

*Paris  
Gartner400*

# Using CODE

**Lab Instructions**



# INTRODUCTION

The **CoOperative Development Environment/400**, better known as **CODE/400** or **CODE**, is a set of integrated development tools that allow you to: create; edit; compile; and maintain your source code. It also provides facilities for debugging programs using a connected PC, and completely organize your programming projects.

With the advent of V5R1, CODE was made a feature of the IBM WebSphere® Development Tools for iSeries (WDT/400) product, which was in turn included with IBM WebSphere Development Studio for iSeries (WDS/400). Subsequently with V5R2, IBM renamed the client-side tools as WebSphere Development Studio for iSeries and (just to confuse us) identified it as V4. This last part is not quite as silly as it sounds. The idea is to match the version number of the toolset with the version of WebSphere that it supports, rather than to an OS/400 release number.

CODE includes the following tools:

## **CODE Editor**

A powerful language-sensitive editor that you can easily customize to meet your own requirements. Token highlighting of source makes the various program elements stand out. It has SEU- like specification prompts for RPG and DDS to help enter column-sensitive fields. Local syntax checking and semantic verification for your RPG, COBOL and DDS source makes sure it will compile cleanly the first time. If there are verification errors, an Error List lets you locate and resolve problems quickly. On-line programming guides, language references, and context-sensitive help make finding the information you need just a keystroke away.

## **CODE Program Generator**

An interface that allows you to submit requests to OS/400 to compile, bind, or build objects on the host. The tool gives you easy access to all the compile options available for all the supported create commands (CRTxxx).

## **CODE Designer**

A rich graphical interface that makes designing or maintaining display file screens, printer file layouts, and physical files easy and fun.

## **CODE Debugger**

A source-level debugger that allows you to use your workstation to debug an application that is running on OS/400. It provides an interactive graphical interface that makes it easy to debug and test your host programs.

## **CODE Project Organizer (CPO)**

An enhanced and more flexible workstation version of the Program Development Manager (PDM). It ties all the parts of CODE together and allows you to quickly access all the power of CODE and to effectively manage and organize your development projects.

Although we will touch on the major functions of CPO in this class, we will not study it in depth. The reason for this is that IBM have largely replaced this component in the current version of the product. The replacement for CPO is significantly different from the current version. If you want to get a preview of what the new version looks like, check out IBM's Eclipse project.

## **The Goal**

Our goal is to teach you the basics of using the features and functionality of the CODE tools by playing with them. We are confident that CODE will save you time and effort in your day-to-day programming tasks. It will make you a more efficient and effective programmer. Now let's spend a few hours "playing" and see if you agree.

### **Note:**

The re-branding of "The system formerly known as the AS/400" causes many problems when producing a document such as this. Should we refer to the system as "AS/400" or "iSeries" or use both all the time? Here's how we've handled it.

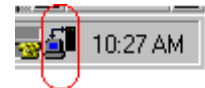
Rather than try to use both names, or have to pick between them, we have instead chosen to use the terms "Host", "Server" and "OS/400" interchangeably depending on the context. We hope that you will find this less confusing.

## Connecting to OS/400

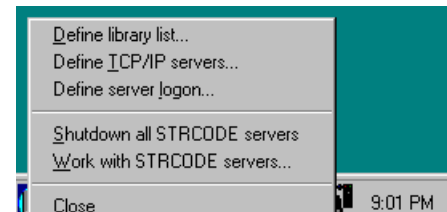
CODE communications between the system and your workstation is normally configured for TCP/IP communications using the native Windows built in TCP/IP support. If you chose to use an STRCODE server (more on this later) you can choose to run it in batch or via an interactive sessions using any 5250 emulator that supports TCP/IP.

CODE can use two kinds of server connections to your OS/400 system. The first one we will look at, normally uses a 5250 session for the server link. This is known as an STRCODE server. The second is an auto start server that does not require the use of a 5250 session. While the 5250 version is more work to start, it is currently the only way to access all features of CODE. For instance prompting of CL source can currently only be performed via a 5250 linked session. With the latest release this is no longer strictly true as IBM have released a graphical extension to CODE that make use of new Client Access CL prompting facilities. However this feature is not enabled by default.

Before using either type of CODE server, on your workstation, the CODE “Daemon” needs to be running in order to allow TCP/IP communication with the system. When CODE is installed, it places the daemon in your **Start Up** folder so that whenever your PC is restarted, the CODE daemon is started for you. Check to see if it active by looking at the system tray in the bottom right hand corner of your screen. You should see an icon that looks like the one circled here.



The daemon waits and listens for the host to contact it on a specific TCP/IP port (by default port 4200 is used) and then makes the connection. You can interact with CODE communications by using the right-click pop-up menu of the daemon’s icon as shown here.



Sometimes when you experience communication problems, it may be necessary to close the daemon. This can be done from the pop-up menu as shown above. It is good practice to shutdown all servers (which can be done from this same menu) before requesting that the daemon close. Once the daemon has been closed it can be re-loaded manually by going to:

**Start – Programs - IBM WebSphere Development Tools for iSeries - Communications - Communication Daemon.**

Ensure that the CODE daemon is running and start it as indicated above if it is not. The latest version of CODE does this automatically when you attempt to use any host related function.

## CODE Library Requirements

**NOTE:** It is *CRITICAL*, regardless of which type of connection you use (STRCODE or Auto Start), that you have a non-IBM library specified as your **CURRENT LIBRARY** (\*CURLIB). This current library should be one that no other programmers will be using for their current library. If your user ID does not have a current library specified or if it is not unique to you (e.g., QGPL), you need to take action to get a unique current library BEFORE you attempt to connect using CODE. Otherwise, you WILL have problems eventually. Ask your instructor for options on how to accomplish this.

Note that a current library that is set by an Initial program will not work this is because the Initial Program entry is ignored when starting server jobs.

In this lab, you may choose which type of code server you want to use: STRCODE or Auto Start. There are a few places in the lab instructions which may be written with the assumption that you are using the STRCODE server. In these cases, you will need to make some small adjustments if you decide to use the Auto Start server. Your instructor can advise you on this.

Make your choice on which type of CODE server to use and proceed to the appropriate page to start that type of CODE server. Don't forget to make sure you have a unique current library assigned before proceeding. If you do not, ask your instructor for help now.

## The CODE Editor and Firewalls

If you are using an STRCODE server, then port 4200 must be open on the firewall and/or VPN circuits being used. The default port number of 4200 can be changed if desired.

If you are using an auto start server, it is not normally necessary to open the port as most firewalls will allow a response to be returned to any port within the firewall that initiated a request.

Other CODE components, notably the debugger, have their own port requirements.

## Starting the STRCODE Server

- ❑ Start a 5250-emulation session and sign on to the server. Unless you are using your normal User Id, your User Id and password for the class will both be **CODELABxx** where **xx** is your team number (01, 02, etc.). If this is not the case, your instructor will notify you of the correct sign on.
- ❑ If your host (iSeries) server is at V5R1 or later, at the OS/400 command line, enter the command **STRCODE** and press F4 for prompting. For **Remote Location Name**, enter the special value **\*RESOLVE**. This will identify the IP address of your workstation for you.
- ❑ If your host system is not yet at V5R1, you will need to use a different command. Enter the command **STRCODETCP**. This will call a CL program, which automatically identifies the IP address your PC is using and invokes the **STRCODE** command for you with the correct information filled in. This command and its associated CL program (**STRCODET**) can be found in the QCLSRC source file in your **CODELABxx** library. The CL program is a simplified version of one previously published by IBM on the CODE web site.
- ❑ If everything goes according to plan, you should see a screen that has **EVFCLOGO** in the upper left-hand corner, and details of the server's name in the center. You may also hear a rising sequence of tones from your PC that indicates that communication has been established. Do not wait for this screen to go away, it will not. This is perfectly normal, the display will remain until you close the CODE server.
- ❑ If you did not have this CL program, you could type or prompt the **STRCODE** command directly as follows:

**STRCODE RMTLOCNAME(PC\_hostname) CMNTYPE(\*TCPIP)**

The entry 'PC hostname' should be either the TCP/IP host name of your PC (if your PC's name is known on the network) or its dotted IP address in quotes, for example '9.21.99.99'. You can determine your host name, by entering the DOS command **hostname** at a DOS command line.

CODE does not require you to be continuously connected to the host -- many of its features are designed to function in 'disconnected' mode as well. This gives you the ability to develop your applications at home or while traveling (oh joy!). In fact several features, like the caching of file information (file definitions used by programs, database reference fields, etc.) on your workstation, is designed to help you work in 'disconnected' mode. We will be exploring these features later.

## The AutoStart Server

This alternative method of communication requires that the Communication Console be used to identify a server and (optionally) sign on information. Once this has been done, CODE will automatically start a communication server whenever one is needed, for instance when you request to download a source member.

