features
DSS: What is it?

- Java API’s to CCE debugger
- Lightweight package with no GUI
  - Faster, as there is no GUI to refresh/repaint/update
  - Errors handled via exceptions, not dialogs
  - However, specific GUI actions cannot be automated
- Scriptable through available 3P tools such as:
  - Javascript (via Rhino)
  - Perl (via “inline::java” module)
  - Python (via Jython)
  - TCL (via Jacl/Tclblend)
Software Development Organization

DSS: What can you use it for?

- Automate testing
  - Use JUnit, tcltest etc
- Automate profile runs overnight
- Type commands instead of clicking a mouse
- Automate a sequence of complex steps
  - Quickly place the target application into a “broken” state for debugging

DSS: API’s

Detailed API documentation is included:
<install>\DebugServer\scripting\docs
DSS: How to use it

- Used from command line or Scripting Console
  - The Scripting Console can be opened from the Window->Show View menu and executes javascript

DSS: Scripting Console

- Scripting Console creates a session variable `dss`
  - session opened from `dss` links with active debug session

Target starts to run

Run Issued
Software Development Organization

**DSS: Scripting Console**

- Terminating the session will terminate the CCE debug session

  ![Terminate called](image)

  ![Session terminates](image)

  **Terminate called**

  **Session terminates**

---

Use `loadCommandFile` to run a more complex script

- Script will be run at every startup

```c
1 // Set the active session
2 session = data.openSession();
3
4 // Start the target
5 session.target.reset();
6
7 // Set breakpoint at the end of main
8 session.breakpoint.add("main.c", 55);
9
10 // Run the target
11 session.target.run();
```
Software Development Organization

**DSS: Scripting Console**

- Place commonly used functions in a script
  - Allows easy keyboard control of the debugger

```javascript
// Breakpoint add at a specific address
function add(address) {
    session.breakpoints.add(address);
}

// Run the target
function run() {
    session.target.run();
}
```

**DSS: Command line**

- DSS can also be used from the command line
- In the CCE install folder, go to `\DebugServer\scripting\examples`
  - This folder contains rhino.bat, which will run any DSS javascript-based file
  - The folder also contains some example DSS scripts
Software Development Organization

DSS: Sample script

// 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 );

// Create Scripting Environment
var env = new ScriptingEnvironment();

// Begin logging - log *everything* to a file and INFO messages to the console
env.traceBegin( "dsslog.xml" );
env.traceSetFileLevel( TraceLevel.ALL );
env.traceSetConsoleLevel( TraceLevel.INFO );

// Open a debug session
var server = env.getServer( "DebugServer.1" );
var session = server.openSession();

// Set start & stop profiling BPs
session.breakpoint.add( "simple.c", 5 );
session.breakpoint.add( "simple.c", 20 );

// Run to first (start) breakpoint
session.target.run();

// Run to second breakpoint and count cycles
var profileServer = env.getServer( "ProfileServer.1" );
var profileSession = profileServer.openSession( session );
var cycleCount = profileSession.runBenchmark();

// Close our Session and Server
session.terminate();
server.stop();
DSS: 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
    - XSTL Tutorial: [http://www.w3schools.com/xsl/xsl_intro.asp](http://www.w3schools.com/xsl/xsl_intro.asp)

- More information
  - Sequence Numbers
  - Timing information (total, deltas, etc)
  - Status messages

