Into each programmer's life, a some bugs must fall. In this session, Susan RDP - Rational Developer for Power Systems and RDi - Rational Developer for i. The same debugger (minus a few features) is in WDSC, for those of you still using the older toolset.

If you've tried using this debugger that's integrated with the RSE tool for editing code and struggled with making it work, you may find that 3 little words can make all the difference: Service Entry Points. Come here to see how to use SEPs when debugging your programs. As an added bonus, we'll even talk about how SEPs can be used with the green screen debugger, just in case you're ever stuck without your RDi-installed workstation handy. We'll also cover using debug configurations for those occasions when SEPs may not be the best choice.

The author, Susan Gantner, is co-founder of Partner400, a firm specializing in customized education and mentoring services for IBM i developers. After a 15 year career with IBM, including several years at the Rochester and Toronto laboratories, Susan is now devoted to educating developers on techniques and technologies to extend and modernize their applications and development environments. This is done via on-site custom classes as well as conferences and user group events.

Together with her partner, Jon Paris, Susan authors regular technical articles for the IBM publication, IBM Systems Magazine, IBM i edition (formerly iSeries Magazine), and the companion electronic newsletter, IBM i EXTRA (formerly iSeries Extra).

You may view articles in current and past issues and/or subscribe to the free newsletter or the magazine on the web. You can also view Jon and Susan's blog: ibmsystemsmag.com/Blogs/iDevelop/

Susan and Jon are also partners in SystemiDeveloper, a company that hosts the RPG & DB2 Summit conferences. See www.SystemiDeveloper.com for more details.

Feel free to contact the author at: susan.gantner @ partner400.com or visit the Partner400 web site at www.partner400.com
Debug Views

Programmer chooses debugging view(s)
- Statement number only
- Source from original source file
- Compiler-generated listing
- No debugging capability

NOTE: Statement number is the default!
- Consider changing defaults for your system if you want to always produce programs that are able to use source/listing view debug

Old RPG programs:
- CRTRPGPGM ... Option(*SRCDBG or *LSTDBG)

With the *NONE option for debug view, the DUMP operation code is not supported.

Note that debug data can be removed with CHGMOD or CHGPGM. Debug data can be removed without removing the ability to re-translate the program.

If source or copy view is chosen, the source member must be available at the time of the debug session. If that source has changed since the module being debugged was compiled, the debugger will produce a warning message.

The *LIST option saves a copy of the source listing report with the program object. It results in a larger module/program size, but it results in a module or program that can always be debugged --- even if it's on a different system from the source code or if the source has changed since compilation.

If you are unfortunate enough to still be working with RPG/400 language programs, they must also have source view debug specified at compile time using Option(*SRCDG) or Option(*LSTDBG). There is no ability to produce both source view and listing view.
Two modes to get the debugger started

The Easy Way (Service Entry Points)
The Not-so-easy way (Create/use a configuration)

Debugger features in common

After it’s started:
Add Breakpoints
Monitor variable values
Step thru logic
and more ...

Some people would not agree that the second way to get the debugger started is harder, but I find it harder. The "not so easy" way does have some advantages if you tend to need to debug the same program often because you can save a sort of profile for debugging that program and quickly recall it. However, I am typically convinced that I'll never need to debug this program again after I find and fix this one bug, so I rarely think it is worthwhile to go to the extra effort to set up the profile up front.

We'll cover both options here.
Starting a Debug Session from RSE

My favorite option

- Service Entry Point
  - Program selected must be written in an ILE language
    - Member types ending in ...LE & compiled since V5R2

For other types of programs

- Create a Configuration
  - Use Debug ...
    - Batch
    - Interactive
    - Job
    - Multi-Threaded
  - With / without Prompt

To clarify the restrictions for using SEPs - when I say it only works with ILE programs compiled since V5R2, that doesn't mean you can't debug those non-ILE programs in a session begun with an SEP. It only means that I can't use a non-ILE program to set the Service Entry Point.

Once you have a debug session going, you can debug any kinds of programs from that session - either by adding them to the debug session manually or by stepping into the code on a call statement from the original SEP program.

The new Service Entry Point debugging is so much simpler than the old way of using the RSE debugger, some programmers even write a small ILE language program (CLLE or RPGLE) to call any RPG/400 programs they need to debug just to kick off the debug process the easy way.
What Are Service Entry Points (SEPs)?