You can start the Console by going to:

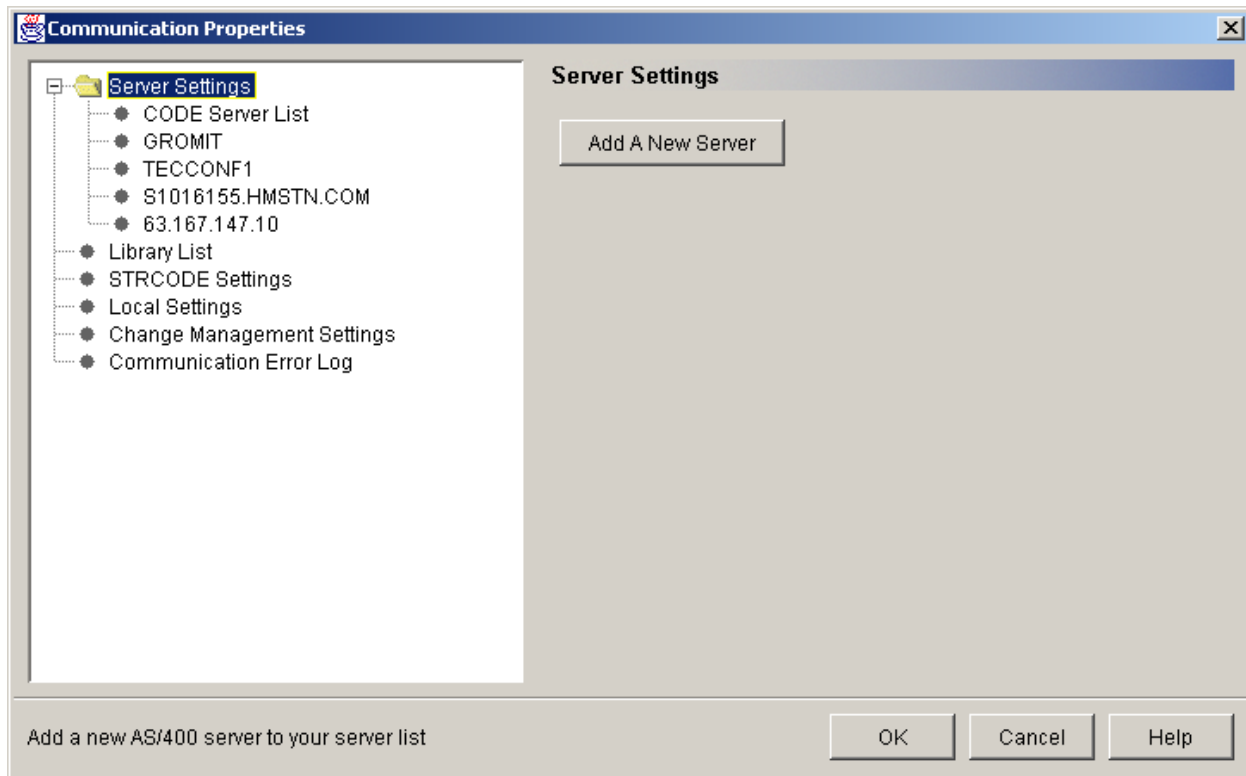
**Start - Programs - IBM WebSphere Studio - Development Studio for iSeries - Communications - Communication Console**

Alternatively, you can start the Console from within the editor by going to:

**Windows - Communications – Properties**

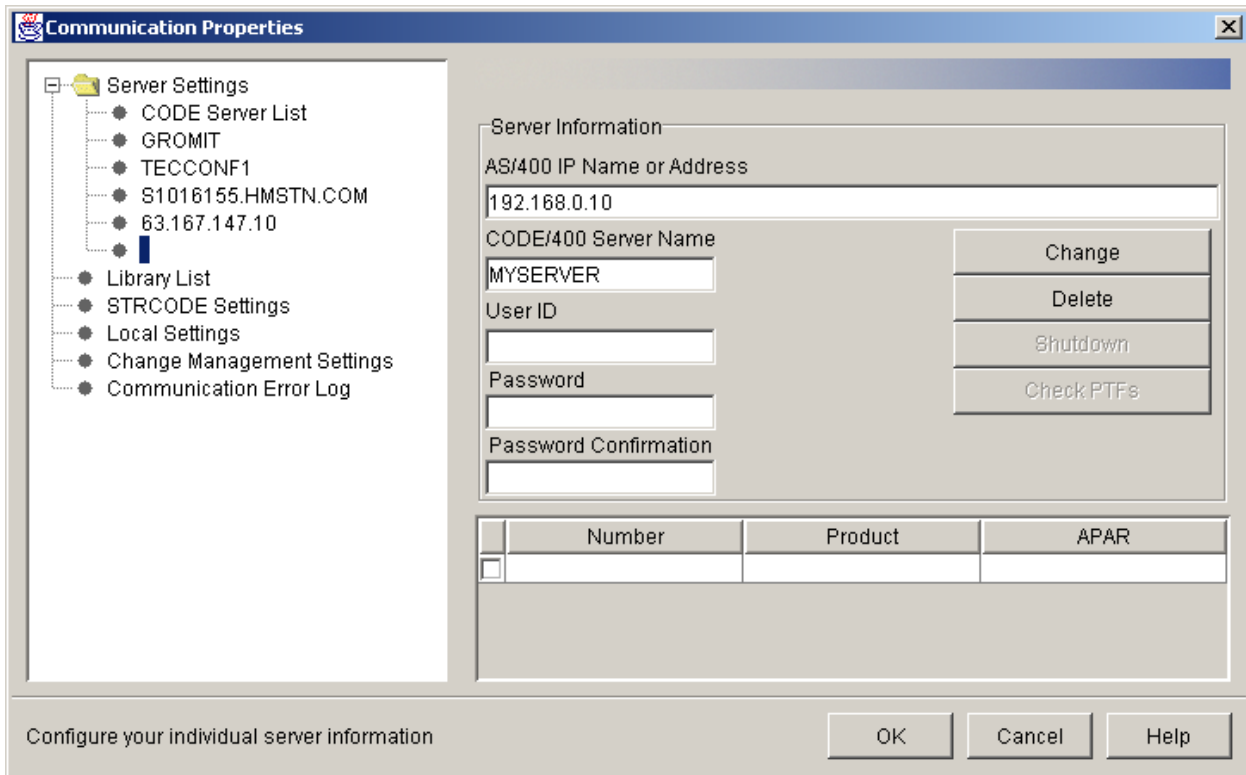
The Console can also be used for defining the current library and other specific entries that need to be added to the library list for a specific server.

The Communications Console looks like this when it starts. (Note that the console is written in Java and can take quite a long time to start). Yours will look a little different because you will not have any servers defined, or one at most. If the server that you will using in the class is not defined yet, follow the steps below.



## Defining a New Server

- ❑ Click on the Add a New Server button and the following window should appear.



- ❑ Enter the name (or IP address as shown in the example) of your server and the name which you wish it to be known by (in our example MYSERVER). If you wish, you can also enter your User Id and password. This saves CODE from having to prompt you for this information and makes the whole process seamless.
- ❑ Next, press the Change button. Yes, we know this is silly and non-intuitive, but unless IBM fixes it, this is the way that it works.
- ❑ Now that you have defined your server, you can exit the Console with a Press on the OK button. That is not a very good choice either but ..... If you prefer, check out some of the other options before you exit.
- ❑ Later when you start the CODE editor and use the File -> Open pull down you should find your newly created server in the list of available servers.
- ❑ Note: If you already have the **File Open** dialog open, your new server will not appear until after you have closed and then re-opened this dialog.

**OK, TIME TO MOVE ON TO THE MAIN PART OF THE LAB. ENJOY!**

## The CODE Editor

The CODE Editor has many powerful features that make it the premiere editor for iSeries and AS/400 programmers:

- It supports RPG III and RPG IV, COBOL and ILE COBOL, C, C++, DDS, CL, REXX, HTML and Java language-sensitive editing.
- Seamless access to OS/400 source members, and local files.
- Token highlighting of source code elements.
- SEU-like prompting to help you edit column-sensitive languages such as RPG and DDS.
- Built-in syntax checking and program verification (a compile with no object generation) on the workstation for RPG, COBOL, and DDS. With this you can work while disconnected from the system.
- Lots of on-line help including programming guides and language references.

In this section, you will be introduced to some of the features which make the CODE Editor so powerful. For the purposes of the exercise, we will be using some RPG IV source. Let's get started.

### Starting the Editor

You can start the editor from the Windows Start menu, a DOS prompt, or CODE Project Organizer:

- From the **Start - Programs - IBM WebSphere Studio - Development Studio for iSeries - CODE**, select CODE Editor.
- The CODE Editor will then appear.
- Use the mouse to select some of the pull down menus. Note that the "View" and "Actions" menus in particular currently have very few options. The content of these menus are highly dependent on the type of source being edited. In a few moments when we open a source member, you will see a big difference.

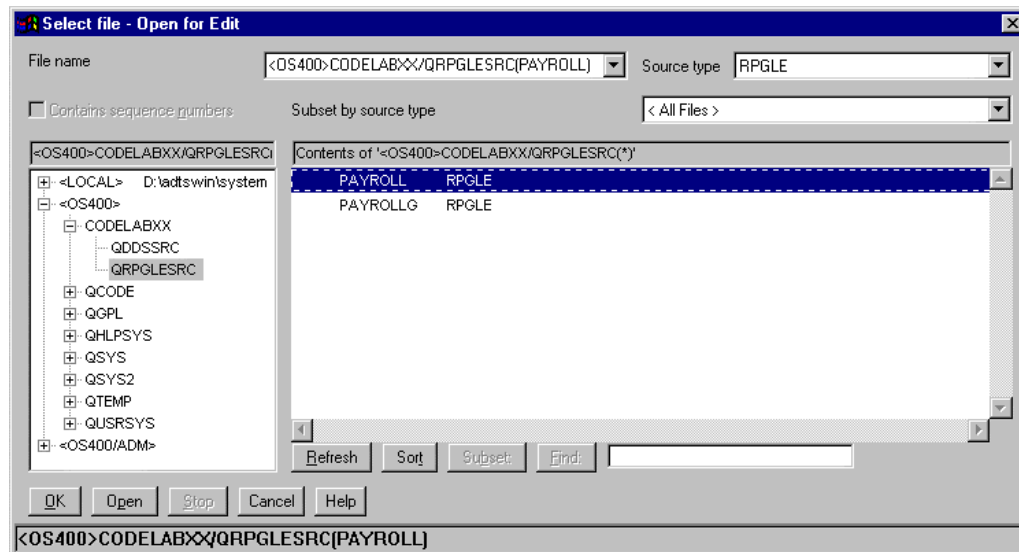
## Opening a Source Member

The editor provides seamless access to workstation files and to source members on the host. Among other facilities supported by the editor:

- Source members are automatically downloaded to the workstation.
- Members are locked during editing just as they are in SEU
- OS/400 members and local files have a source type ( DSPF, RPGLE, RPG ...) associated with them. This allows the editor to load the file in the correct language environment.
- While you need a host connection to edit OS/400 files directly, if you download the files to your workstation then you can work on them without a host connection and then upload them whenever you want.

**Note:** Remember, in the instructions that follow the **xx** part of text such as CODELAB**xx** should be replaced by your assigned team number. In addition the text **< OS400 >** may need to be replaced by the name of the OS/400 server that you are using for the Lab.

