

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.

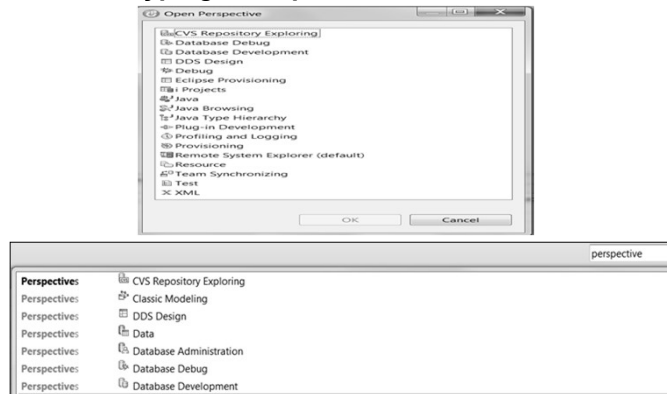
In this session we will review every aspect of this new environment and explore how the days of green screen debugging have become a technology of the past.

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.)

```
*ENCDEBUG.RPGLE **
Line 31      Column 72  Replace 1 change
..... Free-Form+++++
000100      ctl-opt bnddir('UTILITIES' : 'QC2LE') dftactgrp(*no) actgrp('QILE')
000101
000200      option(*srcstmt : *nodebugio) debug(*input);
000300      *
000301      dcl-f custmast disk(*ext) keyed usage(*update);
000302
000303      dcl-pr secretdata char(24);
000304          *n char(24) value;
000305          *n char(1) value;
000308      end-pr;
000309
000311      dcl-s cleardata char(24);
000312      dcl-s encrypteddata char(24);
000313      dcl-s direction char(1);
000317
001400
001600      read custmast;
001700      dow not %eof (custmast);
001800
001900          direction = 'E'; // Value of 'E' tells procedure to ENCRYPT
002000          cleardata = cclrdata;
002100          encrypteddata = secretdata(cleardata:direction);
002200
002300          // Update file CUSTMAST with encrypted data
002400          cencdata = encrypteddata;
002500          update custmstr;
002600
002700      read custmast;
002800      enddo;
002900
003000      *inlr = *on;
```

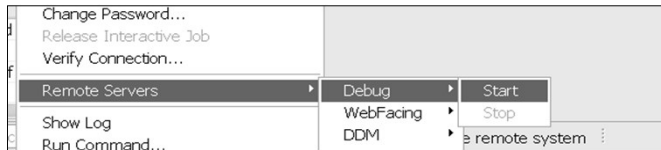
What We'll Cover ...

- 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

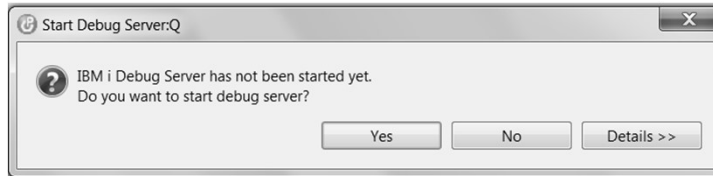
The Debug Server

- 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 -



- 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

Opt	Subsystem/Job	User	Type	CPU %	Function	Status
—	QUSRWRK	QSYS	SBS	.0		DEQW
—	QB5ROUTER	CGUARINO	BCH	.0	PGM-QB5ROUTER	SELW
—	QB5SERVER	CGUARINO	BCI	.0	PGM-QB5SERVER	SELW

Opt	Subsystem/Job	User	Type	CPU %	Function	Status
—	QBATCH	QSYS	SBS	.0		DEQW
—	QDFTJOB	CGUARINO	BCH	.0	PGM-QRSEEXEC	EVTW