A method to start a debug session
- Particularly useful for where you don't know the job to debug up front
  - e.g., batch jobs, web server jobs, DB server jobs, etc.
  - But also often easier even for interactive job debugging

You specify the program you want to debug with a particular user profile running that program
- Typically the user profile is your own, but could be another user

When the specified user begins running the program, the system notifies you so that you can debug that job
- Cumbersome process in green screen - much simpler with RSE
  - The green screen process is covered in the appendix of this handout
- For green screen debug users, can be less tedious than other methods

Service Entry Points provide a great way to debug other jobs besides your own. I much prefer using the RSE debugger with Service Entry Points and we'll cover how to do that shortly. But for those who cannot use the debugger that comes with RSE, this method can be used with the STRDBG green screen debugger.
Service Entry Points in RDi

Specifies a Program, Service Program, Module or Procedure and a User Profile

- User Profile by default is the current user of the connection
- If/when that user invokes the specified program/procedure requested
  - The Debug Perspective starts (if not already active)
  - The program/procedure/module is put into debug mode
- The Service Entry Point View shows details

The SEP contains information about what program is to be debugged and for which user profile. For most occasions the user profile will be the programmer’s own profile name, which is the default. But if you have that “problem user” who is the only one who can make the program fail, it’s handy that you can put other user profile names in as well.

Whenever the specified user calls the specified program (or procedure), the Debug Perspective in RSE will automatically begin (if it’s not already active) and the program will be put into debug mode and will wait at the first executable statement for the programmer to add breakpoints or Step or whatever is needed to begin the debug session.
**SEP: Sequence of Events**

Start Service Entry Point Debug from RSE
- If needed, modify debug preferences first
  - Particularly look at "Update Production Files"
  - Details on this coming later
- Select one or more programs, service programs, etc.
  - Set Service Entry Point

Call the Program to be debugged
- From 5250 emulation screen or submit batch job or server job ...
- Debug perspective will be started (if necessary) and debug session started on the called program or procedure
  - At beginning of execution, waits for potential debug setup
    - e.g., add breakpoints, monitor variables, etc.
  - Run program(s) in debug mode

When you are finished debugging the program
- Right click > Remove in "IBM i Service Entry Points" view

**Note: Refresh after re-creating a Service Entry Point program**
- Right click > Refresh in "IBM i Service Entry Points" view

You may debug any type of job as easily as an interactive job using SEPs. Server jobs, batch jobs or interactive jobs all react the same way to the SEP when the program starts to run.

Don't forget to remove your Service Entry Point for a program after you are finished with debugging it so that subsequent calls to the program won't cause the program to stop.
Debug Configuration Sequence of Events

Select Program to debug
- In RSE Object list or with source member open in editor

Press the Debug icon menu arrow
- Or use Run menu
  - Choose Debug Configurations ...
  - Choose type of Debug session to run
    ▶ IBM i Interactive - Program runs in attached RSE Server job (not recommended)
    ▶ IBM i Batch - Will submit a new batch job and issue your call command
    ▶ IBM i Job - Allows you to "connect" the debugger to another existing job
  - Right click, "New" option to create a new debug configuration of that type
  - Name the configuration and fill in the information
    ▶ For Interactive and Batch, How to Start (call command)
    ▶ For Job, Select an existing job on the system
    ▶ For all, specify qualified program name(s)
    ▶ For all, "Step Into" and "Update Production files" options may be selected
  - Press "Apply" button, then "Debug" button

- For Interactive or Batch configurations
  - Program starts (and job if Batch)
  - Debug Perspective starts

- For Job configurations
  - Debug perspective starts then prompts you to start the program in the job

The old "non-SEP" way to use the debugger can be very cumbersome.

However, in some cases, it could be less cumbersome if you need to debug the same program many times. The reason it's cumbersome is that you need to fill in a lot of detail about the program and how to call it (e.g., parameter values to use) and about the job in which the program will run. However, once you have filled in that information, you save it as a "debug configuration". This makes it very easy to repeat the debug session for this same program/parameters/job - even easier than SEPs, especially after making changes to the program, where you would need to refresh the SEP before calling the program again.

Of the different options for IBM i debugging, the one I use most often is "Job". I use this option even for interactive jobs, because otherwise I need to set up a special emulation session and establish it as my "RSE communications server." Details follow.

Note that if you choose "IBM i Interactive" as the debug type, you must have an interactive session (i.e., an emulation "green screen" session) connected to your RSE workbench. To do this, start an interactive session (using any emulator you like - Access, Mocha, etc.) and sign on to the session. Then enter the command STRRSESVR to connect the session to RSE. When you see a screen that says "IBM RSE Communications Server", go back to the debug perspective and continue with the debug sequence.
This dialog will only be used with the old "non-SEP" way of debugging.

Use this dialog if you want to debug an interactive or batch job. Note that if it's a batch job, you'll need to get all the job ID information before creating this configuration. For that reason, there's a separate option for debugging batch jobs. This is most commonly used for interactive jobs.

Note the "Update Production Files" check box. Later we will see a preference you can set to have this turned on by default.

The wide Browse button is to help you find the job you want to debug. It must already be in the system. The default is to search for all jobs on the system with your user profile.