- From the **File** menu, select **Open...** This will bring up an Open dialog similar to the one shown here:



- Click the + beside **<OS400>** to show the OS/400 library list.

## INTRODUCTION TO CODE

- Click the + beside **CODELABxx** to show the source files in that library.

**NOTE:** If your **CODELABxx** library does not appear in your list of libraries, take the following actions. Otherwise proceed to the next numbered lab step.

**Cancel** the File Open Dialog box. Then Press **F9** (or use the Windows pull down menu to select Command Shell). Check to make sure your correct system name appears on the last command line (where your cursor is positioned). If it is not the correct system name, then use the **Options** pull down menu and select **Servers**, then select your server name from the list.

On the command line where your server name appears, enter the command **ADDLIBLE CODELABxx**. You should see a confirmation that **CODELABxx** was added to your library list.

Now use the **File menu** and select **Open** (as before). If the list of libraries for <OS400> is still expanded (i.e., there is a – sign in the box next to the system name), close the list by clicking on the – sign. The – sign turns into a + sign and the list of libraries disappears.

**Expand** the library list again (click on the + sign next to <OS400>) and your **CODELABxx** library should now be in the list. Click on the + there and proceed with the rest of the lab steps.

- Click on the source physical file **QRPGLESRC** to show its contents on the right.
- Click on **PAYROLL RPGLE**.
- Click on the **OK** push button. The source will then be downloaded and brought into the editor.

Note: As an alternative to pushing **OK**, you could double-click the source member name. However, when you want to edit multiple source files, the best method is to select a member and press **Open** to bring it into the editor. Continue to do this until all of the source members you wish to work with have been loaded. You can either press **OK** to load the final one in the group or simply **Cancel** the dialog when you have finished.

You are going to be working with the edit window for quite some time, so take the time now to size and position the window the way you like it to create the most comfortable working environment for you.

### **An Alternative Method for Opening Files**

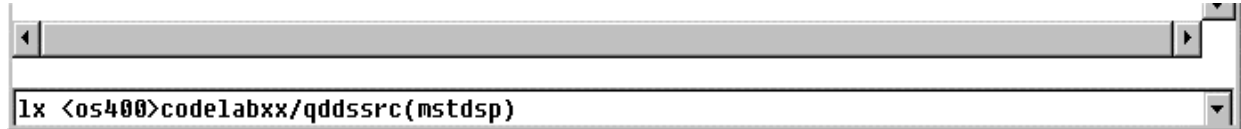
If you know the name of the source then you can load it without starting the Open dialog:

## INTRODUCTION TO CODE

- Press **Esc**. Notice that the cursor moves to the command line at the bottom of the CODE Editor window. If you press **Esc** again you will see that the cursor moves back to its normal position in the editing window.
- Press **Esc** one more time to return to the Command Line and type in the following edit command:

**LX <OS400>CODELABxx/QDDSSRC(MSTDSP)**

where '**OS400**' the name of your server and '**xx**' is your team number.



- As you will see, the DDS source replaces the RPG source in the viewable part of the screen. However, the RPG source member is still open. You can easily switch between files in an edit session in a number of different ways. Let's look at a few.
- Notice that there is a blue bar containing the details of the source file, immediately below the tool bar in the edit window. **Click** anywhere in this bar and a list of the files currently being edited will appear. From this list, select the member that you wish to make the active member.
- Here's an alternative method for switching between open members. Hold down the **Alt** key while pressing the **right cursor arrow** key (bottom right side of your keyboard). This switches your view. Press **Alt+Right Arrow** again. See how it works? Try the left arrow key instead with the Alt key. This is a quick and easy way to switch between multiple files. Practice this switch a few time until you are comfortable with how to do it. Notice that you will sometimes see a blank edit screen. This initial file is created whenever the editor is started with no specific file specified. This file stays in your session until the session ends.
- Before you continue, make sure you have switched back to the RPGLE PAYROLL source member.

## Basic Editor Features

The CODE Editor has all the basic functions that you would expect in any serious editor:

- Cut, copy, and paste
- Block marking of lines, characters, or rectangles and with copy, move, overlay, and delete operations.
- Powerful find and replace functionality.
- Unlimited undo and redo.
- Automatic backup and recovery.

## INTRODUCTION TO CODE

In addition, there are a number of functions that you may not have seen before in a workstation editor:

- Token highlighting - different language constructs are highlighted using different colors and fonts to help identify them in a program.
- SEU-like format-line rulers to show the purpose of each column for column-sensitive languages like RPG and DDS. These rulers automatically update themselves to reflect the current specification.
- SEU-like specification prompting for RPG and DDS.
- Sequence numbers, which allow SEU-style commands in the prefix area.
- Intelligent tabbing between columns for column-sensitive languages.
- Automatic uppercasing in languages that only permit uppercase text (e.g. RPG400, DDS and CL).
- For column-sensitive languages there is a CODE FIELDS command that simplifies text insertions and deletions.
- On-line language reference.

Now let's take a minute to try a few of these features.

### Intelligent Tabbing

For fixed format languages, such as RPG or DDS, intelligent tabbing is a particularly useful feature. Let us see how it works.

- Position your cursor on any of the "live" F specs (they start around line 27). Look at the top of your edit screen (just below the Row/Column status line) and you will see the format line for the F spec. Using the cursor arrow key, move your cursor up to one of the comment lines. Notice that the format line above changes as you move.
- Reposition the cursor on the F spec and press the **Tab** key a few times. Watch the movement of the cursor and the highlighted portion of the format line change as you do so. You may also press the **Shift+Tab** key to tab right to left in the same fashion.
- Now move your cursor down to one of the C specs (they start around line 47). Notice that the format line has automatically switched to a C spec. Press the **Tab** key to get to the Op-code field. Press **Tab** again to get to Factor 2 and then again to reach the for Result. Keep pressing the **Tab** key until you get to the HiLoEq indicators on the C spec. Notice how the highlighting on the format line above makes it very obvious which indicator is which. If you are still having to work with RPG code that makes extensive use of resulting indicators, this can come in very handy!

## SEU Commands

If you are an SEU expert, you will appreciate CODE's ability to use many of the SEU line commands:


- Start by moving the cursor into the sequence number area to the left of the edit area.
- Type **dd** on any sequence number just as you would in SEU.
- Move down a few lines and type **dd** again, then press **<Enter>**. Notice that the lines have been deleted just as they would be in SEU.

**Note:** The cursor must be in the sequence number area for **<Enter>** to activate them. This applies to all of the SEU-style commands. If you wish you can press **Alt+<Enter>** - this has the effect of activating sequence area commands no matter where the cursor is located.


- Now type **i5** in the sequence number area and press **<Enter>**. Five new lines are inserted.

## Undo and redo

Now you are going to undo some of the changes you just made to the file. This is something that SEU certainly can't do easily! As you will see, SEU's F5 is not even close.

- First take a look at the tool bar. Notice that since we have been making changes to the source, the Undo button (circled) is now active. 

- From the **Edit** menu, select **Undo**, or use the **Undo** button if you prefer. Notice that the 5 new lines disappear.
- For those who do not like removing their hands from the keyboard to use the mouse, there is an alternative method. Do another undo action by pressing **Ctrl+Z**. Notice that the deleted lines reappear.

- Notice that now that we have "undone" some of our source edits, the Redo button (circled) is now active.  Either use this button, or from the **Edit** menu, select **Redo**. Notice that the lines are deleted again.

At this point, we would like you to you will reload the source from the host to make sure that it is back to its original form. To do this:

- From the **File** menu, select **Close view**. A dialog will appear asking if you want to save the latest changes. Click the **No** button. Notice that **F3** is the keyboard shortcut for Close view.
- From the File menu, select <OS400>CODELABxx/QRPGLESRC(PAYROLL) to reload the original source.

## Using Language-Sensitive Help

Inside the editor, there is cursor-sensitive language-reference help available. This help is invaluable if you cannot remember the order of fields in an RPG specification or the possible values for a variable field. This help is available from within the CODE Editor window, or from the Help pulldown.

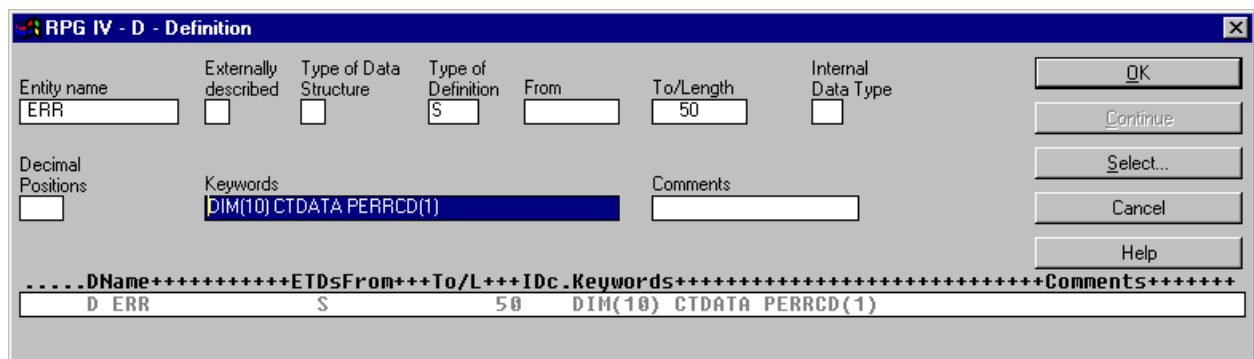
- Position the cursor over the op-code **MOVE** (around row 5600 of the source).
- Press **F1**. Language-sensitive help for the MOVE operation code appears in a browser window. The Help text will appear almost immediately, if you already have a browser open. If you do not have a browser open, it will take a few seconds (to start the browser) before the Help text appears. In some cases, the browser window may not automatically get focus. In this case, simply click on the browser window in the Windows Task Bar at the bottom of your screen.
- Scroll to the bottom of the help page and select the link for the **Table of Contents**. Notice that you now have access to the entire RPG Reference Manual. Take the time to check a few of the links. Notice in particular the number of hypertext links available to jump between pages. Follow a few of the links to get the feel for the system before you move on to the next step.
- Now minimize the Help browser window and select the main editor window if it does not automatically get focus.
- Select the **Help** menu, to see what other help is available. The **ILE RPG Help** menu item will show you all the on-line help that is available for RPG IV. Select any menu item to find more information. Notice that there is a fast-path option to the list of Built-In Functions, this can be very useful when first learning to use these. Minimize the Help window when you are done.
- Last but not least, is the single most useful piece of information in the entire Help system! Select the **Keys Help** item from the **Help** menu. This will show you a list of all the standard keyboard shortcuts available in the editor. As you can see it is quite a long list. If you have a printer available to you, you may find it helpful during your early use of

CODE to print this list, or cut and paste selected entries into your own “Quick Guide” for those keyboard shortcuts you find most valuable.

