

# Getting Started



**Micro Focus**  
The Lawn  
22-30 Old Bath Road  
Newbury, Berkshire RG14 1QN  
UK  
<http://www.microfocus.com>

Copyright © Micro Focus IP Development Limited . All rights reserved.

**MICRO FOCUS**, the Micro Focus logo and are trademarks or registered trademarks of Micro Focus IP Development Limited or its subsidiaries or affiliated companies in the United States, United Kingdom and other countries.

All other marks are the property of their respective owners.

# Contents

<b>Getting Started</b> .....	<b>4</b>
Features .....	4
Introduction .....	4
Develop and Build within Visual Studio .....	4
Build Using Server Express .....	5
Test on Server Enterprise Edition .....	5
Modernization and Integration .....	5
Tutorial: Developing a Simple COBOL Application .....	5
Overview .....	5
Starting Visual Studio .....	6
Creating a Solution and a Project .....	7
Add Files to the Project .....	7
Build and Run the Project .....	8
Project Properties .....	8
Debug the COBOL Project .....	8
Tutorial: Developing .NET Managed COBOL .....	9
Tutorial: JCL .....	15
Overview .....	15
Configure an Enterprise Server .....	15
Create and Build a JCL Application .....	18
Submit and Run a JCL Job .....	19
View the Spool Queues .....	19
View the Catalog .....	20
Debug Dynamically Under Enterprise Server .....	20
Debug Remotely Under Enterprise Server .....	21
More Information .....	22
Tutorial: Compiling, Linking and Debugging Open PL/I Programs Running Under ES/JCL .....	22
Overview .....	22
Prerequisites .....	22
Configure an Enterprise Server to Run Open PL/I Programs Under JCL .....	22
Compile and Link Open PL/I Programs Under JCL .....	24
Debug Open PL/I Programs Under JCL .....	24

# Getting Started

This book shows you how to start using Studio Enterprise Edition to develop COBOL applications. The tutorials introduce the major features of Studio Enterprise Edition within Microsoft Visual Studio.

## Features

### Introduction

Studio Enterprise Edition provides a tool suite that enables you to migrate business applications to lower cost environments. You can also use the product to maintain, improve and modernize the migrated application.

Studio Enterprise Edition supports many mainframe technologies, such as CICS, IMS, JCL, and DB2 on Linux, UNIX, and Windows. This means that the core online and batch application logic can be migrated and reused with minimal modification on any of these new environments.

The product also includes the development and testing tools you need to modernize the migrated applications, to exploit existing interfaces and extend them to a Service Oriented Architecture (SOA), through .NET, Web services or J2EE.

The Studio Enterprise Edition suite supports the entire development life cycle from analysis to test, while its companion product, Server Enterprise Edition, supports the deployment of the migrated application on the new production platform.

### Develop and Build within Visual Studio

You use Microsoft Visual Studio and its integrated development environment (IDE) to edit, compile and debug the COBOL applications. The IDE provides all the functionality to manage projects and debug applications.

You can develop applications that have been migrated from the mainframe and that use mainframe technologies like CICS, JCL, and IMS. These compile to native (Micro Focus) COBOL code, as `.exe` or `.dll`, and execute in conjunction with the run-time system.

You can also develop COBOL applications as .NET managed code. .NET managed COBOL programs compile to Microsoft Intermediate Language (MSIL), and they execute in conjunction with the Microsoft Common Language run-time system (CLR), just like programs in other .NET languages, such as C#.

Development support is provided for:

- Projects — with templates supplied for mainframe applications, console applications, class libraries and more
- Editing — with syntax colorization, background parsing and, for .NET managed code Intellisense
- Debugging code — with the usual features such as stepping, breakpoints, watch windows and querying execution environments
- Mainframe applications — for editing, compiling and debugging CICS programs, JCL applications and IMS applications

## Build Using Server Express

When the target deployment environment is Linux or UNIX, you develop the application under Windows. When the application is working you transfer the project artifacts to the Linux or UNIX server for re-compilation and building with Server Express.

Server Express includes a native code generator that produces optimized native code specific to the target Linux or UNIX platform. Also included is a complete set of character-based tools for debugging and managing diagnostics directly on Linux or UNIX.

## Test on Server Enterprise Edition

The Server Enterprise Edition run-time environment is included within the development product to enable you to perform extensive unit testing. It provides a local run-time environment for the migrated application that supports the execution of COBOL/CICS/IMS applications, JCL procedures and the COBOL programs or utilities they call.

You can use the monitoring and administration tools to start and stop instances of the Server Enterprise Edition environment and to administer the local configuration. These tools enable you to interact with a running Server Enterprise Edition environment in a similar way to a mainframe console. They also provide real-time reporting on the current status of any instance of Server Enterprise Edition for debugging and diagnostics purposes.

## Modernization and Integration

### Terminal Emulator User Interfaces and Modernization

Studio Enterprise Edition includes the Micro Focus RUMBA terminal emulator to enable connectivity from Windows desktops to virtually any host system. This includes connectivity to migrated 3270 based applications executing within Server Enterprise Edition.

If the target environment is Linux or UNIX, RUMBA also provides a VT100 connection to enable you to connect from Windows to the LINUX or UNIX environment without the need to install any other third party software on their Windows desktops.

### Integration with Other Systems

Studio Enterprise Edition and Server Enterprise Edition support CICS Inter-System Communication (ISC). This enables you to develop and test the appropriate interfaces to integrate CICS transactions running off the mainframe with other CICS transactions running on the mainframe.

Server Enterprise Edition includes technology that can be deployed on the mainframe to handle the conversion between TCP/IP protocol stacks on Linux, UNIX or Windows and the SNA/LU6.2 protocol stack on the mainframe.

## Tutorial: Developing a Simple COBOL Application

### Overview

This tutorial introduces the Visual Studio IDE, and shows how to create, build and debug a simple console application using supplied COBOL programs that use ACCEPT/DISPLAY statements.