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

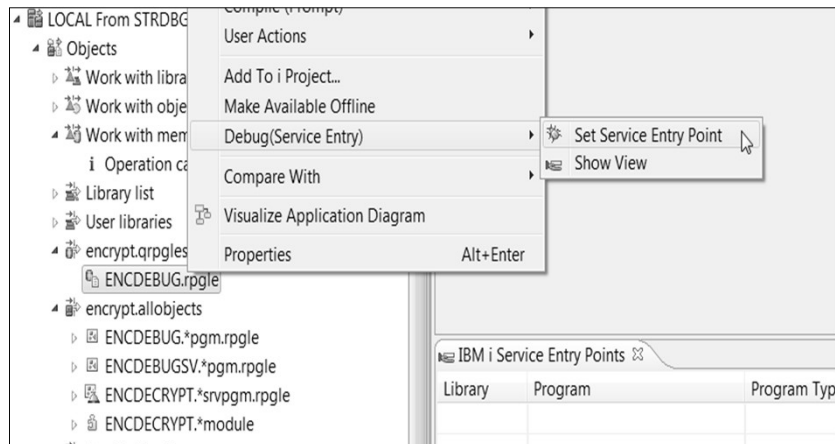
Starting the Debugger

- 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



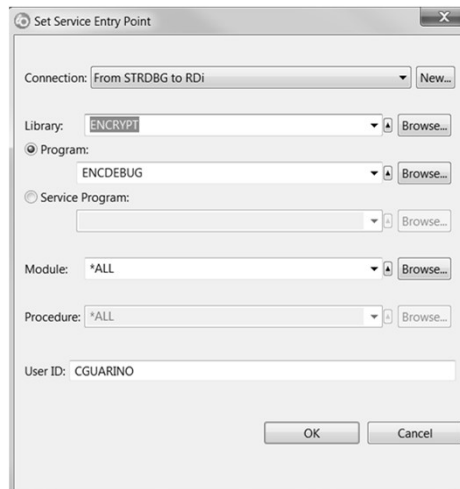
Setting a Service Entry Point from a source member

- Right click on any source member



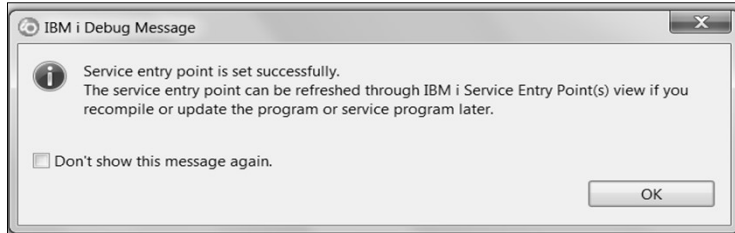
Setting a Service Entry Point from a source member (cont.)

- You will have an opportunity to change any of these values
- This is a HUGE improvement over service jobs and STRSRVJOB



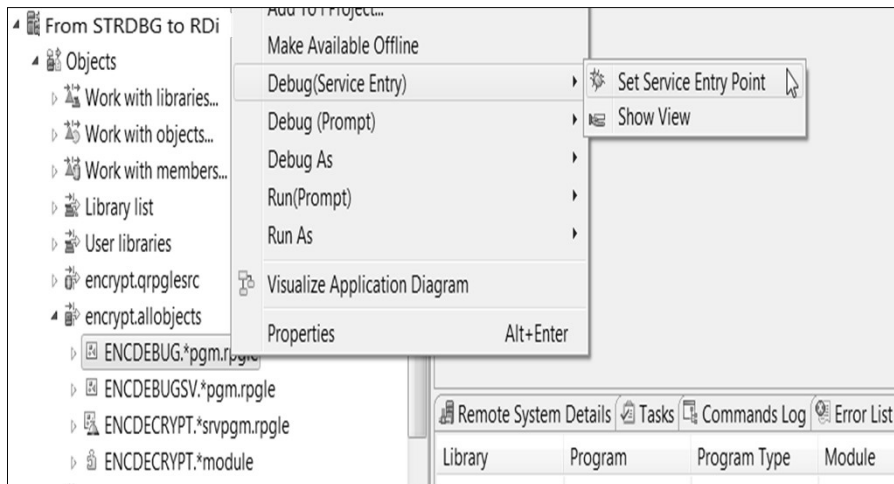
Setting a Service Entry Point from a source member (cont.)

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

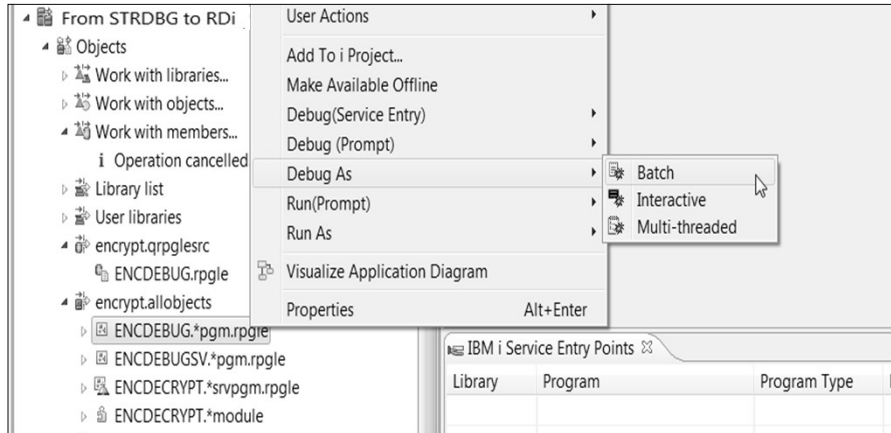


Library	Program	Program Ty...	Module	Procedure	User ID	Connection	Enabled
ENCRYPT	ENCDEBUG	*PGM	*ALL	*ALL	CGUARINO	From STRDBG to RDi	Yes

