Distance Learning hands-on Labs
IBM Rational Developer for i

Maintain an ILE RPG application using

Remote System Explorer

Debug a CL/RPG program member

Lab Exercises  Lab04
## Contents

<table>
<thead>
<tr>
<th>LAB 4</th>
<th>DEBUGGING IBM I PROGRAMS USING THE INTEGRATED IBM I DEBUGGER</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>OVERVIEW...........................................................................</td>
<td>9</td>
</tr>
<tr>
<td>4.2</td>
<td>INTRODUCING THE INTEGRATED IBM I DEBUGGER......................</td>
<td>10</td>
</tr>
<tr>
<td>4.3</td>
<td>STARTING A DEBUG SESSION USING A SERVICE ENTRY POINT..........</td>
<td>10</td>
</tr>
<tr>
<td>4.4</td>
<td>STARTING THE INTEGRATED DEBUGGER USING THE DEBUG ACTION......</td>
<td>30</td>
</tr>
<tr>
<td>4.5</td>
<td>DEBUGGING A JOB..........................................................</td>
<td>37</td>
</tr>
<tr>
<td>4.6</td>
<td>LAB SUMMARY.....................................................................</td>
<td>39</td>
</tr>
<tr>
<td>4.7</td>
<td>OVERALL SUMMARY LAB 01 TO LAB04....................................</td>
<td>40</td>
</tr>
</tbody>
</table>
Lab 4  Debugging IBM i programs using the integrated IBM i debugger

4.1  Overview
This module teaches you how to debug a CL and ILE RPG program. You will learn how to start the debugger, set breakpoints, monitor variables, run, and step into a program, view the call stack in the Debug view, remove a breakpoint, add a memory monitor, and set Watch breakpoints and all from the Debug perspective.

4.1.1  Skill level
Introductory

4.1.2  Audience
IBM i developer

4.1.3  System requirements
• IBM Rational® Developer for i, V7.5 and all software updates through the IBM Installation Manager.
• IBM i V5R3, V5R4, or V6R1

4.1.4  Prerequisites
Basic Microsoft Windows operations such as working with the desktop and basic mouse operations such as opening folders and performing drag-and-drop operations
• It will also help if you understand CL and ILE RPG.

This tutorial is divided into a number of modules, each with its own learning objectives. You can choose to skip the modules. You can go directly to the Debug module if you are only interested in that part, however you should complete module LAB01 to create a connection to the IBM i server. The connection will be used in this module.

Each module contains several lessons that must be completed in order for the tutorial to work as shown in this script.

4.1.5  Expected results
Upon completion of this tutorial you will know how to verify/compile an IBM i application from the Remote System Explorer. You will also know how to run an interactive program using the Remote System Explorer.

4.1.6  Conventions used in this tutorial
• Bold font for user interface controls
• Mono-spaced font for user input and code blocks
• Italic font for variable names and glossary terms.

Important!
You should go through Lab01 ‘Create a connection and get started with RSE’ before you work on this Lab. Lab01 contains the following information:
• Which IBM i server to connect to,
• Which userid to use,
• How to setup the correct library list for this lab.
4.2 Introducing the Integrated IBM i Debugger

The Integrated IBM i Debugger is a source-level debugger that enables you to debug and test an application that is running on an IBM i system. It provides a functionally rich interactive graphical interface that allows you to:

- View source code or compiler listings, while the program is running on an IBM i host.
- Set, change, delete, enable and disable line breakpoints in the application program. You can easily manage all your breakpoints using the Breakpoints view.
- Set Watch breakpoints to make the program stop whenever a specified variable changes.
- View the call stack of your program in the Debug view. As you debug, the call stack gets updated dynamically. You can view the source of any debug program by clicking on its call stack entry.
- Step through your code one line at a time.
- Step return, step into or step over program calls and ILE procedure calls.
- Suspend program execution and get control back to the debug session.
- Display a variable and its value in the Monitors view. The value can easily be changed to see the effect on the program’s flow.
- Locate procedure calls in a large program quickly and easily using the Modules/Programs view.
- Debug multi-threaded applications, maintaining separate stacks for each thread with the ability to enable and disable any individual thread.
- Load source from the workstation or a different IBM i system than the one the program runs on – useful if you don’t want the source code on a production machine.
- Debug client/server and distributed applications.