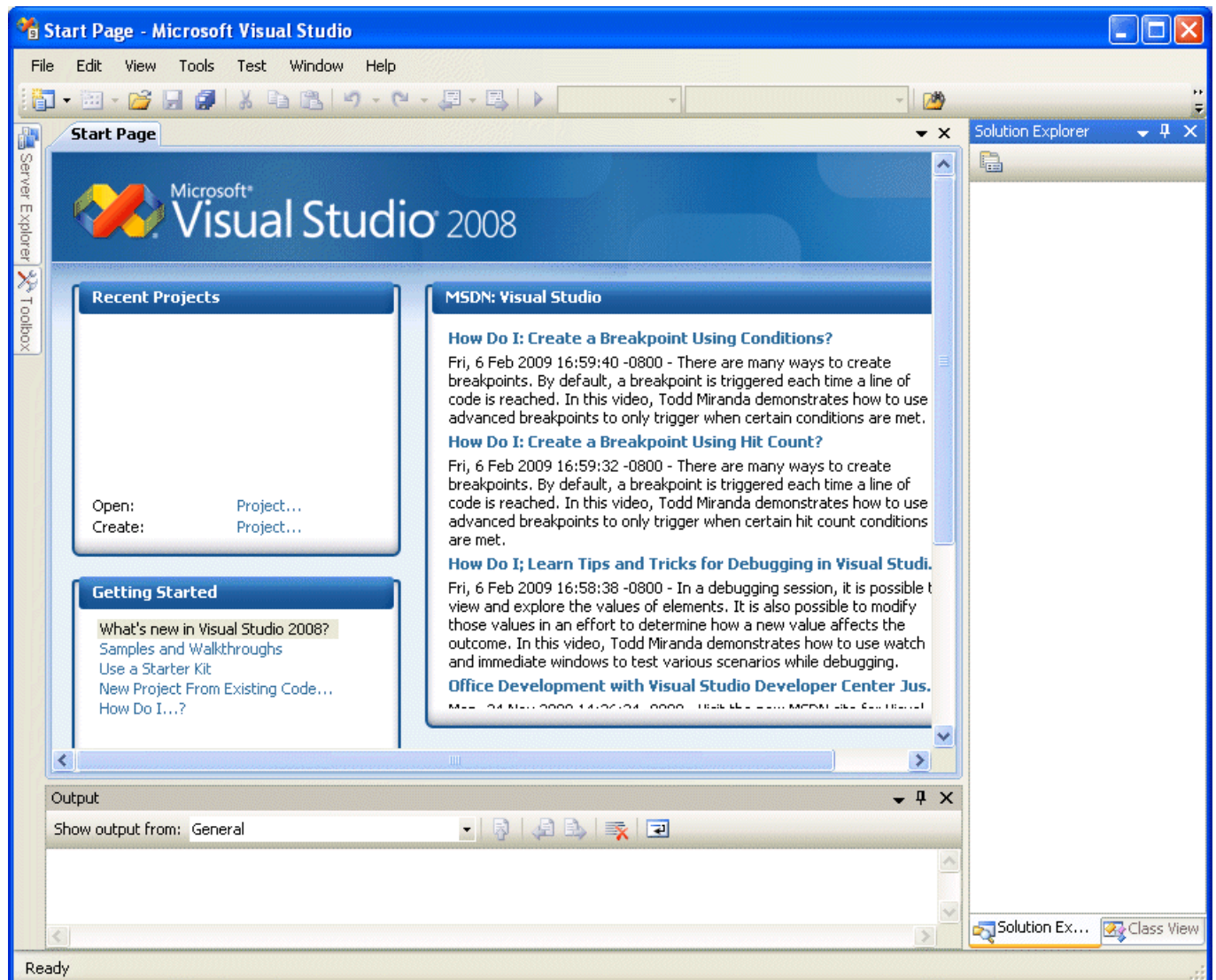
# Starting Visual Studio

To run this tutorial, you must have both Microsoft Visual Studio and Studio Enterprise Edition installed.

To start Visual Studio:

1. On the Start menu, click **All Programs > Micro Focus Studio Enterprise Edition x.x > Microsoft Visual Studio xxxx**.
2. You might be prompted to set the environment, depending on the version installed. If you are prompted, choose **General development environment**. In this setup Visual Studio is customized for work with COBOL projects.

The IDE is shown in the figure below, although the information displayed in the large pane will be different.



The large pane is where your solutions and projects will be opened. At the moment, this shows the Start Page, with up-to-date information on Visual Studio. When you start work, this pane is where your code is displayed.

The pane at the bottom is the Output window, where messages from the IDE and Compiler are displayed.

The right-hand pane is the Solution Explorer, which displays the structure of your solution and projects. At the bottom of the Solution Explorer pane are some tabs. Solution Explorer appears by default.

If any of these panes is hidden, you can show it from the **View** menu.

## Creating a Solution and a Project

The first task is to create a solution and a project. A solution is a container holding one or more projects that work together to create an application. The solution has the extension `.sln`, and is a readable text file. Microsoft recommends you do not edit the file outside of Visual Studio.

A COBOL project has the extension `.cblproj`, and again this is a readable file, but Microsoft recommends that you do not edit it. Different types of project have different extensions, so for example a C# project has the extension `.csproj`.

In this section, you create a project and solution, as follows:

1. In Visual Studio, create a new project by clicking **File > New > Project**.
2. In the **Project Types** pane, expand **COBOL Projects** and click **Native**.
3. Select the **Console Application** template.
4. In the **Name** field, specify **Locking** as the name of the project.

Notice that the **Name** and **New Solution Name** fields have the same name. Changing either one of them automatically changes the other.

5. In the **Location** field, browse to a directory where you want to put this tutorial.

For example, if you create a folder on your `c:` drive called `tutorials`, change the location field to `c:\tutorials`. The solution will be stored in a subdirectory `Locking`, according to the project name.

6. Click **OK**.

This creates a solution and a project. The Solution Explorer shows the `Locking` project. It contains:

- `Program1.cbl`, which is a newly created skeleton program. You can ignore this, because the tutorial uses existing COBOL files.
- `Properties`, which displays the project's properties when you double-click it.

## Add Files to the Project

You can add existing files to the project, as follows:

1. Right-click the project in the Solution Explorer.
2. Click **Add>Existing Item**.
3. Browse to the `Examples\Visual Studio Integration\Tutorials\Locking` folder in the installation of your development system. By default this is `%ProgramFiles%\Micro Focus\Studio Enterprise Edition x.x\Examples\Visual Studio Integration\Tutorials\Locking`.
4. Select all `.cbl` files.
5. Click **Add**.

Alternatively, you can add folders and source files to your project in Solution Explorer using drag and drop. You can drag files and folders from other projects in Solution Explorer or from outside Visual Studio from Windows Explorer. You can use drag and drop with the following keys:

- Hold **Ctrl** to create copies of the files and folders in your project directory
- Hold **Shift** to move the files and folders to your project directory
- Hold **Ctrl+Shift** to link to files or folders in a different location without copying them

# Build and Run the Project

There are two default build configurations for each project type: Debug and Release. These configurations define how to build the project for the different situations.


To build the project and run it:

1. Check the build configuration in use, by clicking **Build > Configuration Manager**.
2. In **Active solution configuration**, choose **Debug** and click **Close**.

Notice in the Standard toolbar at the top of the IDE, that **Debug** shows as the active configuration.

3. Build the project, by clicking **Build > Build Solution**.

The Build option builds only those files that have changed since they were last built, whereas the Rebuild option builds all the files in the project, regardless of whether they have changed since they were last built.

4. Check that the project compiled successfully by looking in the Output view. To display this view, click **View > Output**.
5. Run the application, by clicking **Debug > Start Without Debugging** or .

A console window opens, showing this character-based application running.

6. Go to the console by clicking it. Then type 5 and press **Enter**.

"5" is the code that tells this application to finish and the console closes.

# Project Properties

Project properties define characteristics of the project and enable you to control a project's behavior among other things. Different properties are set by default for different types of projects.

1. To see a summary of the project properties, select the project or solution in the Solution Explorer. By default, the Properties window is displayed below the Solution Explorer.

If the Properties window is not visible, you can show it by clicking **View (> Other Windows) > Properties Window**.

A description of the selected property is displayed dynamically at the bottom of the Properties window. To turn this description on and off, right-click in the description.

2. To display the full project properties, right-click the project in Solution Explorer and click **Properties**.
3. Go to the **COBOL** page of the properties, to display the selected properties for building the project.

Notice that:

- **Configuration** is set to **Active (Debug)**
- **Managed code** is set to **No**, because the current application is being built as native COBOL
- **Generate debugging information** is checked
- **COBOL dialect** is set to **non-mainframe**
- **Output path** shows where the build files will be put

4. Go to the **Debug** page and notice that **Start project** is checked, so that the application starts debugging in the current project.
5. Browse the other properties pages to see what is available and what is set by default for a console application.