## Using Prompts

Instead of entering or changing code directly in the editor window, you can use prompts. When you request a prompt for a specification line, a window appears where you can enter or change that line using entry fields.

- Move your cursor to the D-spec definition of the array **ERR** on line 33.
- From the **Edit** menu, select **Prompt** (or press **F4** as you would in SEU). A window appears showing the specification broken down into its individual fields.



- Move the cursor to the Keywords field using the Tab key. Press F1 or click on the Help push button to see help for this field.
- You will see words in the help that appear in a different color than the regular text. These are help links, and they show that there is additional help available on that word or phrase. Click on any link to see specific help for that item.
- Minimize the Help window.
- There are three ways to close the prompt dialog: **Click** on the **Cancel** push button; Press the **X** in the window’s upper right corner; or Press **Esc**. All of these methods will close the prompt window without changing your source.

**Note:** When you use prompts to edit or add to your source and then click on the **OK** push button, the changes are added to the program in the appropriate column and the syntax is checked automatically.

Most CODE users find that they don’t use the prompt feature very often. Why? Because most of the time we only use prompting in SEU so that the Tab key will work! Since the Tab key works the way it should in CODE we don’t need to prompt to use it. Also, since the prompt line changes as you move from line to line, you can always tell which field your cursor is in. And of course, real context-sensitive help is always only a push of the **F1** key away.

## Indenting Source

When editing ILE RPG source, it can be difficult to determine the beginning and ending points of constructs such as DO loops. The indent option allows you to view your source with these constructs highlighted in an indented structure.

- Move your cursor to row 45500 by typing 45500 in the sequence number area and pressing <Enter>. If your source is not positioned to line 45500 with this action, make sure your cursor is positioned somewhere in the sequence number section of the edit screen before pressing <Enter>.
- Some CODE users prefer to use the **Locate Line** dialog rather than use the SEU method. You can try this for yourself by pressing **Ctrl+L**.
- Once you are positioned on the correct line, select **Indent** from the **View** menu.
- Notice that the editor goes into split screen mode, with the original source at the top and the indented view at the bottom. (\*\* In some releases CODE defaults to a vertical split) Unlike SEU's split screen mode, however, you can change this one to have a vertical split instead. From the View menu, select Split and then Vertical. This is probably a more effective way to view the effect of indent.
- Check what happened to the IFEQ - END blocks. Although indented views are not currently editable, it is very easy to see where changes need to be made and make them in the corresponding edit view.
- Click on the source in the INDENTED view and close it by pressing F3. You should now be back to editing the PAYROLL member.

## Match


Another useful feature of the editor that we will cover here since it serves a similar function to the indented view is "Matching". This allows you to simply identify the matching END (or indeed beginning) of any conditional block of code. Let us see how it works.

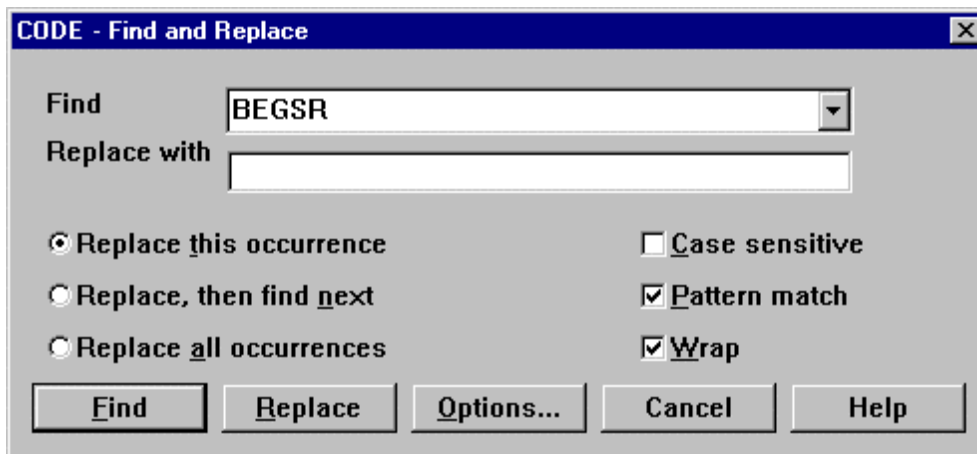
- Start by positioning your cursor on any **END** Op-code in the source. Now press **Ctrl+M** (or use **View – Match – Select** from the menu bar). You will see that all source lines in the corresponding IF, DO, etc. block are highlighted. In most cases, text blocks selected in the editor are de-selected when you click elsewhere in the edit window. This is not true of any text selected by any of the block marking operations such as this. To deselect such blocks **right-click** in the text and select **Deselect all** from the resulting pop-up menu.
- Another useful function is provided by the menu option **View – Match – Jump** (the keyboard shortcut for this is **Ctrl+Shift+M**). Position the cursor on any **IF** Op-code and select this option in whichever way you want. Notice that your cursor is immediately

repositioned to the corresponding **END**. This function works with BEGSR, ENDSR, DO, IF, etc. etc. Try a few other variations and you will soon see how useful this can be.

## Find and Replace

The CODE editor also has a very powerful Find and Replace feature.

- Start by pressing **Ctrl+Home** to go back to the top of the file.
- From the Edit menu, select the **Find and replace...** (or press Ctrl+F, or click on  in the toolbar). The Find and Replace dialog appears. Notice that we can specify if a case-



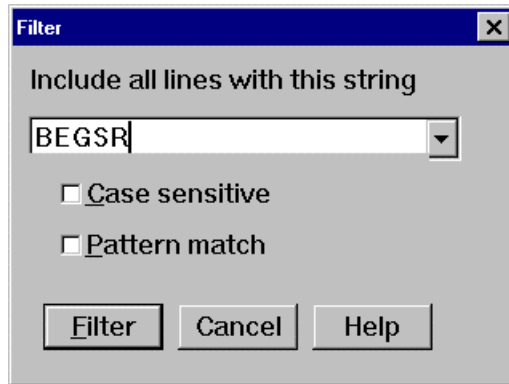
sensitive search is required. Notice that there are also a number of other basic options available. We will leave it to you to explore the Help text and find out more about these options.

- Click** on the **Options** push button to see some of the additional options available to us. For example we could search all the files loaded in the editor, search only in certain columns, etc.
- Click** on the **Cancel** push button to close the Options window.
- In the **Find** entry field, enter **BEGSR** to find the start of a subroutine. Make sure the **Replace with** entry field is blank. You would use this field for text replacement.
- Click** on the **Find** push button. The cursor will be positioned at the first BEGSR in the file.
- Press **Ctrl+N** to go to the location of the next BEGSR in the file. Alternatively, you could use **Shift+F4** or use the **Find next** option on the **Edit** menu.
- In addition to finding the next matching item, we can also move backwards through the source. To do this use **Ctrl+U** (you can also find this option on the **Edit** menu).

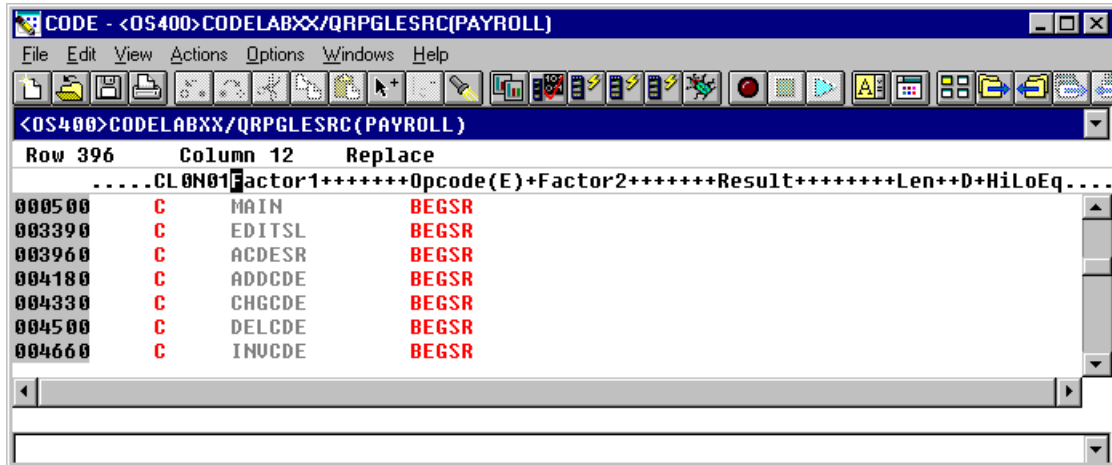
## Filtering Lines

The CODE Editor allows you to filter or subset your source so that you see only lines containing a given string. Filtering lines makes it quick and easy to find lines without having to scroll through your source.

- ❑ From the **View** menu, select **Filter** (or press **Ctrl+I**).



- ❑ Type **BEGSR** in the entry field and Click on the **Filter** button. Only lines containing the string **BEGSR** appear in the window, as shown below.