- Log printf output

### Debug Server Log

<table>
<thead>
<tr>
<th>Sequence</th>
<th>Time (ms)</th>
<th>Delta (ms)</th>
<th>Level</th>
<th>Method</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>FINEX</td>
<td>transferDisplayLevel</td>
<td>RETURN</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>FINEX</td>
<td>getUrlServerName</td>
<td>LegacyServer1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>FINEX</td>
<td>getUrlServerName</td>
<td>LegacyServer1</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>10</td>
<td>FINEX</td>
<td>getUrlServerName</td>
<td>Constructing server</td>
</tr>
<tr>
<td>4</td>
<td>10</td>
<td>0</td>
<td>FINEX</td>
<td>getUrlServerName</td>
<td>Sharding server</td>
</tr>
<tr>
<td>5</td>
<td>30</td>
<td>10</td>
<td>FINEX</td>
<td>start</td>
<td>ENTRY</td>
</tr>
<tr>
<td>6</td>
<td>50</td>
<td>30</td>
<td>FINEX</td>
<td>start</td>
<td>RETURN</td>
</tr>
<tr>
<td>7</td>
<td>50</td>
<td>0</td>
<td>FINEX</td>
<td>getUrlServerName</td>
<td>exit() debug.c yourLogServerDebug(R55948)</td>
</tr>
<tr>
<td>8</td>
<td>50</td>
<td>0</td>
<td>FINEX</td>
<td>exit() coreConfigSave</td>
<td>ENTRY</td>
</tr>
<tr>
<td>9</td>
<td>50</td>
<td>0</td>
<td>FINEX</td>
<td>coreConfigSave</td>
<td>Creating system setup object</td>
</tr>
<tr>
<td>10</td>
<td>2065</td>
<td>2015</td>
<td>FINEX</td>
<td>coreConfigSave</td>
<td>Clearing configurations</td>
</tr>
<tr>
<td>11</td>
<td>4165</td>
<td>1301</td>
<td>FINEX</td>
<td>coreConfigSave</td>
<td>Saving system configuration</td>
</tr>
<tr>
<td>12</td>
<td>2064</td>
<td>2044</td>
<td>FINEX</td>
<td>coreConfigSave</td>
<td>RETURN</td>
</tr>
<tr>
<td>13</td>
<td>2060</td>
<td>0</td>
<td>FINEX</td>
<td>coreConfigReport</td>
<td>ENTRY</td>
</tr>
</tbody>
</table>
Software Development Organization

DSS: Exception Handling

```java
try {
    debugSession.memory.loadProgram(testProgFile);
} catch (ex){
    errCode = dssErrorHdl.getLastErrID();
    dssScriptEnv.traceWrite("errCode: " + errCode + " - " + testProgFile + " does not exist!");
    quit();
}
```

- DSS APIs throw Java exceptions
- Can handle exception in the script to fail gracefully or continue on

ATC 2008

MSP430 Advanced Technical Conference

Software Development Organization

DSS: Example

- Demo
  - Use DSS to automate testing of an MSP430 application
Software Development Organization

Eclipse background

- Eclipse
  - Originally IBM’s Java Development Environment
  - Open platform for tool integration built by an open community of tool providers
  - Operates under a open source paradigm
  - Universal platform for tools integration
  - Multi-language, multi-platform and multi-vendor environment
  - Excellent plug-in framework

- Governed by Eclipse Public License (EPL)
  - Royalty free world-wide redistribution rights
  - Enables you to sell components & products
  - Source changes to EPL governed code need to be made available

- Eclipse Foundation
  - The Eclipse Foundation is a non-profit corporation formed to advance the creation, evolution, promotion, and support of the Eclipse Platform and to cultivate both an open source community and an ecosystem of complementary products, capabilities, and services.

- Other Eclipse members
  - IBM/Rational, QNX, WindRiver, Borland, RedHat, MontaVista, Texas Instruments

Software Development Organization

Benefits of Eclipse

- Excellent Application Framework
  - Modular and extensible

- Rapid Rate of Improvement
  - Many companies are contributing

- Development tool integration
  - Users need more than TI tools to do their development
    - Design tools, Code development, OS awareness
    - www.eclipseplugincentral.com
    - www.eclipse.org/community/plugins.php
  - Eclipse is a standard platform to integrate with
    - Outstanding SDK and plug-in framework

- Host platform independence
  - Windows, Linux…

- TI can focus on unique value-added capabilities

ATC 2008

MSP430 Advanced Technical Conference

ATC 2008

MSP430 Advanced Technical Conference
Software Development Organization

CCE architecture

Eclipse
User Plug-ins
C/C++ Development Toolkit
3rd Party Plug-ins
CDI Adapter
TI Plug-ins
DSS Scripting
Debugger