The Debugger supports RPG/400® and ILE RPG, COBOL and ILE COBOL, C, C++ and CL.

Now that you know the basic features of the debugger, let’s try them out.

4.3 Starting a Debug session using a service entry point

You will be working with the ILE RPG program PAYROLLG.

Instruction for creating a connection, which IBM i system to use, are contained in Lab01 of this series of Labs. If you haven't worked through Lab01, please do this first. The instruction in this Lab depend on the correct setup of a connection to an IBM i server.

Note:
PAYROLLG is the same RPG program as PAYROLL but without compile errors. You are using it instead of PAYROLL in this lesson, to accommodate anyone who decided to skip right to this Lab, without completing Lab02 or Lab03.

To make the lesson more interesting you will use CL program CLR1 to call PAYROLLG and you will pass one parameter to CLR1.

In this lesson, you will use a service entry point to start a debug session for your application. The service entry point feature is designed to allow easy debugging of applications that invoke business logic written in RPG, COBOL, CL, or even C or C++. The service entry point is a special kind of entry breakpoint that can be set directly from the Remote System Explorer. It is triggered when the first line of a specified procedure is executed in a job that is not under debug. Service entry points allow you to gain control of your job at that point. A new debug session gets started and execution is stopped at that location.
Tip:
To use a service entry point to start a debug session for your application and to allow updating of files in production libraries while debugging, the **Update Production Files** must be checked in the preferences of IBM i Debug.

To set the i Debug preference:
__i. Use Window >Preferences
__ii. Expand Run/Debug
__iii. Select IBM i Debug

Since you are using test libraries for the exercises, you don’t have to check this IBM i Debug preference.

To start a debug session using a service entry breakpoint:
__1. In the Remote Systems view expand the **Library list** filter, if it isn’t expanded already.
__2. Expand library RSELABxx, if it isn’t expanded already.
__3.  Right-click program CLR1 in library RSELABxx.

__4.  Click **Debug (Service Entry) > Set Service Entry Point** on the pop-up menu to set a service entry point.
A message displays indicating the service entry point was successfully set.

If you get this error message instead,

indicating that the Debug Server has not been started yet,

__i.  Right-click **Objects** in the Remote Systems view
__ii.  Select **Remote Servers > Debug > Start**
__iii.  Perform step 2 again.
The Service Entry Points view is automatically added to the stacked views at the lower right. It lists all the service entry points. You use this view to delete, activate, de-activate, modify and refresh service entry points.

1. Switch to the 5250 emulation session.

Troubleshooting
If you have done the exercises in Lab03, your 5250 session might still be associated with the RSE server job. You need to release the interactive session. To do so, in the Remote Systems view, right-click Objects and click **Release Interactive Job** on the pop-up menu.

On the command line of the 5250 screen, add the library RSELABxx to the library list and invoke the program CLR1, by entering the following commands:

2. ADDLIB RSELABxx and then
3. CALL PGM(RSELABxx/CLR1) PARM('XX') (where XX is your team number)

As soon as the program enters the system, the service entry point is hit and the debug session is started on the workstation and the perspective displays with the CLR1 source code in the editor. The Debug perspective gives you access to all available debugger features. Let’s look at some of them.

4. Click anywhere in the workbench to give it focus.

4.3.1 Setting breakpoints
You can only set breakpoints at executable lines. One way to set a breakpoint is to right-click on the line in the Source view.