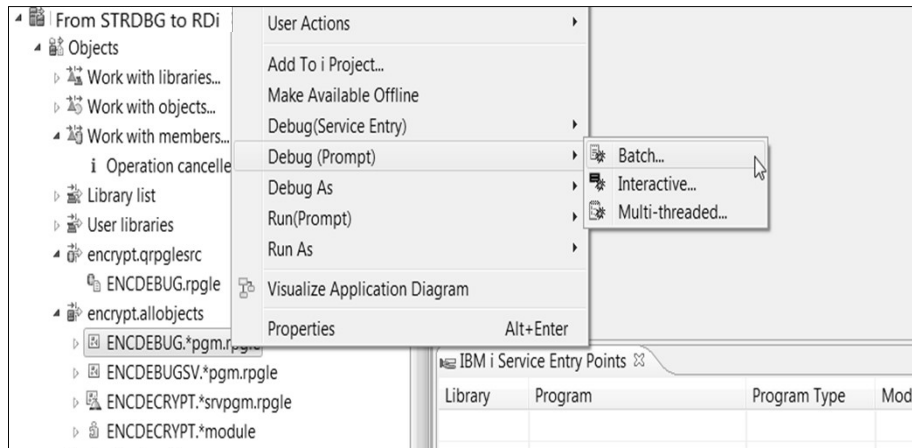
Service Entry Point being set from a program object



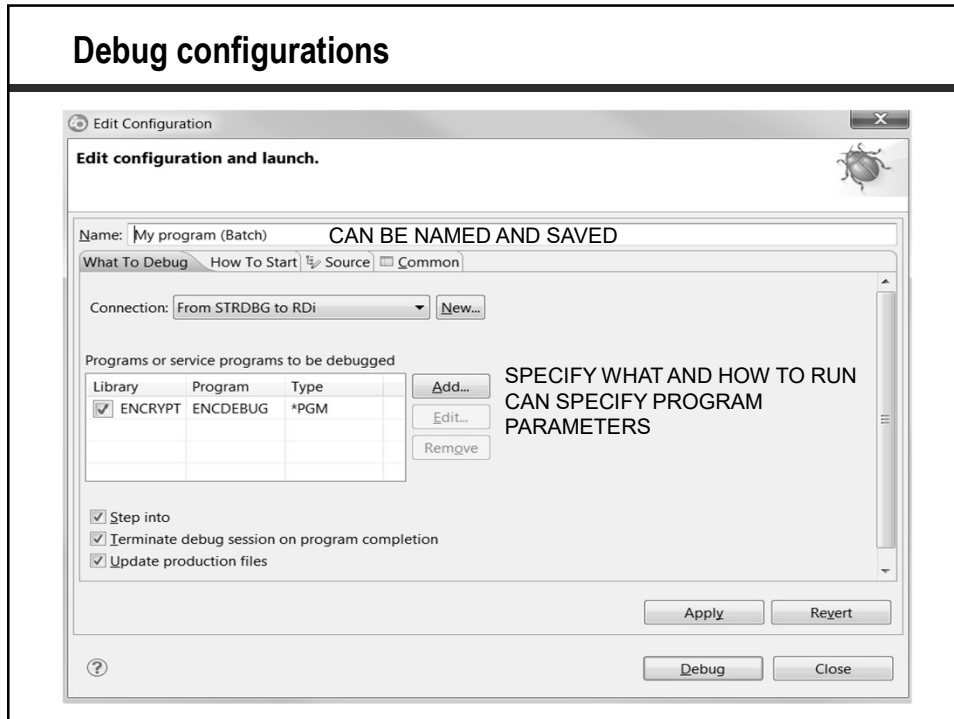
Method 2: Calling and debugging a program directly from RDi