# Debug the COBOL Project


The IDE provides debugging facilities, such as stepping, examining data item values and setting breakpoints. To see some of these facilities:

1. Click **Debug > Step Into**, or .




A console opens for running the built programs.

The source code for `locking.cbl` opens in the COBOL editor. The first statement `DISPLAY SPACE` is highlighted in yellow, with a yellow arrow pointing to it. This is the statement to be executed next.

2. Execute the first statement, by clicking **Debug > Step Into**,  or pressing **F11**.
3. Step the next statement, `DISPLAY LOCKING01-00`, in the same way.

This statement displays some information about the application in the console.

4. Execute the whole of the next `PERFORM` statement. To do this, click **Debug > Step Over**,  or pressing **F10**.

Step Over executes all the code for a `PERFORM` or `CALL` statement in a single step.

5. Continue stepping into the code, until you reach the `ACCEPT` statement *accept choice at 2445*.
6. Step into the `ACCEPT` statement.

The console now appears and waits for your input.

7. Type `5`, as before, and press **Enter**.

Notice that the statement "EVALUATE CHOICE" is now highlighted ready for execution and that the value of the choice variable is now "5". You can see this by hovering over the choice variable in the Evaluate statement, and also by looking in the Autos pane at the bottom.

If the Autos pane is not visible, open it by clicking **Debug > Windows > Autos**.

8. To add the choice variable to the Watch list, right-click it and click **Add Watch**.

This opens a separate pane, the Watch pane, which shows the values of watched variables.

9. Continue stepping to the end.

You can close Visual Studio now. You do not need to explicitly save anything, because the files are automatically saved when you build them.

## Tutorial: Developing .NET Managed COBOL

### Overview

This tutorial introduces the Visual Studio IDE, and shows how to create a solution for a Windows application with a Windows form that says "Hello World".

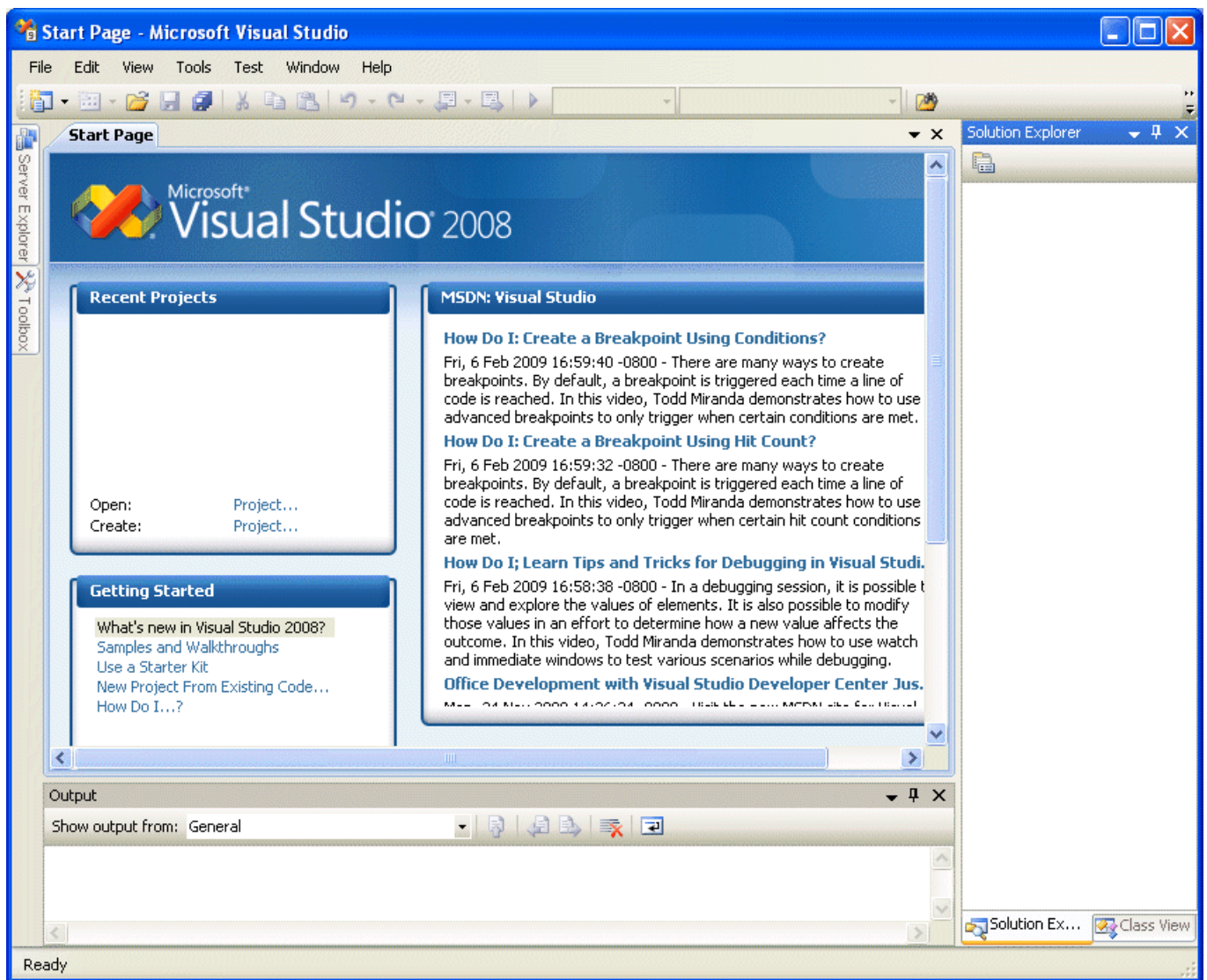
### Starting Visual Studio

To run this tutorial, you must have both Microsoft Visual Studio and Studio Enterprise Edition installed.

To start Visual Studio:

1. On the Start menu, click **All Programs > Micro Focus Studio Enterprise Edition x.x > Microsoft Visual Studio xxxx**.
2. You might be prompted to set the environment, depending on the version installed. If you are prompted, choose **General development environment**. In this setup Visual Studio is customized for work with COBOL projects.

The IDE is shown in the figure below, although the information displayed in the large pane will be different.



The large pane is where your solutions and projects will be opened. At the moment, this shows the Start Page, with up-to-date information on Visual Studio. When you start work, this is the pane where your code is displayed.

The pane at the bottom is the Output window, where messages from the IDE and Compiler are displayed.

The right-hand pane is the Solution Explorer, which displays the structure of your solution and projects. At the bottom of the Solution Explorer pane are some tabs. Solution Explorer appears by default.

If any of these panes is hidden, you can show it from the **View** menu.

### Creating a Solution and a Project

The first task is to create a solution and a project. A solution is a container holding one or more projects that work together to create an application. The solution has the extension `.sln` and is a readable text file. Microsoft recommends that you do not edit the file outside of Visual Studio.

A COBOL project has the extension `.cblproj` and again is human readable but Microsoft recommends that you do not edit it. Different types of project have different extensions, so for example a C# project has the extension `.csproj`.

In this section, you create a project and solution, as follows:

1. Start Visual Studio.

2. Create a new project by clicking **File > New > Project**.
3. In the **Project Types** pane, expand **COBOL Projects** and click **Managed**.
4. In the Templates pane, select **Windows Forms Application**.