To set a breakpoint:
1. Position the cursor on line 11.
2. Right-click anywhere on line 11.
3. Click **Add breakpoint** on the pop-up menu.
   A dot with a checkmark in the prefix area indicates that a breakpoint has been set for that line. The prefix area is the small grey margin to the left of the source lines.
   Now you add a conditional breakpoint to stop in the loop when it loops the 99th time.

### 4.3.2 Adding a conditional breakpoint

1. Select line 8.
2. Click the **Breakpoints** tab in the upper right pane of the Debug perspective.
   The Breakpoints view opens.

3. Right-click anywhere within the **Breakpoints** view.
4. Click **Add Breakpoint > Line** on the pop-up menu.
   The Add a Line Breakpoint window opens.
Tip: You can select an existing breakpoint by right-clicking it and selecting **Edit Breakpoint**.

5. Click **Next**. 
You only want to stop in the loop when it executes for the 99th time or more. You can do that by setting the From field of the Frequency group to 99.

6. Under Frequency in the From field, type 99.

7. Click Finish.

You have added a breakpoint including a conditional breakpoint to your debug session.

4.3.3 Monitoring variables

You can monitor variables in the Monitors view. Now you will monitor the variable &count.

To monitor a variable:
1. In the Source view, double-click the variable &count.
2. Right-click &count.
3. Click Monitor Expression on the pop-up menu.

The Monitors view opens.

The variable appears in the Monitors view. Its current value is zero.

Now that some breakpoints and a monitor are set, you can start to run the application.

4. Click the Resume icon from the Debug toolbar.
The program starts running and stops at the breakpoint at line 8. (Be patient, the Debugger has to stop 98 times but because of the condition continues to run until the 99th time.) Notice in the Monitors view, that &count now has the value 99.
5. Click the Resume icon again.
The program stops at the breakpoint at line 8 again and &count now has the value 100.
6. Click the Resume icon once more so that the program runs to the breakpoint at line 11.
7. If you do not see the error message below, go to “Stepping into a program” topic 4.5.3.

4.3.4 Error Handling
If you don't have any errors, skip to the next section 'Stepping into a program'
If you forget to add the parameter to the CALL program command when you call the program, you will see this error message.
_1. Click **OK**

_2. Click the **Terminate** icon on the Debug toolbar.

The debug session terminates on the workstation but the exception waits for input from the 5250 emulation session.

If you closed the Debug view by mistake, you will need to re-open the Debug view and then terminate the debug session on the workstation.

Here are the steps to re-open the debug view:

_1. Click **Window > Show View > Debug**.

Now terminate the Debug session if you haven’t done so already.

_3. Go to your 5250 emulator

_4. Enter C for cancel and press **Enter** until the program messages complete.
In the workbench,

5. Click the **Remove all terminated launches** icon on the Debug toolbar to clean up the Debug view.

To restart the program and start the debug session again:

On the 5250 command line, call the CLR1 program with the parameter XX.

6. Enter: `CALL PGM(RSELABxx/CLR1) PARM('XX').` **XX** being your team number

### 4.3.5 Stepping into a program

The Debugger allows you to step over a program call or step into it. When you step over a program call, the called program runs and the Debugger stops at the next executable statement in the calling program.

You are going to step into the PAYROLLG program.

To step into a program:

1. Click the **Step into** icon on the Debug toolbar.
IBM Software

The source of PAYROLLG is displayed. Depending on the option you used to compile the program (*SRCDBG or *LSTDBG for RPG, or *SOURCE, *LIST, or *ALL for ILE RPG), this window displays either the Source or Listing View.

If you specified an incorrect parameter for the CALL program command, or your library list does not include RSELABxx, you will see this error message.

Make sure your library list is correct and complete the same steps as covered in the section called Error Handling in “Monitoring variables” in topic 4.3.4.

2. Right-click anywhere in the Source view and click Switch view > Show *LISTING on the pop-up menu.

3. Page down in the source and take a look at the expanded file descriptions. You don’t have any /Copy member in your PAYROLLG program but these would also be shown in a Listing view. Switch back to the Source view.
4. Right-click anywhere in the Source view.
5. Click **Switch view > Show *SOURCE** on the pop-up menu.