Calling and debugging with a prompt

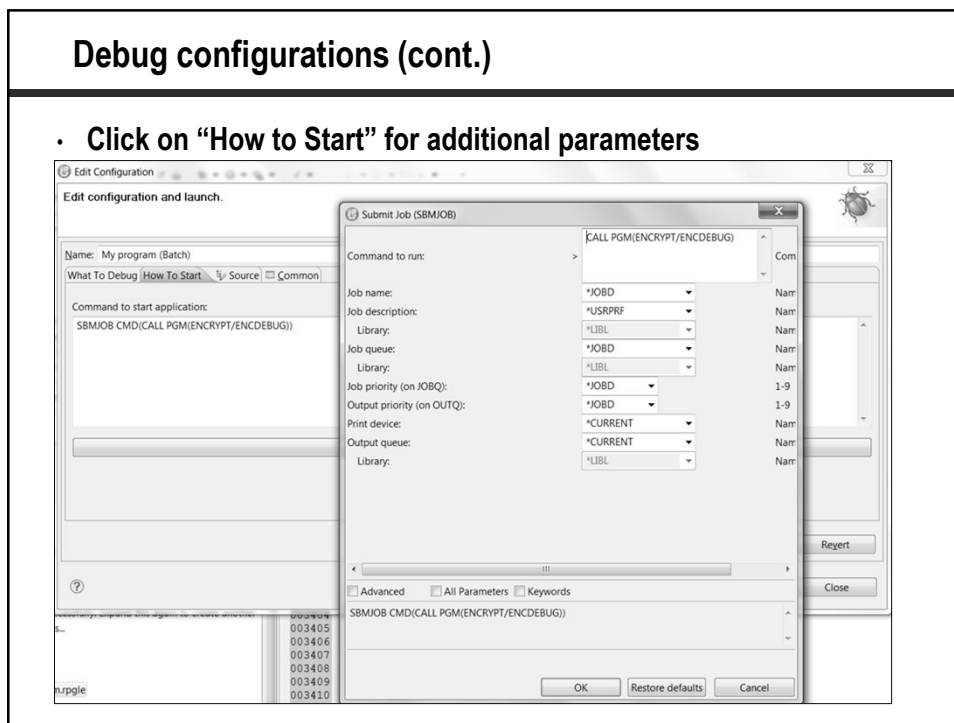


Debug configurations



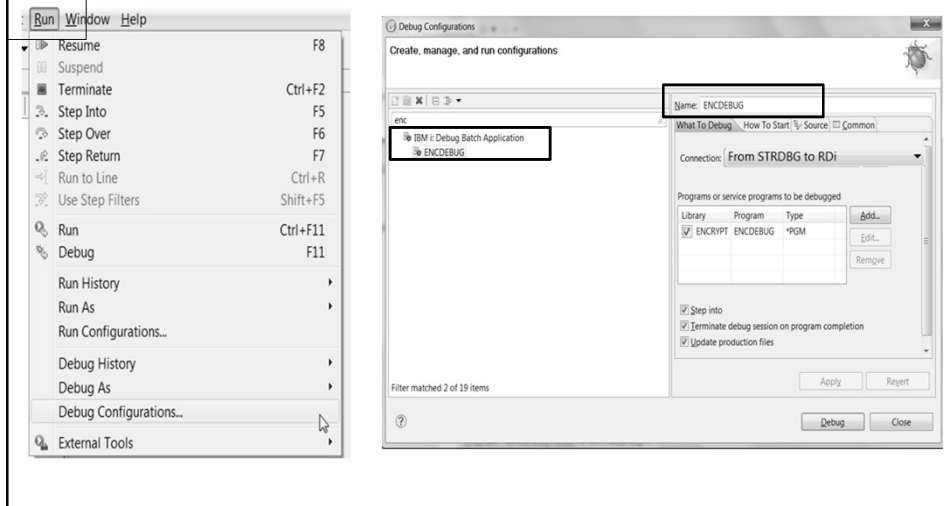
Debug configurations (cont.)

- Click on “How to Start” for additional parameters



Launching an existing configuration

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

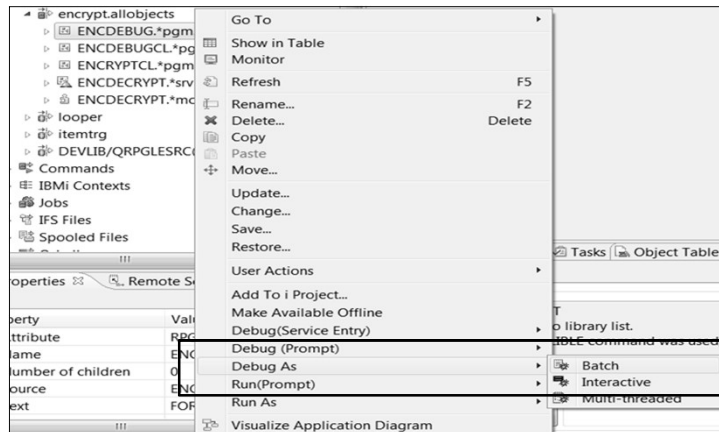


What We'll Cover ...