Don't forget to fill in at least the first program to be debugged in this dialog. Other programs can be added during the debug session - either manually or by stepping into them on a call. You may want to name the configuration as well.

Press Apply to save your configuration, then if you want to start debugging right away, press Debug.

After pressing Debug, the debug perspective becomes active and a dialog box shows up that instructs you to do 2 things:
1) Use Step or Resume and
2) Start your application on the server

So press one of the step buttons or the Resume button, then call the program - from a green screen or whatever mechanism you have to start the program to be debugged in the job specified on this dialog box.
**Shortcut for Debug Job**

If you want to debug one of your own jobs

- You may use the Jobs subsystem in Remote Systems
  - Choose your job (e.g., via "My Active Jobs" or "My Jobs")
  - Right click > **Debug (Prompt)** > IBM i Job
    - Dialog for IBM i Debug job appears
    - Name the configuration & add program(s) for debug as before
    - Apply and/or Debug as before and follow instructions to press Step or Resume and then start the application

This is often a faster way to get into creating a configuration for IBM i Debug Job.

Simply use the Jobs subsystem in the Remote Systems view (below Objects, Commands and IBM i Contexts. Choose the job you want to debug from there and do Debug (Prompt) ... IBM i Job.

At that point, the same dialog we saw before appears where you can name the configuration, add program(s) to be debugged and then Apply and Debug.
This dialog will only be used with the old "non-SEP" way of debugging.

This way, you can tell the debugger to submit the job to call the program to be debugged.

Note the "Update Production Files" check box. Later we will see a preference you can set to have this turned on by default.
What if . . .?

If this message occurs . . .

Click Yes
- and then hope you have appropriate authority!
- If not, ask someone with more authority to run STRDBGSVR CL command on the host
- In earlier releases of RDP/RDi, you may not have the option to start if from the message dialog, so use the CL command

Note: This should only need to be done once per IPL
- Best if you make STRDBGSVR part of the system startup routine

Note that the message that pops up if the debug server hasn't been started tells you exactly how to start the debug server. This typically only needs to be done once per IPL of the system. It's a good idea to add the STRDBGSVR command in your system startup program so the developers should never need to do this.

Note: Sometimes developers don't have enough authority on the system to start the debug server. So you may need to get someone with more authority to start it if you see this message.

You may also start the debug server from within RDi - right click on Objects (under your connection name) and choose Remote Servers > Debug > Start.
Preferences for Debug

- Particularly important - Update Production files, when needed
  - Not selected by default

Use Window (pull down menu) Preferences to find the IBM i Debug preference page. If you use production style libraries for your debugging, you will need to set the Update Production Files on. You may recall that this can be set for each debugging configuration, but if you set it here, you don’t need to remember to set it on each time and it will work for SEP debugging as well.
There are some additional preferences that can be relevant to IBM i debugging on the Compiled Debug preferences page.

Some developers like to use the "Center view on debug execution line" and the option to double click on a field to add it to the Monitor list. We haven't shown the Monitor list yet, but it is coming up in a later chart.
The Debug Perspective

Set breakpoints in editor view

Variable Values
Values recently changed highlighted in red
Tip: see warning in notes for large programs

The line shows the current statement where the program is stopped.

The use of the Variables tab is illustrated here to show the values of ALL variables currently active in the program.

Note that for very large programs with many variables, this could slow down the debug process considerably. In those cases, many programmers try to ensure the Variables tab is not selected during debug. Instead you can use the Monitors tab to monitor only a small number of specific variables. More on that coming up.
Monitoring Variable Values

Add variables desired to the Monitors view
- Position on a variable name, Right click > Monitor Expression
- Monitored variables' values may be changed
  -- Double click on name in Monitors view to change the value

Tip: Hover over variable name in Editor view to see its current value!
Tip: Double click on a value in Monitors to change its value

Note that the value appearance by hovering over the name is enabled in the Preferences under "Compiled Debug." It is normally turned on by default, but if yours isn't working check the value in your preferences. An earlier chart detailed how to see these preferences.

Note that there is also a preference to be able to add a variable to the Monitors View by double clicking on it. This is not enabled by default. If you want to enable this feature (so that you do not need to right click and use Monitor Expression each time) use the preferences dialog shown earlier.
Running the Program in Debug

Resume (F8)
- Run until an event is encountered (breakpoint, watch, etc.)

Suspend
- Stop program at point of execution

Terminate (Ctrl + F2)
- End the debug session

Step Into (F5)
- Debug the called program/procedure

Step Over (F6)
- Run the called program/procedure, stop at the statement after the call

Step Return - for ILE langs compiled on V5R2 or higher (F7)
- Run until back in my caller, then stop at the statement after the call

Run To Location (from source context menu only) (Ctrl + R)
- Run and stop at current cursor position
  - unless another breakpoint/watch/event is encountered before

---

The options shown here are also available from the Run pull down menu in the Debug Perspective.