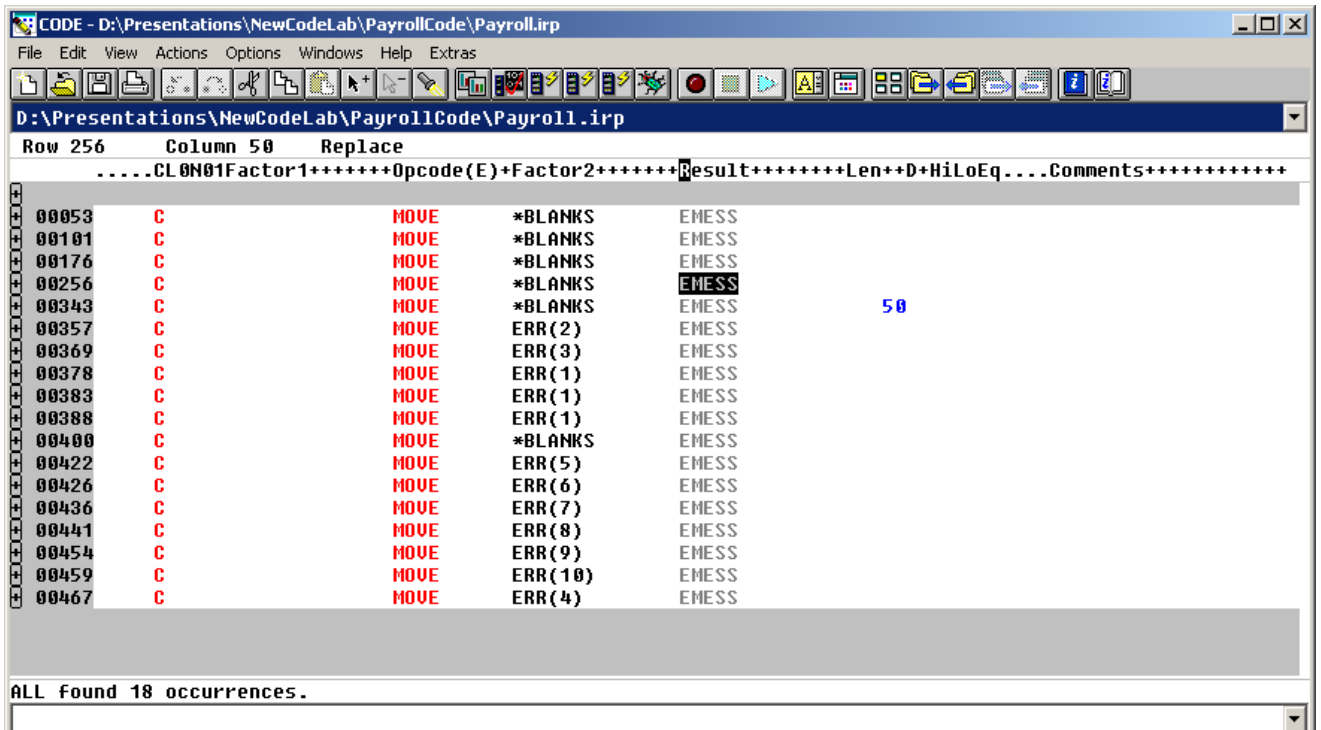
- ❑ Move the cursor to the **ADDCDE** subroutine.
- ❑ Show all of the source again by selecting **View** → **Show all** or by pressing **Ctrl+A**. Your cursor is still positioned on the same line that you moved it to, even though all lines are now showing.

## INTRODUCTION TO CODE

- A recent addition to CODE has made this filtering process even more useful. Move to the editor command line using your mouse or the **Esc** key and enter the command below and press **Enter**:

**set default.expandhide on**

- Now repeat the steps above to filter the **BEGSR** source lines. Notice that + (plus) signs now appear in the sequence area. These allow you to selectively reveal the “hidden” lines.
- **Click** on a few of the + signs to get a feel for how this works. You will see that when you click on one not only is additional source revealed, but the sign changes to a – (hyphen). Clicking on the – will cause the editor to “hide” the lines once again.
- There is yet another way of filtering text that is even easier to use. Place your cursor over the field name **EMESS** in the Result field of any C spec. It appears frequently, so it should be easy to find one. Select the EMESS field name by dragging your mouse over the field name. This will highlight the word.
- Now select the **Filter Selection** option from the **View** menu and you will see only those lines that contain this record name. Your screen should look similar to the example here. Move your cursor to any of the lines shown and press **Ctrl+A**. You should then see the full source, positioned to the line you selected. Alternatively you could click the ‘+’ sign beside the line you selected to cause that group of hidden lines to be displayed.



Row	Column	Replace
00053	C	MOVE *BLANKS EMESS
00101	C	MOVE *BLANKS EMESS
00176	C	MOVE *BLANKS EMESS
00256	C	MOVE *BLANKS EMESS
00343	C	MOVE *BLANKS EMESS
00357	C	MOVE ERR(2) EMESS
00369	C	MOVE ERR(3) EMESS
00378	C	MOVE ERR(1) EMESS
00383	C	MOVE ERR(1) EMESS
00388	C	MOVE ERR(1) EMESS
00400	C	MOVE *BLANKS EMESS
00422	C	MOVE ERR(5) EMESS
00426	C	MOVE ERR(6) EMESS
00436	C	MOVE ERR(7) EMESS
00441	C	MOVE ERR(8) EMESS
00454	C	MOVE ERR(9) EMESS
00459	C	MOVE ERR(10) EMESS
00467	C	MOVE ERR(4) EMESS

ALL found 18 occurrences.

- Another method for selecting a field name supported by most Windows applications is to use the **Shift** key together with the Cursor control keys to select text. Place your cursor anywhere on a source line. Now press the **Shift** key and move the cursor to the right using

the right arrow key. If you are not already familiar with this method of text selection experiment with the other cursor keys and with **Home** and **End**. The scope of a specific key is enlarged by also holding down the **Ctrl** key at the same time – try it and see. Many people who find the “drag to select” method tough to get used to find this keyboard driven approach far easier to use.

**A word of caution.** When working in a filtered view, be very careful when deleting a group of lines, for example by using “DD” commands in the sequence number area. All lines in the block will be deleted – not just the visible ones!! Sometimes it is really useful to have an UnDo feature!

## Using the Show Feature

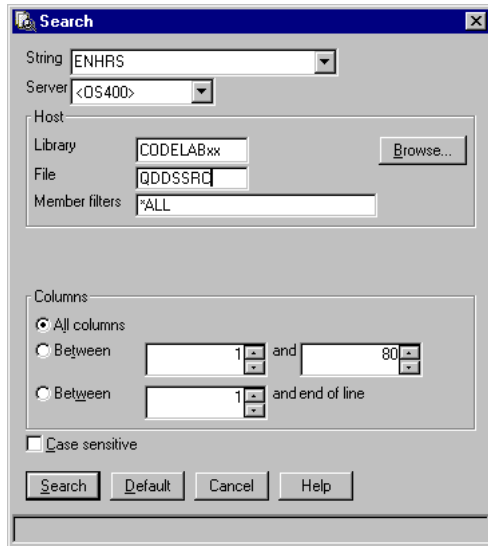
To help you navigate quickly through your ILE RPG source the editor allows you to filter lines based on the line type. In this example, we want to see where all the subroutines are defined in your source:

- From the View menu, select Show and then select Subroutines. All BEGSR and ENDSR subroutine specifications are displayed allowing you to move quickly and easily to the area in your file where the desired subroutine is located.
- Move your cursor to the line that begins the subroutine CHGCDE. (line 433)
- Use the + sign alongside the line containing **BEGSR CHGCDE** to expand the logic of the subroutine. Then position your cursor on any line within the subroutine.
- Next show all lines by pressing **Ctrl+A**. Notice that your cursor is still positioned on the line where you left it.
- Another option available in RPG is to show all Control statements, that is those that effect the flow of control within the program such as IF, DOW, etc. Try this option to get yet another view of your program. As before you will be able to click on the + (plus) signs to “open up” a section of the source. Press **Ctrl+A** when you are finished to get the full source view back.

## Multi-File Search Utility

If you would like to search through the members in a source physical file or through the files in a local directory, you can use the CODE Multi-File Search tool.

- From the editor **Windows** menu, select **Multi-file search**. The Search dialog shown here

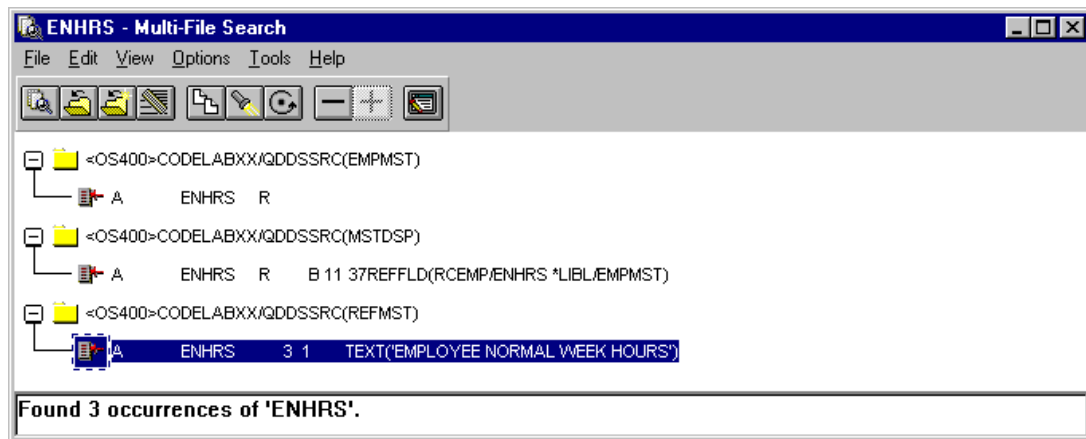


will appear.

- In the String entry field, type ENHRS
- Click on the Server combo-box and select <OS400>.
- In the **Library** entry field, type **CODELABxx** where xx is your team number.
- In the **File** entry field, type **QDDSSRC** to search all members in this source physical file.

## INTRODUCTION TO CODE

- Press the **Search** button. The Multi-File Search window lists all the lines in all the files that reference ENHRS.