You have stepped into PAYROLLG program, switched the view from source to listing and back to source.

### 4.3.6 Listing call stack entries

The Debug view in the upper left pane, lists all call stack entries. It contains a tree view for each thread. The thread can be expanded to show every program, module, and procedure that is on the stack at the current execution point. If you double-click on a stack entry you will display the corresponding source if it is available. Otherwise the message No Debug data available appears in the Source view.

In the Debug view, expand the stack entry of Thread1 if it is not expanded already.

![Debug view](image)

The stack entry allows you to work with and switch between different programs and/or ILE modules.

You have viewed the call stack entries of your program.

### 4.3.7 Setting breakpoints in PAYROLLG

Now you add some breakpoints in PAYROLLG.

To add breakpoints:

1. Select PAYROLLG in Thread1.
2. In the source view scroll to line 57.
3. Double-click the prefix area of line 57.

A breakpoint icon is added to the prefix area of this line to indicate that a breakpoint is set.

4. Repeat the above step for line 58.
5. Right-click in the prefix area of line 87 and click **Add Breakpoint** on the pop-up menu.

To view all breakpoints, select the **Breakpoints** tab from the top left pane.
This view shows all breakpoints currently set in your Debug session. This is a convenient place to work with breakpoints. You can remove, disable/enable, add, or edit a breakpoint. These tasks are available from the pop-up menu when you right-click in the view area. Double-click any entry to show the source where the breakpoint is set.

4.3.8 Removing a breakpoint in PAYROLLG

It is also easy to remove breakpoints.
To remove a breakpoint:
1. Right-click the prefix area of line 58.
2. Click Remove Breakpoint on the pop-up menu.
The icon is removed from the prefix area indicating that no breakpoint is set on that line. The breakpoint is also removed from the list in the Breakpoints view.

Tip:
Double-clicking on a breakpoint in the prefix area will also remove that breakpoint.

Now you are ready to run the PAYROLLG program.

3. Click the Resume icon from the Debug toolbar.
The program starts running and runs to the breakpoint at line 57.

4. Click the Resume icon again.
The program waits for input from the 5250-emulation session.

5. Type an X beside the Project Master Maintenance option.

6. Press Enter in the emulation session. The program runs to the breakpoint at line 87.
You have removed a breakpoint from PAYROLLG and started to run the program.

4.3.9 Monitoring variables in PAYROLLG
Now let’s monitor variables and change them in PAYROLLG.
To monitor variables:
1. In the source view, double-click the variable EMPAPL on line 88.
2. Right-click the variable.
3. Click Monitor Expression on the pop-up menu.
4. Click the **Monitors** tab in the upper right pane. The variable appears in the **Monitors** view. Its value is blank because you did not select the **Employee Master Maintenance** option.

5. In the same way add the variables PRJAPL on line 91 and RSCDE on line 113 to the monitor. Variable PRJAPL equals $X$ because you did select the **Project Master Maintenance** option.

6. In the **Monitors** view, double-click the variable RSCDE. The value changes into an entry field.

7. In the entry field, type in the new value $X$ for the variable.

7. Press **Enter**.
The variable is successfully changed.

### 4.3.10 Adding a memory monitor
Adding a memory monitor for a variable allows you to view the memory starting with the address where the variable is located. The memory can be displayed in different formats, for example hexadecimal and text.

To add a memory monitor:

1. In the **Source** view, double-click the variable ERR in line 33.

2. Right-click and select **Monitor Memory > EBCDIC** on the pop-up menu.
3. Use the scroll bar on the right of the Memory view to scroll down. You can see the current content of the memory.

4. Right-click in the view area.

5. Click Reset to Base Address on the pop-up menu to return to the starting address.

6. To get the hex content of the memory starting with the selected variable, click the tab New Renderings and select Hex for example. A new page with the hex values is added to the Memory view.

