The Art of Debugging:
From STRDBG to RDi

Charles Guarino
Twitter @charlieguarino
Central Park Data Systems, Inc.

About The Speaker
With an IT career spanning over 30 years, Charles Guarino has been a consultant for most of them. Since 1995 he has been founder and President of Central Park Data Systems, Inc., a New York area based IBM midrange consulting and corporate training company. In addition to being a professional speaker across the United States and Europe, he is a frequent contributor of technical and strategic articles and webcasts for the IT community. He is a member of COMMON’s Speaker Excellence Hall of Fame and also Long Island Software and Technology Network’s Twenty Top Techies. In 2015 Charles became the recipient of the Al Barsa Memorial Scholarship Award. Additionally, he serves as a member of COMMON’s Strategic Education Team (SET) and is also a past president and monthly Q&A host of LISUG, a Long Island IBM i User’s Group www.lisug.org.
Charles can be reached at cguarino@centralparkdata.com.
LinkedIn - http://www.linkedin.com/in/guarinocharles
Twitter - @charlieguarino
In This Session …

For years we believed that STRDBG had been adequate for everyday debugging situations. With the introduction of WDSC/RDP/RDi we have been given the ability to extend our productivity in a feature-rich graphical environment.

In this session we will review every aspect of this new environment and explore how the days of green screen debugging have become a technology of the past.

What We’ll Cover …

- Perspectives
- Review program we will debug
- The Debug Server
- Service Entry Points
- Calling a program from within RDi and debug configurations
- Debugging views
- Debugging another user’s program
- Code Coverage
- Wrap-up
Perspectives

- There are many available in RDi
  - This session focuses on the debugging perspective
  - To see all available perspectives click on Window>Open Perspective>Other
    - Or – Typing “Perspective” in Quick Access

What We’ll Cover …

- Perspectives
- Review program we will debug
  - The Debug Server
  - Service Entry Points
  - Calling a program from within RDi and debug configurations
  - Debugging views
  - Debugging another user’s program
  - Code Coverage
  - Wrap-up
Program we will be debugging

- Start program
- Read a record from file CUSTMAST
- If %EOF, leave program loop and exit program
- Call encryption service program, return ciphertext
- Update CUSTMAST with encrypted data
- Read more records from file CUSTMAST

Program we will be debugging (cont.)

```assembly
001000 cltt-opt bmdir('UTILITIES' : 'Q2LE') dtactgrp('no') actgrp('Q1LE')
0000200    option('srcstmt : *nodebugio) debug(input);
0000300    dcl-f custmast disk('ext') keyed usage('update');
0000302    dcl-pr secretdata char(24);  *n char(24) value;
0000305       *n char(1) value;
00000308     end-pr;
0000309
0000311    dcl-s cleardata char(24);
0000312    dcl-s encrypteddata char(24);
0000313    dcl-s direction char(1);
0000317
0010600    read custmast;
0010700    do not %eof (custmast);
0010800    direction = 'E'; // Value of 'E' tells procedure to ENCRYPT
0010900    cleardata = cl3rdata;
0010900    encrypteddata = secretdata(cleardata:direction);
0010200    update custestr;
0010300
0020400    read custmast;
0020500
0020600    *inlr = "on";
```
What We'll Cover …

- Perspectives
- Review program we will debug
- The Debug Server
  - Service Entry Points
  - Calling a program from within RDi and debug configurations
  - Debugging views
  - Debugging another user’s program
  - Code Coverage
  - Wrap-up

The Debug Server

- Listens on the IBM i for debugging instructions from RDi
- It needs to be active before any debugging can occur
  - You will receive a warning message if you try to debug a program and the server is not yet active.
    - Don’t panic! You can start it immediately directly from RDi.
  - Once the debug server is started it will work for everyone
  - There is NOT one server PER USER – only one per system which will service every developer’s RDi debugging requests
    - I recommend putting command STRDBGSVR in your startup program
Starting the Debug Server (3 different ways!)

- OR -

The Debug Server in action

- Runs in subsystem QUSRWRK as jobs and programs QB5ROUTER and QB5SERVER
- Job will use the user ID that started the server
- The debug server will remain active until it is explicitly ended
- There will be an additional job for each program being debugged, serviced by program QRSEEXEC
What We'll Cover …

- Perspectives
- Review program we will debug
- The Debug Server
  - Service Entry Points
    - Calling a program from within RDi and debug configurations
    - Debugging views
    - Debugging another user’s program
    - Code Coverage
    - Wrap-up

Starting the Debugger

- There are three methods to prepare and launch the debugger:
  - **Method 1:** Setting a Service Entry Point
    - When the program is run anywhere using the specified parameters the debugger will be launched
  - **Method 2:** A program can be launched directly from RDi
    - With or without parameter prompting
  - **Method 3:** Debugging an active job
    - Can intercept an active job to identify and resolve issues
Setting a Service Entry Point from a source member

- Right click on any source member

Setting a Service Entry Point from a source member (cont.)