The drop-down list above the Templates pane allows you to change the version of the target framework. Select the framework your clients are using to ensure your application includes classes and methods supported by it. You can leave the default .NET Framework for this demonstration.

5. In the **Name** field at the bottom of the window, specify **WinHello** as the name of the project.

Notice that the **Name** and **Solution Name** fields have the same name. Changing either of these automatically changes the other.

6. In the **Location** field, browse to a directory where you want to put this tutorial.

For example, if you create a folder on your `c :` drive called "Tutorials," change the location field to `c : \tutorials`. The solution will be stored in a subdirectory `WinHello`, according to the project name.

7. Click **OK**.


This creates a solution, a project and an empty form, which is opened in the form designer. The Solution Explorer pane shows the `WinHello` project, which contains:

- `Properties`, which contains the full project properties, including the resources for the form and references to some system packages, such as `System.Windows.Forms`, which support the system classes that the project uses.
- `Form1.cbl`, which contains your code for the form. Under the expanded `Form1.cbl` is `Form1.Designer.cbl`, which is created by the form designer. Don't edit this code, because this defines the form, and if you change it you might not be able to open the Design window. Also, any changes you make might be overwritten by the form designer.
- `Main.cbl`, which contains the trigger program that displays the main form.

### Painting a Button and a Label

This section "paints" a button and a label on the page or form.

1. Paint a button as follows:

- Click  or **View > Toolbox**.
- If the **Button** object isn't listed, click **Common Controls** in the Toolbox.
- Point to the **Button** object and drag it onto the page or form.

2. Change the text on the button to "Say Hello", by editing the button's Text property, as follows:

- Click on the button.
- In the **Appearance** section of the Properties pane, scroll to the **Text** property.
- Edit the text to `Say Hello` and press **Enter**.

Notice that the Name property of the button (in the Design section of the Properties pane) is, and remains, "button1."

3. Paint a label somewhere on the form by dragging the **Label** object in the same manner as you did the **Button** object.

4. Clear the default text in the label, by deleting the text in the Text property field and pressing **Enter**.

The label essentially disappears upon this action, but remains on the form, as you will see when you build and run the application. Notice that the name property of the label control is, and remains, `Label1` (or `label1`).

### Adding a Click Event Handler

Now, you add an event handler to display "Hello World" when the button is clicked.

1. Create a click event handler for the button by double-clicking the button.

This generates a method called `button1_Click` in the code.

Look at the code for the `button1_Click` method. It is just a skeleton and does nothing. The procedure division is empty. We will look at the rest of the code when we have run the application.

2. In the procedure division of the `button1_Click` method, insert the following line:

```
move "Hello World!" to self::"label1"::"Text"
```

3. or you could equally write:

```
set self::"label1"::"Text" to "Hello World!"
```

This moves the text "Hello World" into the Text property of the label you painted.

## Building and Running the COBOL Application

There are two default build configurations for each project type: Debug and Release. These configurations define how to build the project for the different situations.


To build the project for debugging and to run it:

The Build menu offers the choice of building and rebuilding the solution, as well as building and rebuilding our project. The Build option builds only those files that have changed since they were last built. Whereas the Rebuild option builds all the files in the project, regardless of whether they have changed since they were last built.

To build and run the project:

1. Check the build configuration in use, by clicking **Build > Configuration Manager**.
2. In **Active solution configuration**, ensure **Debug** is selected and click **Close**.

Notice in the Standard toolbar at the top of the IDE, that Debug shows as the active configuration.

3. Click **Build > Build WinHello**.
4. Check that the project compiled successfully by looking in the Output pane. To display this pane, click **View > Output**.
5. Run the application, by clicking **Debug > Start Without Debugging** or .
6. When the page or form displays, click the **Say Hello** button.

The label should now say "Hello World!"

7. Click the X at the top right to close the page or form.

## Project References

When you create a project, the full project properties contain the references to assemblies containing classes that are used by the project, such as the System assembly and, for our Windows application, System.Windows.Forms. You can see this on the References page of the full project properties.

If you write code that uses other classes in other assemblies, you add the assemblies as references as follows:

1. Right-click the WinHello project in Solution Explorer.
2. Click **Add Reference**, or double-click the Properties folder in Solution Explorer and on the References tab, click **Add**.
3. On the .NET tab in the Add Reference dialog box, you can select .NET Framework assemblies or other .NET assemblies. Scroll to MicroFocus to see the available Micro Focus assemblies.
4. On the COM tab, you can select the COM objects that you have registered on your machine.
5. On the Projects tab, you can select other assemblies that your .NET code refers to. There are currently none available to the project in this tutorial.
6. Click **OK** when you have completed your selections.

For more information, see the Visual Studio Help.

## Project Properties

Project properties define characteristics of the project and enable you to control a project's behavior, among other things.

Brief project properties are by default dynamically displayed in the Properties pane, below the Solution Explorer. Here are some tips on viewing brief project properties:

- To see brief project properties, select the project or solution in the Solution Explorer. The Properties pane is displayed below the Solution Explorer, by default.
- If the Properties pane is not visible, you can show it by clicking **View (> Other Windows) > Properties Window**.
- A description of the selected property is displayed dynamically at the bottom of the Properties pane. To turn this description on and off, right-click in the toolbar of the Properties pane.

Full project properties are available by selecting the project in Solution Explorer and clicking **Project > ProjName Properties**. The full properties that you can specify for your native or managed project can apply to all configurations or only to the selected configuration, such as Debug or Release.

## Assemblies

Each project is built into a code assembly.

Assemblies are documented in the Visual Studio Help, which describes them as "a collection of one or more files that are versioned and deployed as a unit. An assembly is the primary building block of a .NET Framework application. All managed types and resources are contained within an assembly and are marked either as accessible only within the assembly or as accessible from code in other assemblies.

Assemblies also play a key role in security. The code access security system uses information about the assembly to determine the set of permissions that code in the assembly is granted". For the basic concepts on assemblies see the Visual Studio Help.

Assemblies are created automatically when you build projects. You need to create a strong-named assembly if you want to include it in the Global Assembly Cache (GAC), which makes it secure and shareable among all the applications on your machine.

## Examining the Generated Code

The form designer creates broadly-speaking two partial classes for the page or form. One partial class contains the generated code for your application, and you can edit and add to this. The other partial class is in the file `Form1.Designer.cs`. It contains code owned by the designer, and you must not edit it.

The partial class containing the generated code contains, among other things:


- A declaration of the class as a partial class such as the following for a Windows form:

```
class-id. Form1 as "Hello.Form1" is partial
    inherits type "System.Windows.Forms.Form".
```
- The object Working Storage section. This allocates storage for the controls for the form, or more precisely, for object references for the controls.
- The New method, which invokes the `InitializeComponent` method in the designer-owned partial class.
- The `button1_Click` method, which you edited to say "Hello World."



## Examining the Event Handlers in the Design View

When you double-click a control in the Design view, a click event handler is created and a method is inserted in the generated code. You created the `Button1_Click` event handler in this way.

You can rename the event handler and add and delete others.

 **Note:** You must do this in the Design view and not in your application code. If you do it in the code, it might get overwritten when the code is regenerated from the design.