7. Click the Toggle Split Pane icon to display the character values as well.

You have added a memory monitor for the variable ERR.
4.3.11 Setting Watch breakpoints

A Watch breakpoint provides a notification to the user when a variable changes. It will suspend the execution of the program until an action is taken.

To set a Watch breakpoint:

1. Go to the Line number field at the bottom of the source area. In this field enter 116 to go to that line.

2. Double-click variable *IN60 to highlight it.

3. Right-click and click Add Watch Breakpoint on the pop-up menu.

The Add a Watch Breakpoint window opens. The Expression field is pre-filled with the highlighted variable *IN60.

By default the Number of bytes to watch field is set to zero, which means the variable will be watched in its defined length.

4. Click Finish. The Watch breakpoint is now set.

5. Click the Resume button on the Debug toolbar.

The application waits for input from the 5250-emulation session.
In the 5250 emulation session, type:

6. 123 for Project Code and D (for delete) in the Action Code field.

7. Press Enter.

A message is displayed indicating that the variable *IN60 has changed.

8. Click OK. The program stops at line 465. This line is located immediately after the statement which caused the variable *IN60 to change.

You have added a Watch breakpoint for the variable *IN60 and run the program to see the notification that the variable has changed.

4.3.12 Terminate a debug session

To close the debugger:

1. Click the Resume icon on the Debug toolbar. The application waits for input from the 5250 emulation session.

2. Switch to the 5250 emulation session.

3. Press F3 to end the program.

A message Program terminated appears:
4.4 Starting the Integrated Debugger using the Debug action

Besides using service entry points to start a debug session, you can start the Debugger in several ways: directly from the pop-up menu of a program or service program in the Remote Systems view, or from a Launch Configurations window. Starting directly from the Remote Systems view without prompt doesn’t allow you to specify parameters to be passed to the program. The Launch Configurations window allows you to modify how the program is invoked and to specify parameters.

CLR1 requires a parameter.

Before starting the debugger you need to associate a 5250 emulation session with the connection in the same way you did for running the program.

1. In the command line of the 5250 emulation session, type the command `STRRSESVR connectionName`
2. Press Enter. The `connectionName` parameter is the name of your connection defined in the Remote Systems view. This associates the interactive job with the Remote System Explorer communications server.
3. Return to the workbench.

To start the debugger:

5. Right-click and select Debug (Prompt) > Interactive from the pop-up menu.

The debugger prompt window opens for the selected program. The connection is prefilled with the name of the current connection. The program to be debugged is also prefilled with the library, program and type.
6. In the Name field, type the program name CLR1. This gives your debug configuration a unique name so you can use it again when you debug this program.

Note:
If you invoke the debugger even without prompting, a launch configuration with the default configuration named My program (Interactive) will be reused.

7. You can leave the Step into and Terminate debug session on program completion check boxes selected and Update production files check box deselected, since you are working with a test library.

8. Click the How To Start tab.
By default, the page contains a call for the program specified in the **What To Debug** tab.

9. Click **Prompt**.

The Call Program (CALL) window opens.
10. In the **Parameters** field, type ‘XX’ where ‘XX’ is your workstation number.
11. Click **OK**.

The complete start command for the program appears
12. Click Debug.

The Debug perspective opens. If not, you may see this message.

The interactive connection has been shut down in the meantime. Go to your 5250 emulator and restart the interactive connection following the instructions in the message. You don’t have to cancel the message. It will be removed as soon as the connection between the Remote System Explorer communications server and the interactive session has been established. The Debug perspective is displayed in the workbench.
Now that the program is active on IBM i and stopped at the first executable statement, the debugger displays the source and the debug functions are available.

You have started an interactive debug session.
__13. Remove all breakpoints. In the Breakpoints view, click the Remove All Breakpoints button in the toolbar or select Remove All from the pop-up menu.