- You will have an opportunity to change any of these values
- This is a HUGE improvement over service jobs and STRSRVJOB
Setting a Service Entry Point from a **source member** (cont.)

- Once the SEP has been set you will receive this confirmation
- You will see your parameters in the SEP view in the RSE

![IBM i Debug Message](image)

<table>
<thead>
<tr>
<th>Library</th>
<th>Program</th>
<th>Program Type</th>
<th>Module</th>
<th>Procedure</th>
<th>User ID</th>
<th>Connection</th>
<th>Enabled</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENCRYPT</td>
<td>ENCODEBG</td>
<td>PGM</td>
<td>*ALL</td>
<td>*ALL</td>
<td>CGUASNO</td>
<td>From STRDBG to RD1</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Service Entry Point being set from a **program object**

![Service Entry Point](image)
Method 2: Calling and debugging a program directly from RDi

Calling and debugging with a prompt
Debug configurations

- Click on "How to Start" for additional parameters

- CAN BE NAMED AND SAVED
- SPECIFY WHAT AND HOW TO RUN CAN SPECIFY PROGRAM PARAMETERS

- Edit Configuration
- Edit configuration and launch.
- Name: My program (Batch)
- What To Debug: How To Start
- Source: Common
- Connection: From STRDBG to ROI:
- Programs or service programs to be debugged:
- Library: ENCRYPT, ENCDDEBUG, *PGM
- Type: Add
- Add
- Edit...
- Remove
- Step into
- Terminate debug session on program completion
- Update production files

- Apply
- Debug
- Close
Launching an existing configuration

- Very useful if you will debugging programs multiple times
- The configuration will remember all of your settings

What We’ll Cover …

- Perspectives
- Review program we will debug
- The Debug Server
- Service Entry Points
- Calling a program from within RDi and debug configurations
  - Debugging views
  - Debugging another user’s program
  - Code Coverage
  - Wrap-up
Submitting and debugging a job directly from RDi

- Debug as submits with your current session’s settings
  - This includes library list, updprod settings, etc.
- Debug (prompt) brings up a debug configuration

Introducing the DEBUG perspective

- “Wakes up” automatically when a program launched in debug mode or an active service entry point is encountered
- Green line is the current line of execution
- Boxes shows shortcuts, breakpoints and current line of execution pointer
What We'll Cover …

- Perspectives
- Review program we will debug
- The Debug Server
- Service Entry Points
- Calling a program from within RDi and debug configurations

Debugging views

- Debugging another user’s program
- Code Coverage
- Wrap-up

Introducing the DEBUG view

- This is the call stack and communication area between RDi and the IBM i
- Can be used to debug multiple jobs at the same time
  - Simply click on the job you want to debug
The DEBUG view

- When debugging multiple jobs at once keep the debug view open
  - Makes it easier to keep track of current job being debugged

Introducing the VARIABLES view

- All program variables are displayed and updated in real time
  - Each variable will change color when its value changes
  - This view is customizable using the drop-down menu
  - Right click to change view and add to monitors view
  - Values can be changed by simply over-typing
Green screen equivalent to variables view!

- Type the debug command EVAL %LOCALVARS to see all variables!

Unreferenced Field Fun Fact!

**RNF7031**

Debug (*input)  Option (*nounref)
Introducing the MONITORS view

- You decide which variables will appear in this view
- Useful when watching a specific set of fields
- You can right click on a field to switch between character or hexadecimal view

Introducing the BREAKPOINTS view

- Breakpoints can be set at the source level or at runtime
- Breakpoints can be conditional or unconditional
- Can also be disabled so you don’t have to delete them
- Watch breakpoints are set at runtime – here we’re watching the variable named “direction”
Adding a breakpoint

Adding a watch breakpoint
Watching for changes in the field “direction”

```
column 91 replace browse
....
    dcl-f custmast disk('ext') keyd usage('update');
    dcl-pr secretdata char(24);  *n char(24) value;
    end-pr;
    dcl-s clearedata char(24);
    dcl-s encrypteddata char(24);
    dcl-s direction char(1);  
    read custmast;
        do not leof (custmast);
    direction = 'E';    // Value of 'E' tells procedure to:
    clearedata = ccl-data;
    encrypteddata = secretdata(clearedata:direction);
    // Update file CUSTMAST with encrypted data
    concdata = encrypteddata;
    update custmast;
```