To change the name of the event handler:

1. Display the Design view. One way to do this is to select the page or form in Solution Explorer, and then click **View > Designer** or .
2. Select the **Say Hello** button.
3. In the Properties pane, display the events, by clicking .  
(The events are found in the Action section of this pane.) You can scroll through the events to see what is available.
4. Change the name of the click event handler to `readButton1_Click` then press **Enter**.
5. Display the Code view and notice that our method at the end has changed name to **readButton1\_Click**, and that our procedure division is still intact.

Similarly, if you double-click a control by mistake (say on the label control), you generate an unwanted event handler. You cannot simply delete this event handler in the code view, because some event handler code is also in the design. Instead you need to delete the event handler in the design view. Do this in much the same way as you edited the name of the event handler, but instead delete the name.

### Changing the Startup Form

This section adds another form and uses this as the startup form.

Notice that the form you created was called **Form1**. This form is opened when you run the application. If you rename the form, you also need to change the name of the form that starts the application. Similarly, if you delete this form, you need to specify another form as the startup form.

To demonstrate this, add another form and use this as the startup form, as follows:

1. Create a second form similar to the first, as follows:
  - a. Right-click the project in Solution Explorer.
  - b. Click **Add > New item**.
  - c. Choose **Windows Form** and specify a name for the form, such as "HelloAgain."
  - d. Click **Add**.
  - e. Paint the form however you wish and add an event handler. You could paint a form similar to the Hello World one, but with slightly different text, such as "Hello again."
2. Set the main form to the new form, by editing the lines that define and create the main form in the program `main.cbl`, as follows:

- a. Double-click `Main.cbl` in Solution Explorer.

- b. In the local-storage section, change from:

```
01 mainForm type "WinHello.Form1".
```

to:

```
01 mainForm type "WinHello.HelloAgain".
```

- c. In the procedure division, change from:

```
set mainForm to new "WinHello.Form1"()
```


to:

```
set mainForm to new "WinHello.HelloAgain"()
```

3. Build and run the application.

### Debugging the COBOL Project

The IDE provides debugging facilities, such as stepping, examining data item values and setting breakpoints. To see some of these facilities:

1. On the toolbar, click **Debug > Step Into** , or  .  
If you are prompted to rebuild the project configurations, click **Yes**.
2. You can see a yellow arrow at the left of the window pointing to the first line to be executed. Click **Debug > Continue** (or the green **Continue** arrow on the toolbar).

Close the IDE window now. You do not need to explicitly save anything, because the files are automatically saved when you build them.

## Tutorial: JCL

### Overview

The Server Enterprise Edition execution environment is built on top of the Micro Focus Enterprise Server technology that provides the infrastructure not just for executing COBOL, CICS, IMS, DB2, VSAM and JCL, but also for deploying existing transactions as Web Services for easier re-use within new architectures like .NET, J2EE or SOA.

JCL applications written to run on a z/OS or VSE mainframe can be run with the Server Enterprise Edition environment. You maintain the applications in Visual Studio.

You can maintain your JCL files on the PC using any text editor.

### Demonstration Application

This tutorial uses the demonstration application JCL supplied with your COBOL development system. The JCL application performs four functions:

1. Uses IDCAMS to delete from the catalog the data set created in step 2, if it is already there.
2. Runs a COBOL program that writes a simple data set, which is added to the catalog.
3. Uses IEBGENER to create a temporary data set.
4. Runs a COBOL program that reads through the data set created in step 3 and displays it.

### Outline

In this tutorial you perform the steps typically required in managing, maintaining and running JCL applications. In summary, you:

1. Configure an Enterprise Server to run JCL applications.
2. Create a native COBOL project comprising all the parts of your application, such as the COBOL source and the JCL batch files.
3. Associate the project with your Enterprise Server.
4. Build the COBOL project in your IDE.
5. Submit the JCL file on your Enterprise Server.
6. Run the application in your Enterprise Server.
7. View the spool and the catalog.
8. Debug dynamically.
9. Debug remotely.

## Configure an Enterprise Server

These are the steps to create and configure an Enterprise Server to run JCL applications.

## Configure an Enterprise Server for JCL Support

To enable JCL support in an Enterprise Server, you need to do the following tasks:

1. Create a Mainframe Sub System (MSS)-enabled Enterprise Server.
2. Enable Job Execution System (JES) in this Enterprise Server.
3. Configure locations for programs and data files and so on.

### Create an MSS-enabled Enterprise Server

To open the Enterprise Server Administration page in the IDE:

1. Launch Visual Studio.
2. Open Server Explorer.

If it is hidden, display it by choosing **View > (Other Windows >) Server Explorer**.

3. Right-click **Micro Focus Servers** and choose **Administration** from the context menu.

To create an MSS-enabled Enterprise Server:

1. On the Home page of Enterprise Server Administration, click **Add** at the bottom of the table of servers.
2. On the first Add Server page:
  - a. Type JCLDEMO in the **Server Name** field.
  - b. Make sure 32-bit is selected in the **Working Mode** section.
  - c. Click **Next**.
3. On the second Add Server page:
  - a. Click **Micro Focus Enterprise Server with Mainframe Subsystem Support**.
  - b. Click **Next**.
4. On the third Add Server page:
  - a. Delete the check mark by **Create TN3270 listener**.
  - b. Type `Server for JCL` in the **Description** field (near the bottom of the page).
  - c. Click **Add**.
5. The Home page is redisplayed, and a row for the new Enterprise Server is visible in the table of servers.

### Enable JES

To enable the JES support within MSS:

1. Click **Edit** next to the JCLDEMO row in the table of servers.

The Edit page that appears contains a hierarchy of tabs.

2. Make sure the **Server** and **Properties** tabs are selected, and click **MSS** and then **JES**.
3. Check the **Job Entry Subsystem enabled** check box.

Don't click **Apply** yet — there is more to do on this page.

### Configure the Program Paths

You need to specify the directories and other details needed for the Enterprise Server to find deployed programs. To do this:

1. Enter values into the fields on the **JES > General** tab, as shown in the table below. The paths you enter here point to the directory of your project.



Field	Value to enter:	Details
JES Program Path	c:\tutorials \jcl\bin \x86\Debug	Path for the directory containing executable files to be run by your JCL jobs. This is the path to the Debug folder of your project that contains the .dll files. For debugging, the .idy and .cbl files are used.
System Catalog	c:\tutorials \jcl\catalog \catalog.dat	Path and name of the system catalog. This is a file that holds the details of data files, their attributes and locations.
Default Allocated Dataset Location	c:\tutorials \jcl\catalog \datasets	The default path for the directory where new data sets are to be created, if you do not specify a directory.
System Procedure Library	SYS1.PROCLIB	Set of cataloged PDS data set names that will be searched to resolve JCL procedure names.

2. Click **Apply**.
3. Click **Home** on the Enterprise Server Administration.

### Enable the Console Window

It is useful to have the character-mode console daemon window showing when the JCLDEMO region is running. To enable the console window:

1. Click **Edit** next to JCLDEMO.

The Edit page that appears contains a hierarchy of tabs.

2. Make sure the **Server**, **Properties** and **General** tabs are selected.
3. If **Show Local Console** is already checked, click **Cancel**. If it isn't, check it and then click **OK**.

### Enable Dynamic Debugging

1. Click **Edit** next to the JCLDEMO row in the table of servers.

The Edit page that appears contains a hierarchy of tabs.