- 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

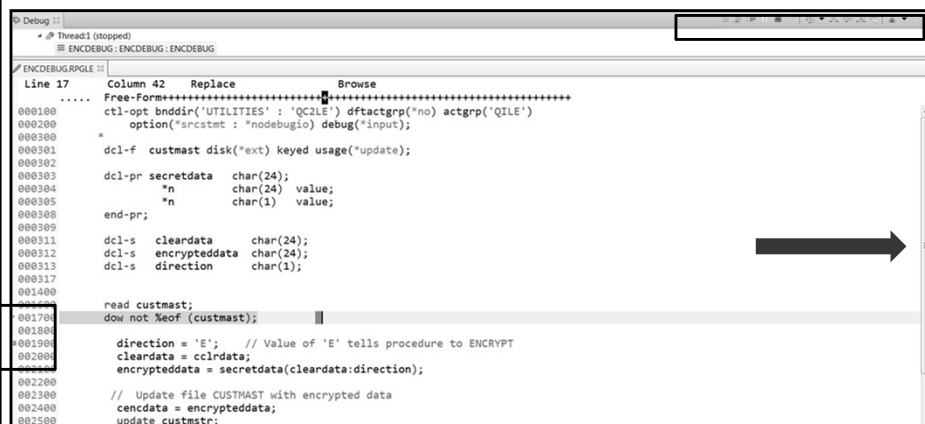
Submitting and debugging a job directly from RDi

- 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

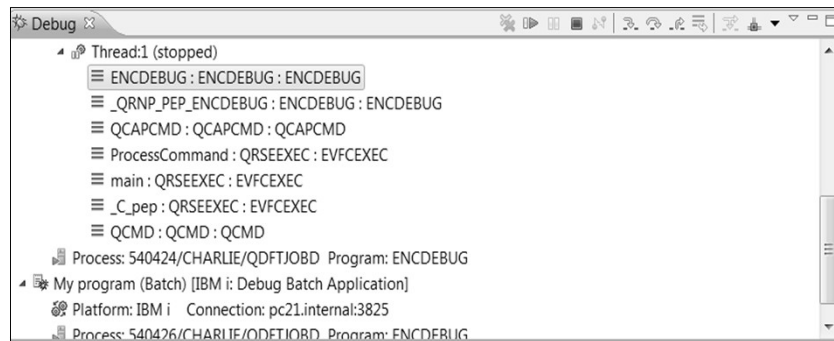


What We'll Cover ...

- 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

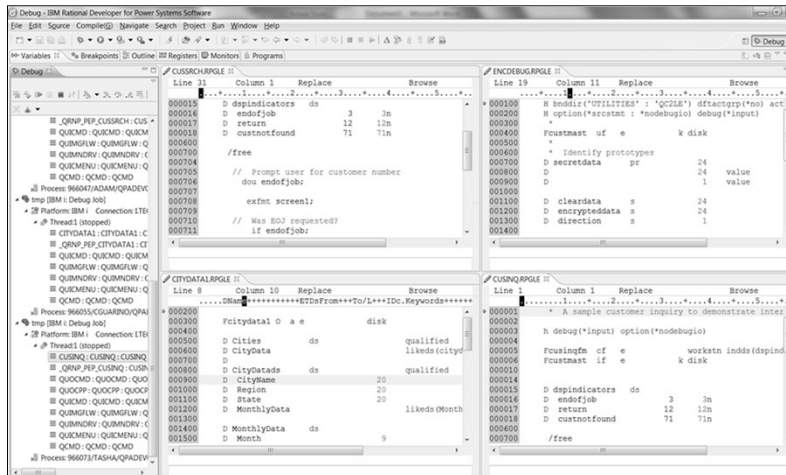
Introducing the DEBUG view

- 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

Name	Value
*IN	
CADDR1	
CADDR2	
CADDR3	
CAVGSAL	
CCLRDATA	0000000.
CCMP	00.
CCSNAM	
CCUSNO	0000000.
CDTLSPM	0001-01-01
CDTLSSL	0001-01-01
CENCDATA	
CLEARDATA	
CPGMRUN	00000.
CSTRLEN	00000.
CYDPRF	0000000.
CYDSL	0000000.
CYDSL	0000000.
CYDSL	0000000.
CYDSL	0000000.
CYDSL	0000000.
CYDSL	0000000.
DIRECTION	
ENCRYPTEDDATA	

Green screen equivalent to variables view!

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

```
Evaluate Expression

Previous debug expressions

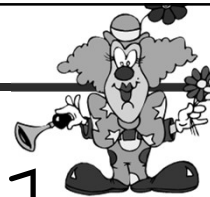
*IN(98) = '0'
*IN(99) = '0'
CADDR1 = '18 Ridgepage Road
CADDR2 = 'Ohio
CADDR3 = '
CAVGSAL = 0045678.
CCLRDATA = '1234567890
CCMP = 01.
CCSNAM = 'John's Hardware Store
CCUSNO = 0000145.
CDTLSPM = '2009-12-22'
CDTLSSL = '2009-12-01'
CENCDATA = 'S1 B 0Wg8 0P 90 5 *
CLEARDATA = '1234567890

More...

Debug . . . eval %localvars

F3=Exit F9=Retrieve F12=Cancel F16=Repeat find F19=Left F20=Right
F21=Command entry F23=Display output
```