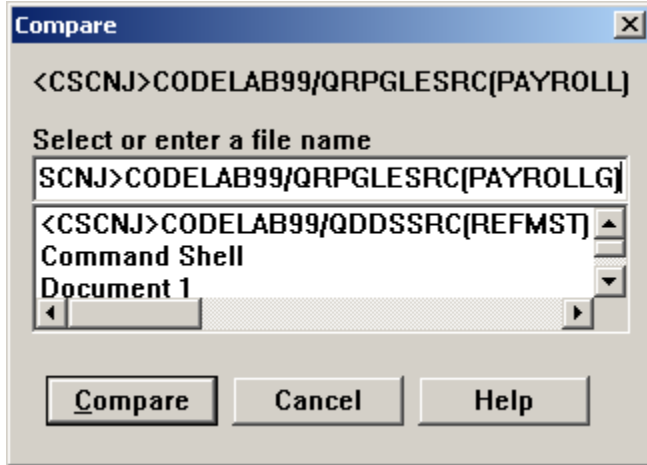
- Double-click on the last line in the list:  
`ENHRS 3 1 TEXT('EMPLOYEE NORMAL WEEK HOURS')`  
The member REFMST should be automatically loaded into the editor and the cursor placed on the correct line.
- If you are interested in knowing what went on “behind the scenes” to achieve the multi-file search, press **F9** to go to the command shell and you will see that **FNDSTRPDM** was the actual command used by CODE to effect the search. Don’t forget to make your source window active again before proceeding – you should know how to do that by now.

## Comparing Files

If your code undergoes many changes, you will find the Compare utility useful. It allows you to compare different versions of a program and find the differences. You can edit your source directly in the utility -- it contains all of the CODE Editor's features.

- **Switch** back to `<OS400>CODELABxx/QRPGLESRC(PAYROLL)` using the blue drop-down list beneath the toolbar (or any of the other methods to switch files that we discussed earlier.)
- From the **Actions** menu, select **Compare...** The **Compare** dialog appears.

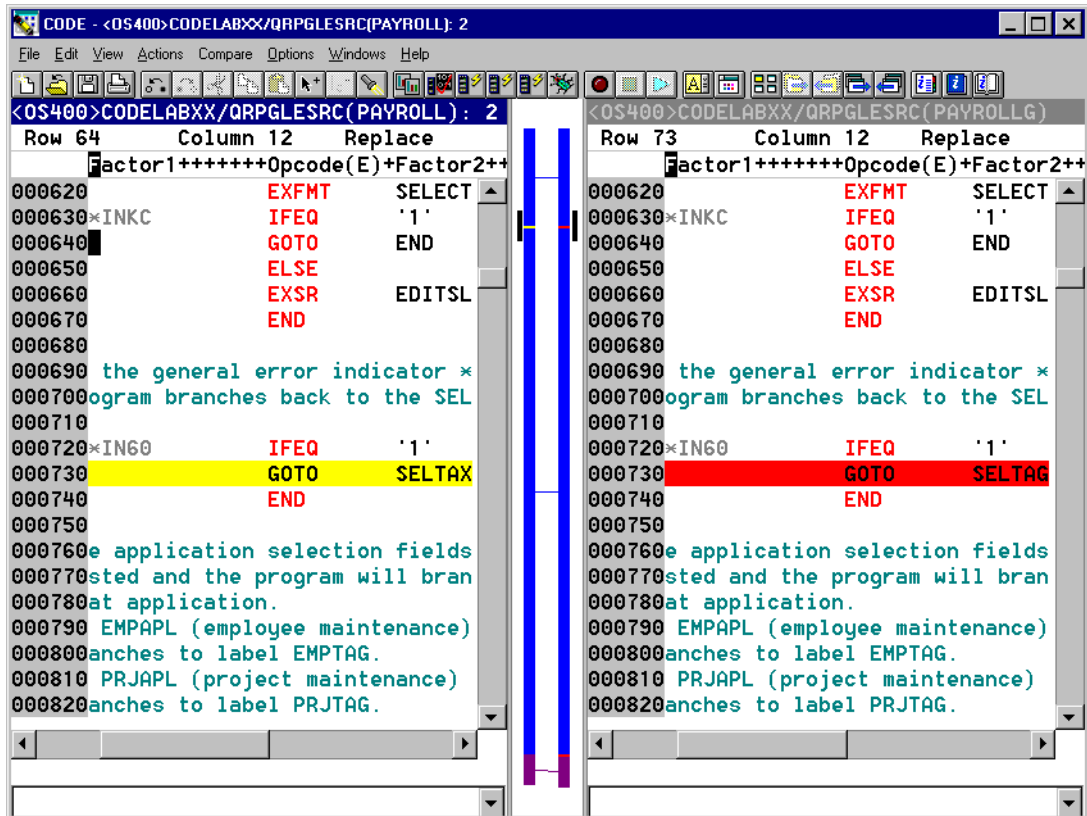
## INTRODUCTION TO CODE



- In the entry field, type  
    <OS400>CODELABxx/QRPGLESRC(PAYROLLG)  
    where 'OS400' is the name of your server and xx is your team number. PAYROLLG is a version of the PAYROLL source file in which we have corrected some coding errors.

Note that if the PAYROLLG file was already loaded in the editor, it would have appeared in the pull-down list in the dialog box. You can see the file REFMST in the example. Selecting a member from the list has the same effect as typing it in.

- Click on the **Compare** push button. The editor now has the PAYROLL member loaded on the left and member PAYROLLG loaded on the right. In between the two members are two long vertical blue lines with horizontal yellow and red bars highlighting the



## INTRODUCTION TO CODE

differences in these members.

- Use the vertical scroll bars to move within the files. As you scroll, you will see where the differences are in the members. RPG experts will notice that PAYROLL has some errors in it. We will fix these in a few moments.
- Click on the **Compare** menu (it was inserted when we started the compare action), and check out some of the options. For example - when comparing two files it can be useful to “lock” one of them, perhaps to prevent accidental changes to a “Master” version of the source file. Try selecting one of the windows and then locking it. You should only be able to make changes to the unlocked window.

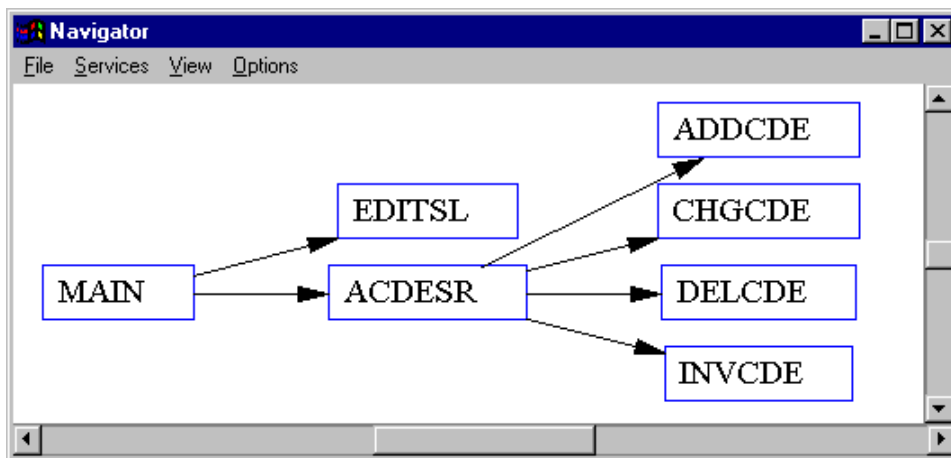
You might also like to check out some of the options for ignoring spaces. This can be very useful with languages like C or Java (or even /Free-form RPG) where the code may be identical except that some “prettification” has been done to increase readability. By ignoring leading spaces, such changes have no effect on the comparison.

- When you have finished, select **Exit Compare** to go back to the original view.

## The Navigator

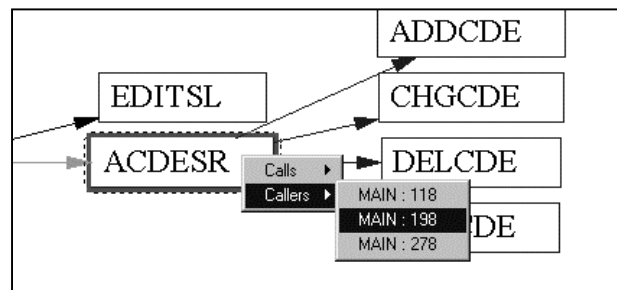
The Navigator allows you to view the relationships between calling and called procedures or subroutines. The arrow direction indicates which ones are calling and which ones are being called. The Navigator is available for ILE RPG, C, and COBOL languages.

- Switch back to editing the PAYROLL file, if necessary.
- From the View menu, select Navigator menu and then select Open. A Navigator window opens up showing that the MAIN procedure calls EDITSL and ACDESR. The subroutine ACDESR calls ADDCDE, CHGCDE, DELCDE, and INVCDE. If the default horizontal view does not work well for a particular source, you can switch the orientation by using **Options – Orientation – Vertical**.



Note that the latest versions of CODE include in this view an additional box representing the main source file itself. It would appear to the left of the block **MAIN** in this picture.

- If you right click on the ACDESR subroutine you can use the pop-up menu go directly to any line from which that subroutine is called or to any of the other subroutines that it calls.



- From the Services menu, select Topology. This opens a file called NAVIG.OUT that lists the called and calling functions. For large, complex programs these tools are invaluable. When you have studied the list press **F3** to close NAVIG.OUT.

## INTRODUCTION TO CODE

- If you wish you can print any of the views from the **File** menu.
- Select the Navigator window and from the File menu, select Close.

## Syntax Checking

One of the powerful features that the CODE Editor shares with SEU is its ability to syntax check your source. Syntax checking can be done either as the cursor leaves each line of source or all at once on either the currently selected source or on the entire file. In this part of the exercise you will deliberately create a syntax error and will be immediately prompted to correct it.

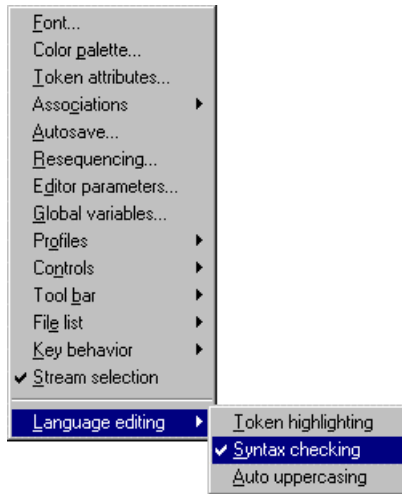
- Move** the cursor to the row which contains 'EXSR ACDESR' (approx line 198). You might already be on that line but if not, you already know different ways of getting there. You could type the line number in the sequence number column, or you could use **Locate Line** (Ctrl+L), **Find and Replace** (Ctrl+F), **Filter** (Ctrl+I) or the **Navigator** function.
- Append** an X to the EXSR op-code to make it EXSRX.