2. Make sure the **Server** and **Properties** tabs are selected, and click **General**.
3. Check **Allow Dynamic Debugging**.
4. Click **OK**.

## Start Your Enterprise Server

You can now start JCLDEMO:

1. In Server Explorer, right-click JCLDEMO under the **Micro Focus Servers** group.
2. Choose **Start** from the context menu.

## Define Batch Initiators and Printers

You can run JCL with the following kinds of Service Execution Processes (SEPs):

- Batch initiator (often abbreviated to simply "initiator") — for running JCL jobs
- Batch printer SEP (often abbreviated to simply "printer") — for sending the output from a JCL job to a printer

You can define initiators and printers to start up automatically whenever the Enterprise Server is started, and you can also create ones that last only for the current session.

To define an initiator to start automatically:

1. On the home page of Enterprise Server Administration, click **Edit** next to JCLDEMO.
2. Make sure the **Server** and **Properties** tabs are selected, and click **MSS > JES**.
3. Define a batch initiator as follows:

- a. Click **Initiators > Add**.
- b. Enter values in the fields as shown in the following table:

Field	Value	Comment
Name	INITAB	Not case sensitive — gets folded to upper case
Class	AB	Not case sensitive — gets folded to upper case
Description	Initiator for class A and B jobs	

- c. Click **Add**.
4. Click **Home** on the Enterprise Server Administration.

## Create and Build a JCL Application

These are the steps to create a Visual Studio project comprising all the parts of your application, such as the COBOL source and the JCL batch files. We build the demonstration application in Visual Studio, deploy and run it on an Enterprise Server configured to run JCL applications and debug it dynamically.

### Create a Project

To create a native COBOL project to hold the JCL demonstration program:

1. Create a working folder on your machine such as `c:\tutorials`.
2. Start Visual Studio.
3. Click **File > New > Project > COBOL Projects > Native > Mainframe Subsystem Application**.
4. Specify a name for your project such as `jcl`.
5. Specify your working folder in the **Location** field. This is the folder you defined in step 1 above.
6. Disable **Create directory for solution**.
7. Click **OK**.

This creates a `jcl` folder in your working folder that holds your solution and project.

8. Right-click the solution in Solution Explorer and choose **Open Folder in Windows Explorer**.
9. Using Windows Explorer, copy the demonstration files from the subfolder `Examples\Visual Studio Integration\Tutorials\Mainframe\JCL` in your installation folder. Copy the files to your project folder, `c:\tutorials\jcl`.
10. Using Windows Explorer, create a `catalog` folder in your project folder — `c:\tutorials\jcl\catalog`.

Note: Your project folder needs to have the `catalog` folder created beforehand in order to submit the JCL successfully.

### Import the Files from the Demonstration Program

To import the files from the JCL demonstration program into your Visual Studio project:

1. In Solution Explorer, right-click your project and select **Add > Existing Item**.
2. Set **Objects of type** filter to **All Files (\*.\*)**.
3. Select the `.jcl` and the two `.cbl` files in the project folder.
4. Click **Add**.

### Specify Project Properties

To specify the project properties:

1. In Solution Explorer, double-click the **Properties** folder.
2. On the **COBOL** tab, make sure **Source Format** is set to **Fixed**.
3. Make sure **Enterprise COBOL for z/OS COBOL dialect** is selected.

4. In the **Additional Directives** field, type `charset(ascii)`.
5. On the **Debug** tab, select **JCL** from the drop down list for **Choose Active Setting**.
6. Click the JCL tab on this page — there is no need to specify any settings on it at this point.
7. Press **CTRL + S** to save the changes in the project properties.

## Associate Your Project with the JCL Enterprise Server

The IDE enables you to associate your project with the running JCLDEMO Enterprise Server. To associate your project with the JCL server that you created, JCLDEMO, you need to:

1. Open Server Explorer.  
If it is hidden, display it by choosing **View > (Other Windows >) Server Explorer**.
2. Start JCLDEMO by clicking **Start** in the **Status** column.
3. In Server Explorer, right-click **JCLDEMO** and choose **Associate with project > your project**.

## Build the Solution

To build the solution:

1. Right-click the solution in Solution Explorer.
2. Choose **Build Solution** from the context menu.

This creates the application `.dll` and `.idy` files in the `Debug` subfolder of your project.

## Deploy to Enterprise Server

You do not need to deploy the application to your JCLDEMO enterprise server as you configured the search path of the server to be the `Debug` subfolder of your project. Enterprise Server automatically finds the application files that it needs to run.

## Submit and Run a JCL Job

To submit the JCL application to the running JCL configured Enterprise Server:

1. In the IDE, right-click the `.jcl` file in Solution Explorer.
2. Choose **Submit JCL**.

See the output messages in the Enterprise Server Console Daemon for the result of running the JCL job. Alternatively, you can use drag and drop feature to submit the JCL application to the JCL configured server as follows:

1. Select the `.jcl` file in Solution Explorer.
2. Drag the file to Server Explorer and drop it on the JCLDEMO server.

## View the Spool Queues

The Enterprise Server Monitor and Control (ESMAC) works in conjunction with Enterprise Server Administration and enables you to view and control Enterprise Servers, and the services, packages, and handlers that run on those servers. The spooler is another name for the job entry subsystem, JES. It maintains a number of spool queues, showing the status of jobs in the system.

The IDE enables direct access to the Spool and Catalog on the ESMAC pages.

To see the spool queues:

1. In Server Explorer, right-click **JCLDEMO** under **Micro Focus Servers**.
2. Select **Show spool** from the context menu.

The page that appears shows one of the spooler queues. Select a queue to display using the radio buttons.

### 3. Click **Output**.

This displays the Output Queue. It contains the output from the job you have just run.

### 4. Select the check box to the left of the JCLTEST entry, and click **Display**.

This shows details of the job.

### 5. You can view details of a particular step. For example, click **Details** on the line for the READ step.

### 6. On the same page, click **Display** to view the output from the step.

The output from this step consists of a small number of simple records.

### 7. Right-click in the window and select **Back** to return to the ESMAC pages.

## View the Catalog

The catalog file holds the details of data sets needed for the JCL jobs that run in an Enterprise Server. To view the catalog:

### 1. In Server Explorer, right-click **JCLDEMO** and choose **Show catalog**.

The page that appears shows the catalog file, `MFIJCL. OUTFILE. DATA`, created by the example job.

### 2. Select the check box in front of the catalog file.

### 3. Click **Details**.

### 4. Select **EBCDIC** from the drop down list next to **Codeset**.

### 5. To view the file, click **Display** on the page showing the catalog entry.

The file contents are displayed as a page in the browser.

## Debug Dynamically Under Enterprise Server

The IDE enables you to debug a job running under Enterprise Server. It's not possible to debug the JCL file but you can debug the programs that are invoked by the job that is `EXEC PGM=PROG`.

Before proceeding, stop any Enterprise Server region that may be running.

### Enable Enterprise Server Debugging

Before you can debug within the IDE, you must enable the dynamic debugging feature within your Enterprise Server. This option is available on the General properties page within Enterprise Server Administration.

### Ephemeral Ports

By default, Enterprise Server uses ephemeral ports for its communication listeners, to prevent clashes occurring with other network software on your machine when a listener starts. However, this can make debugging more complex, since Enterprise Server is likely to be assigned a different port each time it starts.