ATC 2008
MSP430 Advanced Technical Conference

Software Development Organization

Eclipse Plug-ins

• Well documented API enables 3rd parties
  – Many Eclipse add-ons available
    • Editors
    • Source control
    • Testing
    • Documentation
    • UML

• Caveat
  – Not all plugins will work in CCE, or work together
Example - Terminal plug-in

- CCE ships with the Terminal View plug-in
  - Available from the Other... menu in Show View

Terminal plug-in

- Supports COM port connection to your board
Software Development Organization

**Where to get additional plug-ins**

- Eclipse.org community
  - Eclipse Plug-in Central
    - Browse through plug-ins
    - [www.eclipseplugincentral.com](http://www.eclipseplugincentral.com)
  - Older Eclipse Plug-in Site
- If you still can’t find what you want Google usually can

---

**Installing plug-ins**

- As simple as copying files into your eclipse\plugins and eclipse\features folders
  - Typically distributed as a zip file
  - Shutdown Eclipse, unzip, start Eclipse
- CCE 3.0 uses Eclipse v3.2.0 and CDT v3.1
  - Plug-ins not designed for this version may not operate correctly
  - CDT plug-ins are unlikely to work at all due to modifications to it in CCE
- Do not update any core plug-ins
  - Overwriting these plug-ins will remove MSP430 modifications
Software Development Organization

**Plug-in example**

- **Demo**
  - Installing Java support to CCE
  - Running DSS automated tests from within CCE

---

Software Development Organization

**Plug-in Development Environment**

- **Eclipse “Plug-in Development Environment” (PDE)**
  - Used to create, test, debug, and deploy plug-ins
- **Plug-ins are written in Java**
  - “Java Development Toolkit” or JDT must be installed
  - Available from [http://www.eclipse.org](http://www.eclipse.org)
Software Development Organization

PDE perspective

• Window->Open Perspective->Other
  – Select Plug-in Development

Plug-in development references

• Several different templates provided
• Tutorials available in the help and on the web
• A number of books available
Workspaces

- What are they?
  - A workspace is a folder that contains information relevant to what you are working on
- A workspace contains
  - Projects
    - Either physically in the workspace or referenced by the workspace
  - Window and toolbar arrangements
  - Preference settings
- You can switch between multiple workspaces
- Automatically saved on exit
- Can be automatically loaded at startup

Perspectives

- Background:
  - Each Workbench window contains one or more perspectives
  - A perspective defines the initial set and layout of views
  - Each perspective is aimed at accomplishing a specific type of task
- As you work in the Workbench, you will probably switch perspectives frequently
- Customization
  - Perspectives control what appears in certain menus and toolbars
  - They define visible action sets, which you can change to customize a perspective
  - You can save a custom perspective that you build
- Default perspectives in CCE
  - C/C++: Editing and building
  - Debug: Debugging your programs
- Perspectives can be opened in the same window, or a new one
Software Development Organization

Window Types

- **Detached:** Can float outside of the main window

- **Fast View:** Hidden until you click on the button to restore them. Click on another window to hide

- **Tab Group:** Several windows grouped together

ATC 2008

Software Development Organization

Windowing Tips

- Double-clicking a window’s title bar will maximize it
  - Double-clicking again will restore it to its previous size
- Fast-view windows are great for windows used infrequently which require a lot of space
- The window that has focus is indicated by a blue border and heading

ATC 2008
Customizing perspectives

- You can customize the menu items and toolbars in your perspective
- Commands
  - Controls menus & toolbars
- Shortcuts
  - Controls special sub-menus (new, show view…)

Accessing views

- To open a new view go to the Windows -> Show View Menu
  - Common views for the current perspective
  - Recently opened views
- To access views that are not listed select Other…
**Software Development Organization**

**Editor features**
- Code Completion (ctrl + space)
  - Complete word
  - Auto-member information
  - Auto-parameter information
  - ...
- Navigation
  - Back/Forward buttons
  - Back to last edit button
  - Go to definition
  - Go to declaration
- Show line numbers
- Code Folding
  - Collapse functions