Watching for changes in the field “direction” (cont.)

```
disk('ext') keyd usage('update');
    dta char(24);  char(24) value;
    dta char(24);  char(1) value;
    custmast();
    'E';    // Value of 'E' tells procedure to:
    clearedata = ccl-data;
    encrypteddata = secretdata(clearedata:direction);
    // Update file CUSTMAST with encrypted data
    concdata = encrypteddata;
    update custmast;
```
Introducing the OUTLINE view

```
Outline
  * Global Definitions
    - Files
      - custmast : DISK (Externally Described)
        - custmr
          - 17
          - 18
          - 32
      - Fields
        - CADDOR1 : Character (30)
        - CADDOR2 : Character (30)
        - CADDOR3 : Character (30)
        - CAVGSAL : Packed Decimal (7,0)
      - ceidata : Character (24)
      - CCMP : Packed Decimal (2,0)
      - CCSNAM : Character (30)
      - CCUSNO : Packed Decimal (7,0)
      - CDTLSLM : Date (10)
      - Cmncdata : Character (24)
      - Ccleardata : Character (24)
      - CPGMRUN : Packed Decimal (5,0)
      - CSTRLEN : Packed Decimal (5,0)
      - CYTOPRF : Packed Decimal (7,0)
      - CYTDSL : Packed Decimal (7,0)
      - CYTDSLA : Packed Decimal (7,0)
      - CVIDSLB : Packed Decimal (7,0)
      - CYTDSLC : Packed Decimal (7,0)
      - dir : Character (1)
      - encrypteddata : Character (24)
  * Indicators
  * *INLR
  * Prototypes
    - encrypteddata : Character (24) EXTPROC ("SECRETDATA")
  * Main Procedure
```

Introducing the PROGRAMS view

- Functionally similar to pressing F14 from DSPMODSRC screen

```
Variables Breakpoints Registers Monitors Programs
*PGM ENCRYPT/ENCDEBUG
```

```
Add Program

Program name

Program type
- Program
- Service Program
- Java Class

OK Cancel
```
Field hovering

- Position the cursor over a field and its value appears.
- Much easier than typing "ev cleardata" or pressing F11!

```c

dcl-s cleardata char(24);
dcl-s encrypteddata char(24);
dcl-s direction char(1);

read custmast;
dow not %eof (custmast);

direction = 'E'; // Value of 'E' tells procedure to ENCRYPT
cleardata = cclrdata;
encrypteddata = secretdata(cleardata:direction);

ENCRIPTEDDATA : y/1bKgS0&*E=

*inlr = "on;  
return;
```

What We’ll Cover …

- Perspectives
- Review program we will debug
- The Debug Server
- Service Entry Points
- Calling a program from within RDi and debug configurations
- Debugging views
- Debugging another user’s program
- Code Coverage
- Wrap-up
Debugging Another User’s Job

- Locate the active job, right click on it and select “Debug As > IBM i job”

What We’ll Cover …

- Perspectives
- Review program we will debug
- The Debug Server
- Service Entry Points
- Calling a program from within RDi and debug configurations
- Debugging views
- Debugging another user’s program

- Code Coverage
- Wrap-up
RDi Line level Code Coverage Analysis Capability

- Code coverage can be launched on any program or service program that can be debugged – independent of language
- You can see exactly which lines were covered and not.
- This can be used to determine the effectiveness of automated or manual tests

* This slide courtesy of IBM

Code Coverage Report

- After running code coverage, a report is shown as an editor.
- You can drill down through programs, modules and procedures and see the coverage statistics for each

* This slide courtesy of IBM
Coverage annotated in the editor

- Drilling down from the report, the editor will be opened on the related member with green and red annotations showing which lines were covered.

New enhancements to code coverage - *beta

RDi – IBM Rational Developer for i Hub

New Beta program for RDi code coverage users!

Overview
Recent Updates
Members
Wiki
Blog
Feeds
Bookmarks

A Beta is now available for an experimental technology optimizing test coverage and utilization that is provided in the form of IBM Installation Manager offering that can be installed into existing instances of IBM Rational Developer for iV9.1.1 or IBM Rational Developer for AIX and Linux V8.1.1. This is an enhanced code-coverage experience which allows your testers and developers to understand your automated or manual test coverage. You will gain insights into what source files are covered and what tests need to be run when changes are made to source or where coverage is missing so that new tests can be added. This can help you optimize your testing reducing the time needed for continuous integration.

Here is the link to the beta site where you will find information and resources that you need to experience the beta (e.g. download info). The beta is included as a part of the RAI Beta program, so make sure to download the Installation Manager packages and documentation for Compiled Languages for use with RDI and RDA. Also see these installation instructions.

If you are new to code coverage, check out this article to get started: Article on Code Coverage. There are also tutorials specific to the new technology included in the Beta download.

Please be sure to use the beta forum to report any issues, and feedback! I look forward to seeing your questions and feedback soon.
What We'll Cover …

- Perspectives
- Review program we will debug
- The Debug Server
- Service Entry Points
- Calling a program from within RDi and debug configurations
- Debugging views
- Debugging another user’s program
- Code Coverage
- Wrap-up

The Art of Debugging: From STRDBG to RDi

Charles Guarino

Twitter @charlieguarino

THANK YOU !!!