To force Enterprise Server to use a fixed port number for debugging purposes:

### 1. In the IDE, open Server Explorer.

### 2. Right-click **Micro Focus Servers** and choose **Administration** from the context menu.

### 3. On the row for JCLDEMO on Enterprise Server Administration page, click **Details** for the Listeners in the Communications Processes column.

### 4. Click **Edit** on the row for Web Services and J2EE.

### 5. The existing setting should be `*:*`.

### 6. The second `*` indicates use an ephemeral port number.

### 7. Change the Endpoint Address to `*:5055` in order to use a fixed port.

### 8. Click **OK**.

### 9. Click **Home** on the left to return to the Enterprise Server Administration page.

10. Start the Enterprise Server.

During the startup sequence, watch for a mention of dynamic debug enabled. If you missed it, check the Enterprise Server Console in the Server Diagnostics page of Enterprise Server Administration.

## Associate Project with New Listener Port

Now that you have a fixed port, you should reassociate the region with your project in Server Explorer in Visual Studio.

## Start Debugging

Debugging requires no changes to your program code. It requires that your project is associated with the JCLDEMO Enterprise Server and that the Enterprise Server is configured to enable dynamic debugging, which you performed in the previous steps.

1. Double-click any of the .cbl files in Solution Explorer.
2. Choose **Debug > Start Debugging** or press **F5**.

The debugger enters a wait state.

3. Introduce a breakpoint in the procedure division section of the code.

If you now submit the JCL file, the debugger stops in your JCLCREATE program.

4. Step through a few lines and press **F5** to continue.

The debugger should stop again in the JCLREAD program.

5. Once you've completed your debug session, select **Debug > Stop Debugging**.

Notice the messages that have appeared in the Enterprise Server console window.

The debugger offers you a number of choices when specifying what you want to debug. You may choose a Job Name, a Job Number, a Step Name or the top-level program on the **Debug** tab of the project properties.

These choices enable you to perform remote debugging, to support multiple users debugging on the same region.

Given that you're running Enterprise Server locally and no one else is connecting to it, you do not need to specify any options and the debugger debugs any debuggable program that it finds.

## View the List of Attached Debuggers

You can view the list of debuggers waiting to attach to your region using ESMAC:

1. Click **Edit** on the row for JCLDEMO in Enterprise Server Administration.
2. Click the **Control** tab under **Server**.
3. Click **ES Monitor & Control**.

This opens the ESMAC page.

4. Click the **Dyn-Dbg** button at the bottom left of the page.

You can also remove these debuggers from the list.

## Debug Remotely Under Enterprise Server

You can submit JCL to a region working on the remote machine, to enable it to access Enterprise Server. For this purpose the remote machine must have Enterprise Server with JCLDEMO configured and working.

Make sure the firewall is not working on the remote machine to be able to access Enterprise Server.

1. In the IDE, right-click **Micro Focus Servers** in Server Explorer.
2. Select **MFDS Configuration**.

3. Type the name of a remote machine in the **Host name** field, or specify its IP address in the **IP Address** field.
4. Click **OK**.
5. Refresh the Micro Focus Servers group.
6. Right-click **JCLDEMO** on the remote machine and choose **Associate with project > your project**.
7. In Solution Explorer, right-click the `.jcl` file and choose **Submit JCL**.
8. See if you can debug the JCLCREATE and JCLREAD applications running on the remote machine.

## More Information

For more information about using Studio Enterprise Edition, refer to the Help either in the product or on the Micro Focus Web site.

# Tutorial: Compiling, Linking and Debugging Open PL/I Programs Running Under ES/JCL

## Overview

In this tutorial, you:

1. Create and configure an enterprise server to run Open PL/I programs under JCL.
2. Compile and link Open PL/I programs for execution under the control of JCL.
3. Debug Open PL/I programs.

## Prerequisites

To run this tutorial, you need to:

- Have Microsoft Visual Studio and Studio Enterprise Edition installed.
- Copy to your root the `nxopdemo` folder located in the `Examples` directory in the installation of your development system. By default this is `C:\Program Files\Micro Focus\Studio Enterprise Edition 6.0\Examples\OPEN-PLI`.
- Create two empty directories called `data` and `debug` under `C:\nxopdemo`.

## Configure an Enterprise Server to Run Open PL/I Programs Under JCL

These are the steps to create and configure an enterprise server to execute Open PL/I programs under the control of JCL.

### Configure an MSS-enabled Enterprise Server

1. On the Start menu, click **All Programs > Micro Focus Studio Enterprise Edition x.x > Configuration > Enterprise Server Administration**.
2. On the Home page of Enterprise Server Administration, click **Add** at the bottom of the table of servers.
3. On the first Add Server page, type `PLIDEMO` in the **Server Name** field.
4. Make sure 32-bit is selected in the **Working Mode** section.



**Note:** Open PL/I is currently a 32-bit product.

5. Click **Next**.
6. Select **Micro Focus Enterprise Server with Mainframe Subsystem Support**.

7. Click **Next**.
8. Under **Creation Options**, uncheck **Create TN3270 listener**.
9. Type the following in the **Configuration Information** field:

```
[ES-Environment]
CODEWATCH_SRCPATH=c:\nxopdemo
CODEWATCH_STBPATH=c:\nxopdemo
ES_PL1_MFFH_JCL=Y
```

10. Click **Add**.
11. The Home page is redisplayed, and a row for the new enterprise server is visible in the table of servers.

## Enable JES

1. On the **Enterprise Server Administration** home page, click **Edit** next to the PLIDEMO row in the table of servers.

The Edit page that appears contains a hierarchy of tabs.

2. Make sure the **Server** and **Properties** tabs are selected, and click **MSS** and then **JES**
3. Check the **Job Entry Subsystem enabled** check box.

Don't click **Apply** yet - there is more to do on this page.

## Configure Program Paths

You need to specify the directories and other details needed for the enterprise server to find deployed programs. To do this:

1. Enter values into the fields on the **JES > General** tab, as shown in the table below. The paths you enter here point to the directory of your project.

Field:	Value to enter:	Details:
JES Program Path	c:\nxopdemo\debug	Path for the directory containing executable files to be run by your JCL jobs. This is the path to the Debug folder of your project that contains the Open PL/I .dll files.
System Catalog	c:\nxopdemo\catalog.dat	Path and name of the system catalog. This is a file that holds the details of data files, their attributes and locations. If the catalogue does not exist, it will be created.
Default Allocated Dataset Location	c:\nxopdemo\data	The default path for the directory where new data sets are to be created, if you do not specify a directory.

2. Click **Apply** to save the changes.

## Add an Initiator

Initiator (or batch initiator) is a Service Execution Process for running JCL jobs. You can define initiators to start up automatically whenever the enterprise server is started, and you can also create ones that last only for the current session. We will define one initiator to start automatically.

1. Go to the **Initiators** tab.
2. Click **Add**.
3. Type *INIT1* in the **Name** field.
4. Type the job class in the **Class** field.

It can be *abcdefghijklmnopqrstuvwxy* or *0123456789*. For the purposes of this tutorial, type *abcdefghijklmnopqrstuvwxy*.

5. Click **Add**.
6. Click **Home** on the Enterprise Server Administration page.

## Start the Enterprise Server

1. On the Start menu, click **All Programs > Micro Focus Studio Enterprise Edition x.x > Compatibility Tools > Net Express 32-Bit Command Prompt**.
2. Change to `c:\nxopdemo`.
3. Enter `casstart -rPLIDEMO`.