**Local history**
- Eclipse keeps a local history of source changes
- You can compare or replace your current source file with any previous saved version
Breakpoint tips

- Breakpoints are more than just the pre-made selections in the drop down
  - Right clicking on one can bring up a properties display
    - And additional triggers together
    - Define a condition on which to halt
    - Perform actions when it’s hit such as saving memory to a file, or writing memory from a file

Breakpoint tips

- Have a set of breakpoints for a specific purpose?
  - You can save one or more breakpoints to a file to use later in another workspace, or even share with other users
Software Development Organization

Detect invalid branch

• Fill memory with 0x4343
• Load code with the option Retain Unchanged Memory
  – 0x4343 is the opcode used for software breakpoints
  – If the program branches to invalid memory, it will halt

- Trace, if available, can detect where the faulty branch occurred

Additional References
Software Development Organization

**DSS: Target setup**

- DSS used from the Scripting Console uses the active debug session automatically
- DSS used from the command line requires you to indicate what the target is setup

Choose the **Setup tab**

Select **TI Debug Settings**

Click **Export**

Save to `<install>\DebugServer\bin\win32\SystemSetup.xml`

---

**DSS: Generated xml-based log**

```
<?xml version="1.0" encoding="windows-1252" standalone="no"?>
<?xml-stylesheet type="text/xsl" href="SimpleTransform.xsl"?>
<log>
<record>
   <date>2007-05-02T15:30:15</date>
   <millis>1178134215917</millis>
   <sequence>0</sequence>
   <logger>com.ti</logger>
   <level>FINER</level>
   <class>com.ti.ccstudio.scripting.environment.ScriptingEnvironment</class>
   <method>traceSetConsoleLevel</method>
   <thread>10</thread>
   <message>RETURN</message>
</record>
<record>
   <date>2007-05-02T15:30:15</date>
   <millis>1178134215917</millis>
   <sequence>1</sequence>
   <logger>com.ti</logger>
   <level>FINER</level>
   <class>com.ti.ccstudio.scripting.environment.ScriptingEnvironment</class>
   <method>getServer</method>
   <thread>10</thread>
   <message>ENTRY sServerName: LegacySetupServer.1</message>
</record>
```

---

**ATC 2008**

MSP430 Advanced Technical Conference
DSS: What about Perl?

- Is it possible?
  - YES
- Is it complicated?
  - NO
- Required Components:
  - Perl 5.8.3 or higher
    - [http://www.activestate.com/Products/ActivePerl/?tn=1](http://www.activestate.com/Products/ActivePerl/?tn=1)
  - Java 2 SDK (for javac)
  - Inline::Java
    - [http://theoryx5.uwinnipeg.ca/ppms/Inline-Java.ppd](http://theoryx5.uwinnipeg.ca/ppms/Inline-Java.ppd)

DSS: Inline::Java for Perl

- What?
  - DSS APIs are in Java. Inline::Java perl module lets us access Java classes from perl
    - *Perl Inline::Java method interfaces directly to the supported DSS Java APIs*
    - A Java compiler is launched and the Java code is compiled.
    - The Java classes and methods are available to the Perl program as if they had been written in Perl.
    - The process of interrogating the Java classes for public methods occurs the first time you run your Java code. The namespace is cached, and subsequent calls use the cached version

- Why?
  - Perl is one of the most popular scripting languages. Huge availability of modules on CPAN is big strength.