__14. Click Resume

PAYROLLG is called and waits for input from the 5250 session. Only Terminate and Suspend buttons are available on the Debug view toolbar. To get control back to the Debug session:
__15. Click Suspend
You can now set breakpoints and use all the Debug features.

Tip: Suspend is a valuable feature to debug a looping program.

__16. Click Terminate to end the Debug session.

The Debug session is terminated, but this does not end the program.
__17. Leave the program running, we will use it again a little later.
__18. Switch to the Remote System Explorer Perspective

Window > Open Perspective > Remote System Explorer

Tip: You can edit, delete and create debug configurations by clicking the arrow beside the Debug icon on the workbench toolbar and selecting Debug Configurations from the list.
You can also click **Run** on the workbench menu and select **Debug Configurations**. The Debug Configurations window opens.

Here you can see the CLR1 configuration that you just created. This is your saved configuration to debug CLR1 as
an interactive application. You could now modify this configuration to use a different parameter, copy this configuration, or create a new one. Notice the list of configurations you can choose from.

You are now ready to remove the service entry point you created earlier and close the debug perspective.

To remove the service entry point
__19. Click Close in the Debug configuration window, if it is still open.
__20. In the Remote System Explorer perspective, expand library RSELABxx, if it isn’t expanded already.

__21. Right-click program CLR1 in library RSELABxx
__22. Click Debug (Service Entry) > Remove Service Entry Point.

The service entry point is removed.

You have started the debugger using a debug action, and removed a service entry point.

4.5 Debugging a Job

In addition to being able to debug a program, you can also debug a job. To debug a job:
In the Remote System Explorer perspective, under your active server connection, `s400a`,

1. expand Jobs > My Active Jobs > QINTER

2. Right-click the active job under QINTER, and select Debug As > IBM i Job.

The debug session begins and connects you to the running application.
Now you can set breakpoints, monitor variables and memory in the same way you did before.
3. Terminate the debug session by right-clicking the job in the Debug view and selecting **Terminate** from the pop-up menu.

The debug session is terminated

### 4.6 Lab summary

In this module, you learned how to debug a program using the Integrated IBM i Debugger.

**Lessons learned**

- Start a debug session using service entry points
- Add a breakpoint
- Add a conditional breakpoint
- Edit a breakpoint
- Monitor a variable in the Monitors view
- Step into your payroll program
- Show a Listing view
- Display source from call stack entries
- View all breakpoints
- Remove a breakpoint
Monitor memory
Set a Watch breakpoint
Close the debugger
Invoke the debugger from a Debug Configurations window.

4.7 Overall summary Lab 01 to Lab04

This tutorial has taught you so far how to maintain a payroll application using the Remote System Explorer. You learned how to start the product and open the Remote System Explorer perspective and how to use tools and views in this perspective to connect to an IBM i system and edit, verify, compile and debug the payroll application.

The other labs available will guide you through using the Screen designer to edit DDS display file source and using the Application Diagram Viewer to visually show the Call hierarchy of procedure/subroutines/programs in a source member or show the binding relations of ILE programs and service programs.

Congratulations!

You have successfully completed the introduction to the RDi Debugger.

Feel free to exploit the other Labs that are available.

More information and material can be found at our RPG CAFE


Look for the RDi hub
Enjoy Rational Developer for IBM i!
Notices

© Copyright IBM Corporation 1992, 2009 All rights reserved.

U.S. Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

This information was developed for products and services offered in the U.S.A. IBM may not offer the products, services, or features discussed in this documentation in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user’s responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this documentation. The furnishing of this documentation does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing IBM Corporation
North Castle Drive
Armonk, NY 10504-1785 U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation Licensing
2-31 Roppongi 3-chome, Minato-ku Tokyo
106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:
Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this documentation and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM’s future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Copyright license

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. ©Copyright IBM Corp. 1992, 2009. All rights reserved

Trademarks and service marks

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

- i5/OS
- IBM
- OS/400
- Rational
- WebSphere

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States,
other countries, or both.

Other company, product or service names may be trademarks or service marks of others.