One advantage of looking at the Run pull down menu is that it shows you the keyboard shortcuts for some of the options, in case you forget. F5 is Step Into (equal to F22 in the green screen debugger) and F6 is Step Over or a "normal" step, equivalent to F10 in green screen.
Running the Program in Debug

From the "Run" menu
- You can resume, step, etc.

From the Source context menu
- Run to Location
- Set/remove Breakpoints (or dbl click)
- Monitor Expressions (variables)
- Switch View (source/list)

Note that there are several actions that can be done from either the Run pull down menu or the pop up context menu.

Note the Run to Location option from the context menu (i.e., the right click menu) in the source view. This sets a temporary breakpoint at the statement where it is used and runs to that breakpoint - then removes the breakpoint so the program will not continually stop at that point.
Adding a Watch Breakpoint

A watch breakpoint triggers whenever a variable changes.

- Right click on variable name
- Then select:
  - Number of bytes
    - 0 means “all”
  - Frequency of monitoring

A watch breakpoint isn’t associated with a specific line number, but it will stop when a variable value changes.
Watch Breakpoint Triggered

When breakpoint is triggered
- Message pops up

- If the variable is Monitored, Monitor window shows updated value
  - Or you can simply hover over the variable name in the source to see value

Note that if you just want to "peek" at the variable's current value, rather than watch it as it changes from breakpoint to breakpoint or step to step, you can simply hover your mouse over the variable name for a second or so.

To change the value, put it in the Monitors view or you can edit a value from the "Variables" view that has all the variable values.
Adding a Conditional Breakpoint

Break when a specific condition is true

1. Set a "regular" breakpoint
2. Right click on the breakpoint icon (left of seq #)
3. Select "Edit Breakpoint..."
4. Click Next button on 1st "Edit ..." dialog box
5. Enter expression in next dialog box
6. Press Finish

You don't get a chance to put conditions into the breakpoint at the time you make the breakpoint initially. But if you go back and edit an existing breakpoint, you can then add conditions.

The syntax for the condition is similar to the green screen debugger. The following logical operators can be used:

- <  Less than (a < b)
- <= Less than or equal (a <= b)
- >  Greater than (a > b)
- >= Greater than or equal (a >= b)
- =  Equal to (a = b)
- <> Not equal to (a <> b)

AND (a > b AND b <> c)
NOT (NOT a > b)
OR  (a > b OR b = c)

Conditional breakpoint expressions can include some built-in functions such as:

- %SUBSTR Substring a variable - %substr(email 1 10)
- %ADDR Return the address of a variable - %addr(email)
- %INDEX Set the index of a table, array or multiple-occurrence data structure - tab1 = %index(4)
- %VARS Escape a reserved debugger name as a variable - %vars(eval)
Note that there is extensive help via F1 for the debugging process.
Resources to learn more...

An Easier Way to Debug Batch Jobs by Scott Klement
- iprodeveloper.com/article/rpg-programming/an-easier-way-to-debug-batch-jobs-18029

Debugging With WDSC Service Entry Points by Aaron Bartell

More WDSC Debugging Tips by Aaron Bartell

Three Little Words That Simplify Debugging
- www.it jungle.com/fhg/fhg030707-story02.html

Debugging Tools

Any Questions?
**Batch Jobs (Green Screen) - SEPs**

Start debug on the program (STRDBG) on the program

- Find a statement to put a breakpoint on (probably near the beginning
- Set an SEP breakpoint on that line
  - Use the `SBreak` (SEP breakpoint, not Break) on debug command line
  - If you want a different user aside from yourself, specify the user
  - For example: `sbreak: 145 user: QUSER`
- When the break point is set, press F12 to let the debugger run

Start the program in a batch/server job

When the specified user reaches the breakpoint in program

- The user’s job stops, notification msg appears on the STRDBG job
- Press F1 on the notification message
  - In the text there, find fully qualified job name mentioned for the STRSRVJOB command (copy & paste works well here!)

From a separate 5250 session/job

- Enter (or paste) the STRSRVJOB and STRDBG commands
  - as directed in the notification message details, including job name

Set a new breakpoint & debug as normal

---


The notification message will contain some text such as  "Do a Start Service Job (STRSRVJOB JOB(123456/PGMRNAME/QPADEV000X). Then do Start Debug (STRDBG) on the spawned job from where the STRSRVJOB was done. Set a local breakpoint at or after the Service Entry Point. Return to the original job and press enter to release the spawned job."

After the second breakpoint has been set, the SEP breakpoint gets removed, so you can feel free to end the debugger on the first 5250 screen.

©Copyright Partner400, 2010-2016.