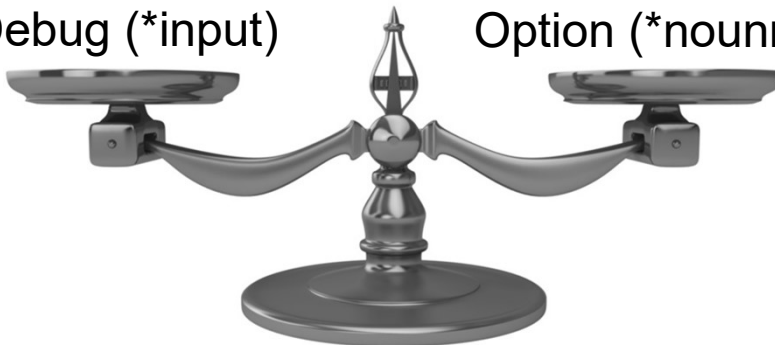
Unreferenced Field Fun Fact!



RNF7031

Debug (*input)

Option (*nounref)



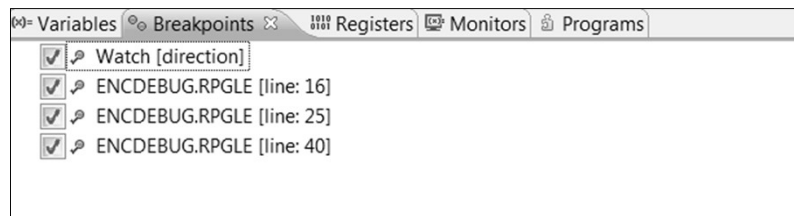
Introducing the MONITORS view

- 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

The screenshot illustrates the steps to add a breakpoint to a specific line of code. The code being edited is as follows:

```

002004      direction = 'E'; // Val
002100      cleardata = cclrdata;
002200      encrypteddata = secretdata;
002400
002500
003101      // Update file CUSTMAST wi
003102      cencdata = encrypteddata;
003103
003404      dow not %eof (custmast);
003405      cencdata = encrypteddata;
    
```

The 'Edit a Line Breakpoint' dialog box shows the following configuration:

- Optional parameters: Make the breakpoint conditional upon the following parameters
- Frequency: From: 1, To: Infinity, Every: 1
- Expression: ccusno = 0037828

Adding a watch breakpoint

The screenshot illustrates the steps to add a watch breakpoint to a specific line of code. The code being edited is as follows:

```

001801
002001      /free
002002      read custmast;
002003      dow not %eof (
002004
002100      direction =
002200      cleardata =
002400      encrypteddat
002500
003101      // Update fi
003102      cencdata = e
003103      update custm
003105
003106      read custmast;
003107      enddo;
003108
003300      *inlr = *on;
003400      return;
003401
003402      begsr *inzsr;
003403      read custmast
003404      dow not %eof
003405      cencdata =
003406      update cust
003407      read custmast
003408      enddo;
003409      setll *zero c
003410      endsr;
003411
    
```

The context menu shows the following options:

- New
- Prompt (F4)
- Syntax Check Line
- Cut (Ctrl+X)
- Copy (Ctrl+Insert)
- Paste (Ctrl+V)
- Select
- Selected
- Deselect (Alt+U)
- Filter view
- Show all (Ctrl+W)
- Source
- View
- Edit Breakpoint...
- Remove Breakpoint
- Disable Breakpoint
- Add Watch Breakpoint...
- Run To Location
- Monitor Expression
- Monitor Memory
- Switch View

Watching for changes in the field "direction"

GRPGLE

1	Column 91	Replace	Browse
....	Free-Form	+++++	+++++
	ctl-opt	bnddir('UTILITIES' : 'QC2LE')	dftactgrp(*no) actgrp('QILE')
	option	(*srcstmt : *nodebugio) debug(*input);	
*			
	dcl-f	custmast disk(*ext) keyed usage(*update);	
	dcl-pr	secretdata char(24);	
		*n char(24) value;	
		*n char(1) value;	
	end-pr;		
	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)	
		// Update file CUSTMAST with encrypted data	
		cencdata = encrypteddata;	
		update custmstr;	

read custmast;
dow not %eof (custmast);

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

// Update file CUSTMAST with encrypted data
cencdata = encrypteddata;
update custmstr;

Add a Watch Breakpoint

Required information

Sets a breakpoint to stop execution when data changes at a specific address

Address or expression: direction

Number of bytes to watch: 0

User label (optional):

< Back Next > Finish Cancel

Watching for changes in the field "direction" (cont.)

disk(*ext) keyed usage

1	Column 91	Replace	Browse
	ata	char(24);	
		char(24) value;	
		char(1) value;	
	ata	char(24);	
	eedata	char(24);	
	son	char(1);	
	ustmast);		
	'E'; // Value of 'E'		
	cclrdata;		
	a = secretdata(cleardata		
	le CUSTMAST with encrypt		
	ncrypteddata;		
	str;		

ustmast);