- When?
  - Perl Inline module has been around for 7+ years, initially developed to leverage C code from within a perl environment. In the last ~5 years more language support has been added. Inline::Java was initially developed in 2001
**Software Development Organization**

**DSS: DSS_SCRIPTING.pm**

- Perl module that hides a bunch of generic stuff that every .pl script would otherwise have to know/learn
  - To use:
    - # import module abstracting the DSS class details
    - use DSS_SCRIPTING;
    - # call the DSS class constructor(s) we need
    - my $dss = new DSS_SCRIPTING();

- Details (purpose of module is to hide these from user)
  - We tell Inline::Java to explicitly 'STUDY' main DSS class (ScriptingEnvironment) so we can use it from perl
  - We turn on AUTOSTUDY which makes Inline::Java automatically study unknown classes as it encounters them
  - We proxy Java's importPackage() by assigning a new namespace TraceLevel to the fully scoped package name so we can then use the shorter namespace $TraceLevel::ALL instead of $DSS_SCRIPTING::com::ti::ccstudio::scripting::environment::TraceLevel::ALL
  - Finally, we enable access to the ScriptingEnvironment Java class constructor

**ATC 2008**

**DSS: Perl – Simple Example**

- Example
  - Simplistic example to show that Perl works with DSS

- What does the Script do?
  - Starts the debug server
  - Connects to the first target
  - Loads the program passed as a parameter
  - Enables analysis events
  - Sets a breakpoint at main
  - Runs the target
  - Checks where it is halted
  - Runs the target
  - Disables the events
  - Exits

- Executing the script
  - perl dss_simple.pl [outfile] [setup config file]
DSS: Example: automated testing

• A common use for DSS is to automate testing
• If you don’t want to write your own framework, use something freely available such as junit
  – JUnit requires raw java code, not javascript
  – You’ll need a .bat file to setup the path and classpath correctly first
    • Simply modify rhino.bat to call your java object

```
8th include JUNIT.bat in the path
and PATH environment variable to the java path
and JUNIT_TEST.bat to call.
and setup .bat to setup the path and classpath correctly first
and rhino.bat to call the Java object
```

Create a junit testing object and compile it
– Create a function with @BeforeClass to initialize a DSS session
– Create a function with @AfterClass to terminate the session
– Create functions with @Before and @After to setup the target and clean up after any test function
– Then finally use @Test to write a bunch of testing functions
  • Validate algorithms are correct with different data sets
  • Validate application performance with the profile functions

• Demo
DSS: References

- Javascript References
  - W3Schools – Javascript Tutorial: [http://www.w3schools.com/js/default.asp](http://www.w3schools.com/js/default.asp)

- XSLT (XML Stylesheet Transforms) References.
  - If the format of the log file isn’t exactly what you need.
  - XSLT Tutorial: [http://www.w3schools.com/xsl/xsl_intro.asp](http://www.w3schools.com/xsl/xsl_intro.asp)

- JUnit References

Installing java support

- CCE does not ship with java support, but it can be easily added
  - Go to [http://www.eclipse.org](http://www.eclipse.org) and click on PROJECTS near the top
  - Click on Show me all the projects
  - Click on Eclipse Project
  - Click Download in the corner
  - From here, there should be a link to archived builds of earlier versions
  - Click on the link for version 3.2
  - Download eclipse-SDK-3.2-win32.zip from this page
  - Extract all files on top of the CCE 3.2 install, but do not overwrite existing files
    - Existing files contain TI specific changes necessary to debug TI devices
  - Now that CCE supports java, you can use it to develop and run DSS based JUnit tests
Creating a DSS JUnit project

- Switch to the Java perspective
- Right click on the project view and select project and give it a name
- Right click on the new project and add a new class
- Right click on the new project and select Build Path->Add External Libraries…
  - Select the DSS jars (all the jars in DebugServer\scripting\lib)
  - Download JUnit from the web (http://www.junit.org/) and add it's jar

Creating a DSS JUnit test

- You'll need to include the necessary imports, and then just define startup, shutdown routines
Creating a DSS JUnit test

- Then just define test routines

```java
public void verifyProjectItem() throws Exception {
    // Run to the end of main
    session = new browser().setServer(“http://example.com”), 80);
    session.login(true);
    // Verify that the elements in the array were sorted
    for (int i = 0; i < arr.length; i++) {
        // Verify that the elements in the array were sorted
        for (int j = 0; j < arr.length; j++) {
            if (arr[i] > arr[j]) {
                throw new Exception("Sort error!");
            }
        }
    }
}
```