001960	C	*INKC	IFEQ	'0'	
001970	C	EMPNO	CHAIN	EMPMST	50
001980	C		EXSRX	ACDESR	
001980	RNF5014E Operation entry is not valid; specification is ignored.				
001990	C		ELSE		
002000	C		GOTO	END	
002010	C		END		

- Move** the cursor off of the line. An error message appears to draw attention to the error.
- Move** the cursor onto the pink error message.
- Press F1**. This opens a window with second level help for the error.
- Minimize** the help window.
- Change **EXSRX** to **EXSR** to correct the error (or you could use the Undo function!)
- Move** the cursor off the line you just fixed. The error message is automatically removed from the editor. Just as SEU performs syntax checking whenever a control key is pressed, CODE performs the check as you leave the line.

## INTRODUCTION TO CODE

- Sometimes you want to just do some heads-down coding and not have to worry about syntax error. To facilitate this CODE allows you to toggle automatic syntax checking on or off by using the **Options - Language editing - Syntax checking** menu item. You can




perform a full syntax check at any time by selecting **Syntax check file** from the **Actions** menu at any time.

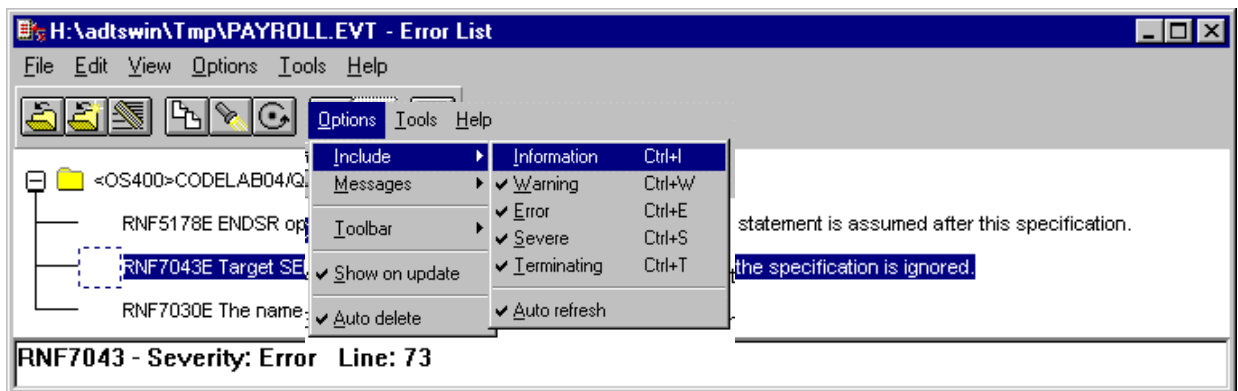
## Verifying Your Source

Now we get to play with one of the most powerful features of the CODE Editor -- the Program Verifier. The verifier checks for semantic (compile) errors on your workstation so that you can guarantee a clean compile on the host. Think of the time you will save not waiting for a batch compile – not to mention the host cycles you’ll save. It is especially handy when you are writing code while you are disconnected from the host. You can do this because CODE ported the parsing and checking code from the AS/400 host compilers to the workstation. An Error List window identifies the errors that are found and their severity, and the error messages are inserted directly into the source to help you to identify the errors. As you will see the Error list also provides a simple mechanism to locate errors in the source.

### Invoking the Verifier

Before you compile your code on the host, you can make certain that there are no errors by invoking the Program Verifier: Note that the first time you invoke the Verifier there will be a delay. Loading the “compiler” causes part of this delay. The other factor is that the verifier has to communicate with the host to retrieve the external file descriptions. We will show you later how you can “cache” these descriptions on the PC., this makes the verifier really fly!

- Ensure that the PAYROLL source file is the active edit window. Then from the **Actions** menu, select **Verify program** and then select **No prompt** (or click on ). A verify is performed and the **Error List** appears. You may see several **informational** messages that are not shown in the picture below.



- If so, you can find it useful to filter out all the informational messages. To do this, toggle the **Options – Include - Information** menu item.
- To determine the severity and line number of an error, **Click** on any error in the list. The status line at the bottom displays the severity of the error and the line number in the source.

## INTRODUCTION TO CODE

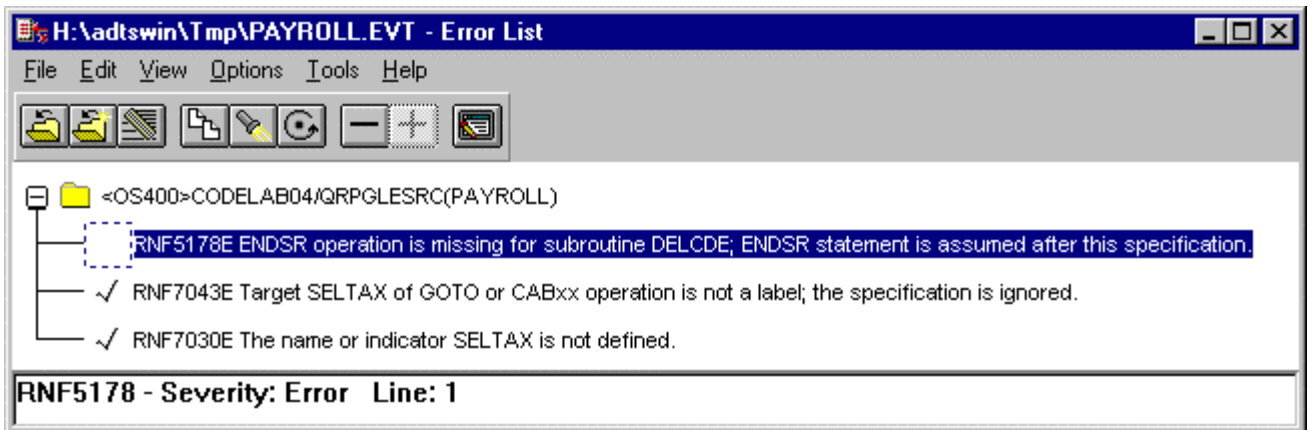
- Press **F1**. Second-level help for that error message appears. Read the help.

Minimize the Help window.

### Fixing Errors

Now you will use the editor and the error list together to fix the errors found when verifying the program.

- To simply locate the source line in error (or at least the one the compiler is complaining about) you simply have to double click on the error message. **Double-click** on the error **RNF7043E**. You will see that the error text is inserted into the source and you are taken to the offending line.
- The error on line **73** is a typing mistake. Think about how you might use CODE's view filtering options to help find the correct target name. Hint: You could filter for 'TAG' or perhaps select the first part of the name (say 'SELT' and the filter on the selection. Once you have discovered the correct name, make the appropriate change.
- Switch back to the Error List window. To switch back, use **Ctrl-E** or the **Windows** pull down option **Error List**. Notice that some of the errors now have a checkmark beside them. This indicates that you have modified the lines related to these errors. You may not have fixed the problem, but at least you've tried! Another remaining error is **RNF5178E**. This error is caused by a missing **ENDSR**.



- To find where to insert the necessary **ENDSR** line, try using **Show > Subroutines**.
- Position your cursor on the line just **BEFORE** where you need to insert the line for **ENDSR**. Press **<Enter>** to insert a new line and add the code to fix the problem. If you are unfamiliar with RPG ask your instructor for help.

(**Hint:** use the Tab key to quickly move to the appropriate column)


## INTRODUCTION TO CODE

If you have any non-informational errors remaining in your error list, fix them now. Ask your instructor for help if you need it.

It is always a good idea to verify your code again after correcting errors and before submitting the code for a compile. We will do that in a moment, but first, we want to make sure we get our changes to the source member saved back to the OS/400 source file.

### **Saving the Source Member & Re-Verifying the Code with Cache**

Before you lose any of your changes, it's a good idea to save them. You may have noticed that CODE has been auto-saving the files for you. You should be aware though that the save is to a numbered file on the PC. The file is not saved back to the 400 unless you specifically do so.

- You can save the member from either the **File** menu, the toolbar (click on ) , or: Press **Ctrl+S**. The modified source files are uploaded to the AS/400.
- Once you have saved the member, verify the program again. This time select the “Prompt” option. Notice that you can have the verifier produce a full compiler listing, with all the options you might expect for cross reference listings etc. Try this if you are interested.
- Next select the “Settings” tab and select the “Use Cache” check-box.
- Now press the “Submit” button to start the verifier. You will not notice the benefit of the caching this time, since the verifier still has to download the OS/400 file descriptions.
- To see the benefits of caching, immediately perform another verification, again **WITHOUT** prompting. CODE will remember the cache option you set above. Notice how much faster the verify process is. Most CODE users find it useful to cache external file descriptions even when they are connected to their OS/400 systems because of the greatly increased speed of verifying.

Note, however, that another good reason to use CODE's caching feature is for working disconnected from your system. After caching your external file descriptions, you can save your source to your local disk, disconnect from your OS/400 system. You can then continue to edit the source file and even verify your changes without a system connection. This is a telecommuter's dream!

If necessary, you can copy your source files and your cached file subdirectory to diskette. You can then take them to another workstation which has CODE installed, and continue to edit and verify your work disconnected from your OS/400 system.

## **CODE Program Generator**

“Program Generator” is the rather grand name given to the tool within CODE that provides a workstation interface to create programs on the host. It gives you easy access to all the compile options available for all the supported CRTxxx commands. It remembers your compile options both generically and on a program-by-program basis if needed, so that they only need to be entered once.

The Program Generator now also supports several new features for Java, such as compiling on an OS/400 host as well as running your Java program remotely.