'E'; // Value of 'E'

cclrdata;

a = secretdata(cleardata

le CUSTMAST with encrypt

ncrypteddata;

str;

Add a Watch Breakpoint

Optional parameters

Make the breakpoint conditional upon the following parameters

Thread: Every

Frequency

From: 1

To: Infinity

Every: 1

< Back Next > Finish Cancel

dow not %eof (custmast);

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

// Update file CUSTMAST with encrypted data
cencdata = encrypteddata;
update custmstr;

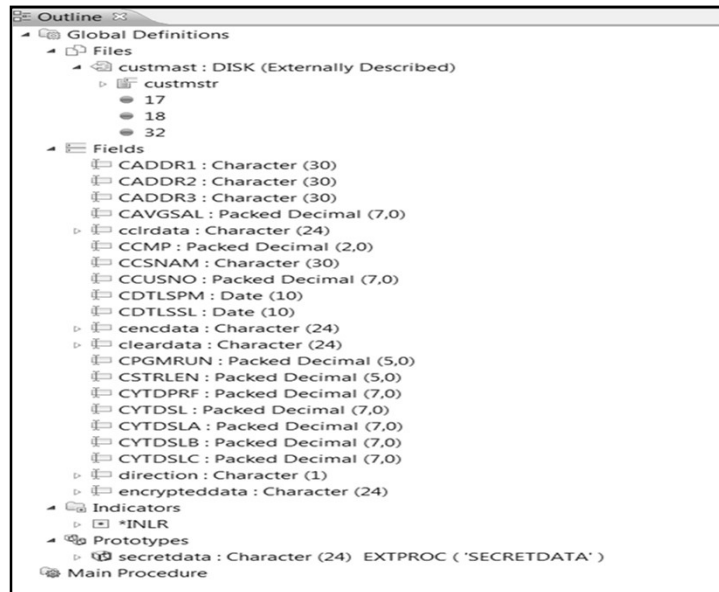
read custmast;

Debug Engine Message

DBG0009I Variable direction has changed in program ENCDEBUG, module ENCDEBUG, which is running in job QDFTJOB /CGUARINO /051498.

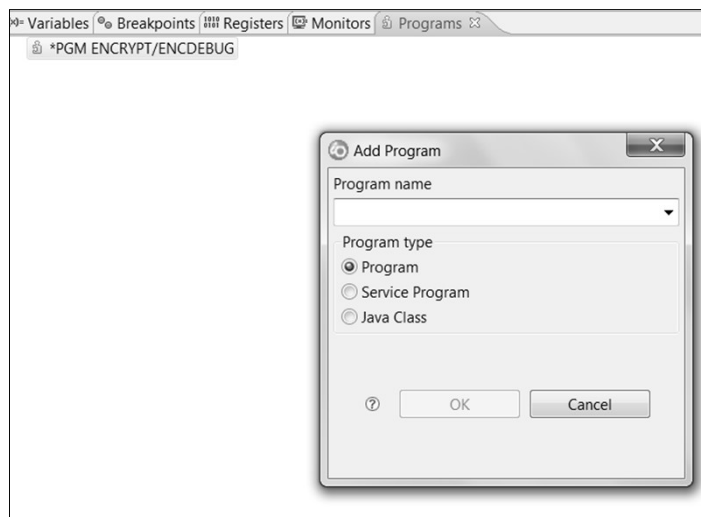
OK

Introducing the OUTLINE view



Introducing the PROGRAMS view

- Functionally similar to pressing F14 from DSPMODSRC screen



Field hovering

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



```
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);

re
en

*inlr = *on;
return;
```

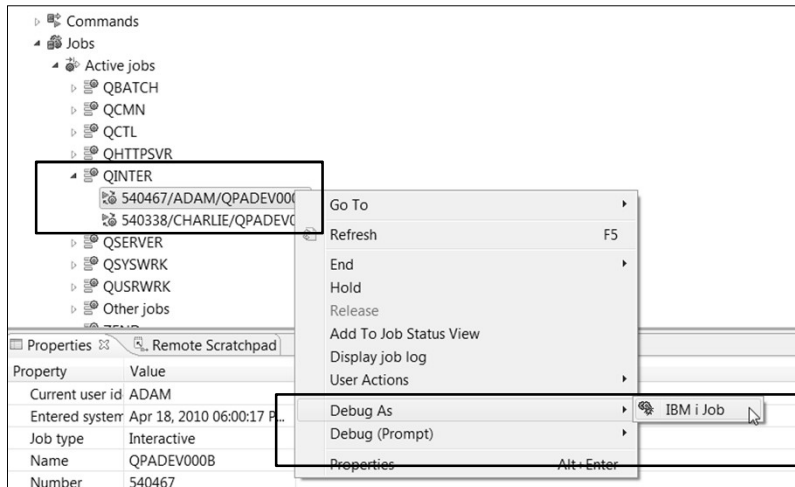


What We'll Cover ...