Don't close the command prompt, as you are going to need it later in this tutorial.



**Note:** If you use the Enterprise Server Administration page to start your server, you will not be prompted to debug your program, unless you have modified your configuration such that your Micro Focus Directory Service runs as your logon ID and has the ability to interact with the Windows Desktop.

## Compile and Link Open PL/I Programs Under JCL

These are the steps to compile and link Open PL/I programs for execution under the control of JCL.

1. Go back to the Net Express command prompt that you used earlier to start the enterprise server.
2. Enter the following command:

```
mfplx opdemo.pli -deb -defext -mvs -dll -out:c:\nxopdemo\debug\opdemo.dll
```

This will compile and link `opdemo.pli`. Files will be copied to `c:\nxopdemo\debug`.

3. Build `opdemo2.pli` in the same way, replacing `opdemo.pli` with `opdemo2.pli` and `opdemo.dll` with `opdemo2.dll` in the command above.

```
mfplx opdemo2.pli -deb -defext -mvs -dll -out:c:\nxopdemo\debug\opdemo2.dll
```

## Debug Open PL/I Programs Under JCL

After you have compiled and linked `opdemo.pli` and `opdemo2.pli`, you can start debugging.

### PLIDEBUG.DAT

The `PLIDEBUG.DAT` file contains settings defining whether the debugger should automatically attach when the program is executed or whether it should shut down when the program completes. It also contains an initial set of debugger commands that are executed when the debugger is attached.

`PLIDEBUG.DAT` is located in the installation directory. By default this is `C:\Program Files\Micro Focus\Studio Enterprise Edition 6.0\Base`.



**Note:** The file is not there when you install the product, it appears when you first try to execute a job.

To view the file and edit it:

1. Start Net Express by typing `mfnetx` in the Net Express command prompt.
2. Click **File > Open**.
3. Browse to the installation directory of Open PL/I, select `PLIDEBUG.DAT` and click **Open**.


Find the lines of `OPDEMO` and `OPDEMO2`. They should look like this:

```
OPDEMO          Y Y shlib opdemo.dll; env opdemo; br %ENTRY; br %EXIT
[det;q];c
```

```
OPDEMO2        Y Y shlib shlib opdemo2.dll; env opdemo2; br %ENTRY; br
%EXIT [det;q];c
```





The `Y` in the first column indicates if the debugger will attach when the step starts. The `y` in the second column shows if the debugger will shut down when the step ends.

 **Note:** You still need to use the `DETACH` command.


The rest is a series of commands that will be issued to the debugger when it starts.

If the two lines are not there, you need to create them.

 **Note:** Make sure that the first `y` is in the 21st column.

 **Note:** Always close down the editing session before you start debugging because otherwise the file will be locked and Enterprise Server won't access it.

## Start Debugging

 **Note:** Make sure you have closed the editing session of `PLIDEBUG.DAT`.

1. Go back to the Net Express command prompt that you used earlier to start the enterprise server.
2. Run `cassub -rPLIDEMO -jopdemo.jcl`.
3. The Open PL/I debugger, CodeWatch, will pop up and you can start debugging.

## Debug Using CodeWatch


CodeWatch provides debugging facilities, such as stepping, setting breakpoints or tracing. You can examine the source program, find specified text strings, move to specified line numbers, and examine included source files.

The CodeWatch graphical interface consists of a source window, output window, and edit-line used for command-line input, plus a set of buttons. You can enter commands either using the buttons or via the edit-line.

To see some of the CodeWatch debugging facilities:

1. Click **Continue** or type `C` in the edit-line. It begins program execution.
2. Click **Step** or type `S` in the edit-line. This executes the first statement.
3. Type `S 4` and this will execute the next 4 statements.
4. Type `P 10` in the edit-line. It will print the next 10 lines in the output window.
5. Type `BR 106` and this will set a breakpoint on line 106.
6. Type `C` and the statements until the breakpoint will be executed.
7. Type `DETACH` to detach from the process. CodeWatch will shut down and then pop up with `opdemo2.dll` ready for debugging.
8. Type `S 7` and this will execute the next 7 statements.
9. Type `EVALUATE SYSPRINT_BUFF` to evaluate the `SYSPRINT_BUFF` variable.
10. You can continue stepping or if you want to stop debugging, type `DETACH`.

For more information on debugging with CodeWatch, see the CodeWatch Reference Manual.

 **Note:** To stop the enterprise server, type `casstop -rPLIDEMO`

## Java CodeWatch




This new user interface has been designed to streamline debugging tasks and has a host of usability improvements such as persistent break points, watch lists, etc. It is written in Java and to use it you need to have the latest Java Runtime Environment (JRE) or Java Development Kit (JDK) installed.

You also need to copy `cw_java.jar` from the `Bin` directory in the installation of your development system to the `debug` directory under `C:\nxopdemo`

Which debugger you use is controlled by an environment variable:

- `MF_USE_JAVA_CW=Y`, if you want to use the Java debugger.
- `MF_USE_JAVA_CW=N`, if you want to use the standard debugger.

To see some of the Java debugger's facilities:

1. When the Java debugger runs, you will see the source code displayed on the left and some panels on the right.
2. Scroll down in the source code panel until you can see the line beginning with `SYSPRINT_BUFFER = 'ABC...'`.
3. Double-click on this line to set a breakpoint at this instruction.
4. Click  on the top left of the debugger to continue to this breakpoint.
5. Click  to step over this line.
6. Place the mouse cursor above the word `SYSPRINT` in the source code panel to see the contents of this variable.
7. Right-click on the word `SYSPRINT` and click **Add 'SYSPRINT' to watch panel** to make it a permanent watch item.
8. You can keep stepping at this point or press the  in the top right corner to close the debugger.

# Index

## B

- Batch enabling
  - Enterprise server 16
- batch initiator
  - defining 17

## C

- Catalog
  - viewing 20
- COBOL
  - about developing in Visual Studio 9

## E

- Enterprise Server
  - Dynamic debugging 20
  - JCL demonstration 15, 18
  - JCL enabling 16
  - JCL tutorial 15
  - MSS enabling 16
  - Remote debugging 21
  - running JCL demonstration 19

## G

- Getting Started 5

## I

- initiator
  - defining 17

## J

- JCL
  - demonstration 15
  - tutorial 15, 18
  - running demonstration 19
  - tutorial 15, 18
- JCL enabling
  - Enterprise server 16
- JCL project 15, 18

## M

- MSS
  - JCL demonstration 15, 18
- MSS enabling
  - Enterprise server 16

## N

- native code 5

- network printer
  - making available to JCL 17

## O

- Overview
  - JCL tutorial 15

## P

- printer
  - making available to JCL 17
- printer SEP
  - defining 17
- projects
  - about COBOL in .NET 9
- properties
  - about COBOL project properties 8, 13

## R

- references
  - about COBOL project references 12
- Running
  - JCL demonstration 19

## S

- Server Execution Process
  - batch and printer 17
- solutions
  - about COBOL in .NET 9
- Spool queues
  - viewing 19
- Spooler 19

## T

- tutorial 5
- Tutorial
  - JCL 15
  - JCL in Enterprise Server 15, 18
  - mainframe 15
- tutorials
  - COBOL in .NET 9

## V

- Visual Studio 5