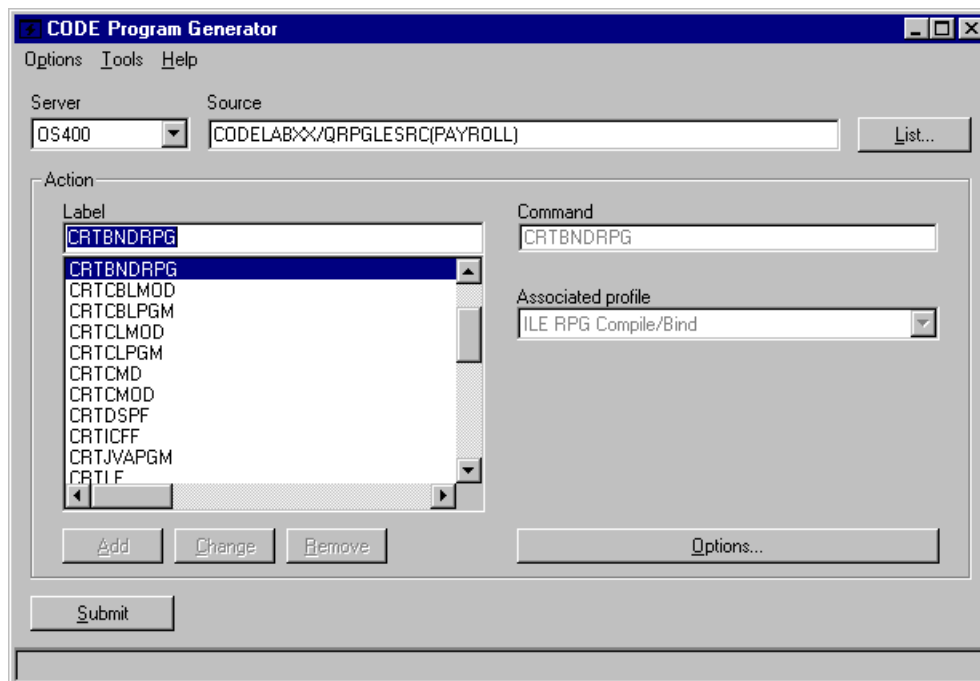
If you’ve used the local program verifiers then these compiles should be successful -- no wasted host cycles. However, if there are errors, the host compiler will send the error information back to the workstation and they will be loaded into the Error List window, just as it did when you did a program verify.

There are two major causes of such errors. The first is that the Library list was different for the compile than the one used by the verifier. The second arises from the fact that the Verifier always uses the latest version of the compiler. If you subsequently compile specifying a Target Release other than \*CURRENT, then the compile will fail if you have inadvertently used features not available in the release you are targeting. In this regard of course, the Verifier is very similar to SEU, which also has no “release awareness”, so it should not cause you too many problems.

# Starting the Program Generator

Whenever you request that source be compiled from within the editor, the Program Generator is started.

- ❑ From the editor's **Actions** menu, select **Compile** and then select **Prompt...** The Program Generator window appears. If you have made source changes to any of the files currently

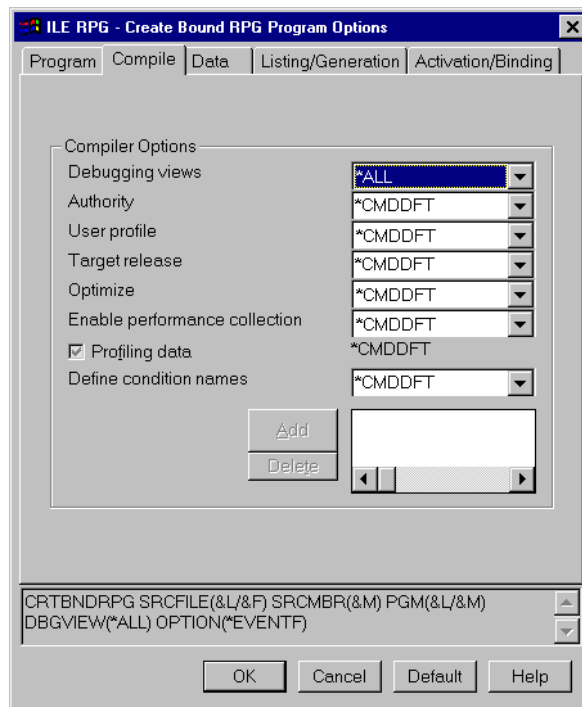


loaded in the editor, you will be asked if you wish to save the changes. Deselect any files in the list that you do not wish to save at this time before pressing the **OK** button.

- ❑ We will now use the Program Generator to select compile options. To start, we have to specify some settings and the program we want to compile.
- ❑ Using the **Options pull-down menu** (not to be confused with the **Options...** button!), select **Settings...** The **Settings** dialog appears. We will use the **Options...** button a little later.
- ❑ On the **General** page **click** on the **Interactive** action mode and make sure the **Notify on completion** is checked. Although this does temporarily tie up your server, we have found it to be a more reliable method of ensuring that the compiler feedback is processed correctly.
- ❑ **Click** on the **OK** push button to save your settings and dismiss the dialog.

## INTRODUCTION TO CODE

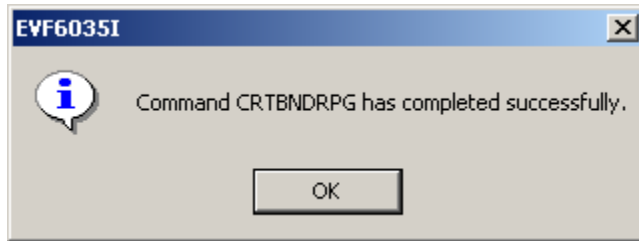
- Both the **Server** and the **Source** entry fields should already be filled in with **OS400** and **CODELABxx/QRPGLESRC(PAYROLL)**, respectively.
- The **Label** identifies the command and options for the command that will be used to compile the program. **Select the CRTBNDRPG label**. Note: You may find that CRTBNDRPG is NOT the automatically selected label. You may have CRTRPGMOD selected by default. This default selection can be changed for ILE languages. But for now, if CRTBNDRPG was not automatically selected for you, just page through the list of labels until you find it and select it.
- Click on the Options...** push button near the bottom right-hand side of the notebook to select the compile options for the program.
- Click on the Compile** tab.
- Change the **Debugging views** from **\*CMDDFT** to **\*ALL**. Select some of the other tabs to explore the compile options available.
- Notice the full command string that is generated by your selections appears near the bottom of the window. Don't be surprised if it looks like a PDM user-defined option!



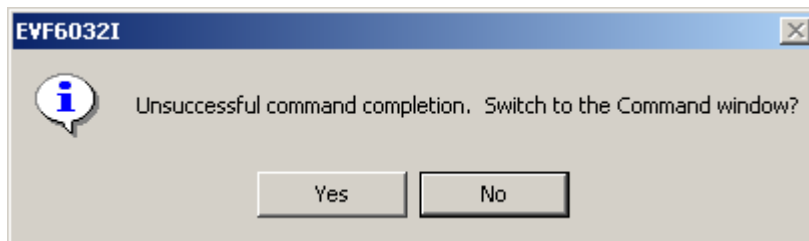
- Click on the **OK** push button when you are done.
- Now click on the **Submit** push button to submit your compile request. Note that your compile dialog box does not go away on its own. You may close this window at any time you want. Most CODE users wait until the compile completes.

## INTRODUCTION TO CODE

- A message will tell you when the compile is complete. Click on the **OK** push button in the message dialog.



- If the compile should fail for any reason the following dialog will appear. Click on the **No** option. We have no idea why the CODE designers think you might want to go to the command shell. There is no useful information to be found there at this point. We will however introduce you to the Command Shell shortly.



- Now click on the **X** in the upper right-hand corner of the Program Generator notebook to close it.

## The Command Shell

You can use the **Command Shell** inside the CODE Editor to submit OS/400 commands to any of your connected servers. For example, you could change your library list.

- From the **Windows** menu, select **Command Shell** (or simply press **F9**). This will bring up the Command Shell where you can enter both workstation and host commands.
- At the **<OS400>** prompt, type **ADDLIBLE QTEMP** Wait for the response from the system. You should see a message something like:

**Library QTEMP already in the library list.**

## INTRODUCTION TO CODE

- You could also try adding one of your work libraries to your library list. Remember that you can only issue commands that do NOT require a 5250 screen unless you are using a STRCODE server. For example, you can do ADDLIB, but you cannot do EDTLIB from an auto start server session. If, on the other hand, you are using a STRCODE server session, then you can issue any OS/400 commands. You may need to click on your 5250 window to see the results in some cases.
- From the **Options** menu, select **Servers** and then select **LOCAL**. This will switch the prompt in the Command Shell to a workstation prompt.
- Type **C:** and press **Enter**.
- Type **DIR** and press **Enter**. Wow! Now you can even scroll up and down to view all files and directories in the c drive.
- From the **Options** menu, select **Servers** and then select **OS400** (or your server name) to switch back to host prompt.

## Running the PAYROLL Program

**Note:** You can only perform this part of the Lab as instructed below if you are using an STRCODE server. It will not work using an Auto-start server since it requires the use of the 5250 session. However, you can still run the program from a regular **5250 session** in this case. Just start a 5250 session and call the payroll program from your CODELABxx library.

In this part of the exercise, we will run the PAYROLL program that we just compiled.

- In the editor, switch to the Command Shell by pressing F9.
- At the <OS400> prompt, type in the following command:  
    CALL PAYROLL  
    then press <Enter>.
- Switch to your 5250-emulation session if it is not automatically selected for you.
- Place an 'x' beside Employee Master Maintenance. Press <Enter>.

```
PRG01                               Time Reporting System          98/04/09
                                   Maintenance Selection          17:26:38

                                   Enter an X beside the application you want to maintain

                                   - Employee Master Maintenance
                                   - Project Master Maintenance
                                   - Reason Code Master Maintenance
```

- Type 123 for the Employee Number.
- Type A for the Action Code to add employee 123. Press <Enter>.
- Type any information you like about the employee. Press <Enter>. Play in the application as much as you like.
- Press F3 to end the PAYROLL program. Notice that when the program ends, the display returns to showing the server details. A common mistake, when using this facility, is to use the command shell to run a program (possibly PDM or some other utility) and forget to exit back to the server screen. The result of this is that subsequent attempts to use the server will be met with a “busy” message. If you ever receive a “server busy” message this should be the first thing to check.