- 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”

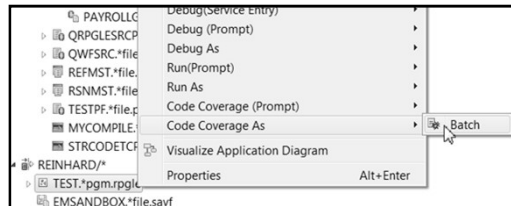


What We'll Cover ...

- 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

Code Coverage Report (RBC4NNF1C_2014_04_23_123357_0632) ::

Code Coverage Report (Line)

Code Coverage Summary

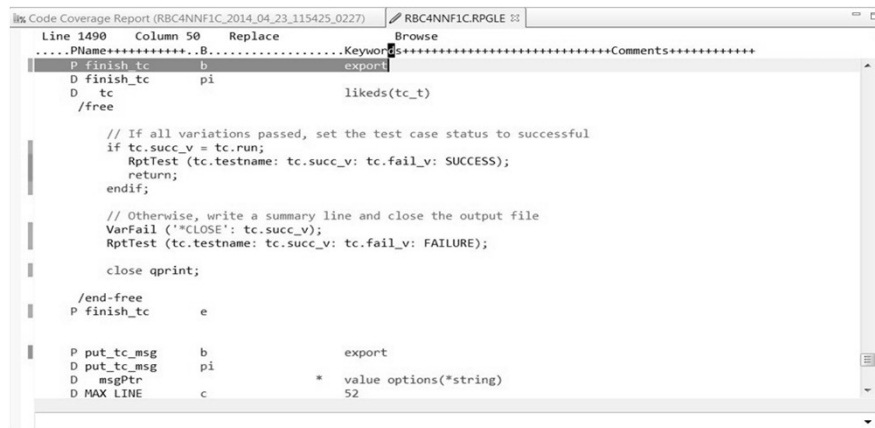
Code coverage report (analyzed at Apr 23, 2014 12:34:22 PM, generated at Apr 29, 2014 10:55:30 AM)

Element	Coverage	Covered	Total
*PGM MKWAN/RBC4NNF1C	88%	617	699
RBC4NNF1C	88%	617	699
RBC4NNF1C.RPGL	88%	617	699
GPHPRCCMP10	0%	0	6
UCSPRCCMP8	0%	0	6
INDPRC10	0%	0	6
CHRPRCCMP120	0%	0	6
PUT_TC_MSG0	0%	0	13
finish_t0	78%	7	9
chk_result0	89%	8	9
show_error0	90%	18	20
RBC4NNF1C0	91%	406	444
chk_subres0	92%	24	26
DATPRCISO0	100%	3	3
DATPRCUSA0	100%	3	3
DATPRCEUR0	100%	3	3
DATPRCYMD0	100%	3	3

* 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.



```
Line 1490 Column 50 Replace Browse
.PName+++++.B.....Keyword+++++.Comments+++++
P finish_tc b export
D finish_tc pi likeds(tc_t)
D tc
/free

// If all variations passed, set the test case status to successful
if tc.succ_v = tc.run;
RptTest (tc.testname: tc.succ_v: tc.fail_v: SUCCESS);
return;
endif;

// Otherwise, write a summary line and close the output file
VarFail ("*CLOSE": tc.succ_v);
RptTest (tc.testname: tc.succ_v: tc.fail_v: FAILURE);

close qprint;

/end-free
P finish_tc e

P put_tc_msg b export
D put_tc_msg pi
D msgPtr * value options(*string)
D MAX LINE c 52
```

* This slide courtesy of IBM

New enhancements to code coverage - *beta

RD_i – IBM Rational Developer for i Hub

RD_i – IBM Rational Developer for i Hub



Overview

Recent Updates

Members

Wiki

Blog

Feeds

Bookmarks

New Beta program for RD_i code coverage users!

Musri | Feb 20 | Visits (417) 1

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 i V9.1.1 or IBM Rational Developer for AIX and Linux V9.1.1.

This is an enhanced code coverage experience which allows your testers and developers to understand your automated or manual test coverage. You will gain insight 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 RAD Beta program, so make sure to download the Installation Manager package and documentation for Compiled Languages for use with RD_i and RDAL. 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 downloads!

Please be sure to use the beta forum to report any issues, and feedback! I look forward to seeing your questions and feedback soon.

8+1 0

in Share

Tweet 1

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 !!!