Running a DSS JUnit test

- Right click on the test class and choose Debug As->JUnit Test or Run As->JUnit Test
- If you get an error about XPCOM.RUNTIME not being defined
  - Right click, and select Debug As->Debug…
  - Select Arguments and under VM Arguments enter XPCOM.RUNTIME=\<install>\DebugServer\win32
    - Spaces in the path won’t work, so use the ~ filenames: for instance
      XPCOM.RUNTIME=C:\PROGRA~1\TEXASI~1\CCESSE~1\DebugServer\win32
  - Apply and then you can debug this normally
Running a DSS JUnit test

• Results will appear in the JUnit tab

Creating a plugin

• Objectives
  – Create a plug-in that adds a menu item and toolbar button to CCS-Eclipse
    • Clicking the button will execute a GEL expression
  – Show that there are good tools for creating plug-ins
Step 1: Open the PDE perspective
• Window->Open Perspective->Other
  – Select Plug-in Development

• Window arrangement will change and you should see “Plug-in Development” in the perspective indicator

ATC 2008
MSP430 Advanced Technical Conference

Step 2: Start a new project
• From the File menu select New->Project
• Select Plug-in Project
Software Development Organization

**Step 3: Name the project**
- Name your project “Toolbar”
- Will automatically create folders for source and binaries
- Select 3.2 in the Eclipse version
- Click “Next”

Software Development Organization

**Step 4: Plug-in properties**
- Fill in the Plug-in provider name – i.e. “Texas Instruments”
- Click “Next”
Step 5: Templates selection

- Select the “Hello World” template as it adds the menu and button we want
- Click “Next”

Step 6: Message box text

- Change the message to whatever you want
- Click “Finish”
Step 7: Create a run configuration

- Click the Run button
- Double-click on Eclipse Application to create a new configuration
- Click “Run”

Step 8: Test the plug-in

- When CCS-Eclipse starts click the yellow arrow to go to the workbench
- Click on the purple Eclipse button on the toolbar to test your plug-in
- Click “Ok”
- Exit this instance of CCS-Eclipse
Software Development Organization

**Step 9: Dependent plug-ins**

- We are going to be using APIs from other components so we have to specify them.
- Click on the "Toolbar" source file.
- Click on the "Dependencies" tab at the bottom of the window.
- Add:
  - org.eclipse.cdt.debug.ui
  - org.eclipse.cdt.core
  - org.eclipse.cdt.debug.core
  - org.eclipse.debug.ui
- Save the file.

**Step 10: Add required imports**

- In the import statement area at the top of `SampleAction.java` add these import statements as we will be using these classes:

```java
import org.eclipse.cdt.debug.core.CDIException;
import org.eclipse.cdt.debug.core.cdi.model.ICDIFrame;
import org.eclipse.debug.ui.DebugUITools;
```
Step 11: Change to evaluate expression

- Double click on SampleAction.java in the Package Explorer to open the file
- In the "run" function replace:

```
MessageDialog.openInformation(window.getShell(),
  "Toolbar Plug-in", 
  "Hello, Eclipse world");
```

- So that it looks like this:

```
public void evalActivationOption() {
    // Get the current debugging context so that we know which context to apply the UI call to
    // framework frame = (IJavaStackFrame) DebugUITools.getActiveContent().getAdapter(IJavaStackFrame.class);
    try {
        frame = getTarget().getSelectedExpressionToEdit(frame, " Он кодиж ");
    } catch (IDEException e) {
        e.printStackTrace();
    }
    m.printStackFrame();
}
```

- Save

ATC 2008

Step 12: Export

- You want to be able to give your plug-in to other people
- File->Export
- Select “Deployable plug-ins and fragments”
**Step 13: Export location**

- Specify the location to export the plug-in to
  - Specify the `eclipse` folder of your CCS-Eclipse installation
- Click “Finish”

---

**Step 14: Test in CCE**

- Close CCE
- Start CCE
- Start the debugger
- Load a program
- Click on your button