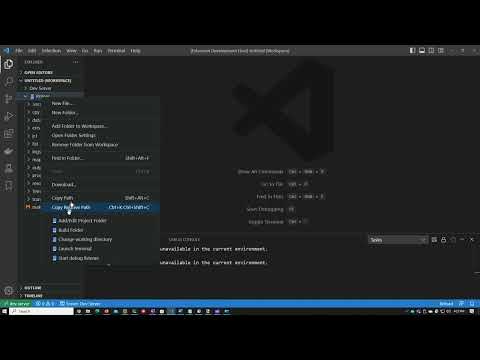
Link

<https://www.youtube.com/embed/tGEyy3zUS-w?modestbranding=1&showinfo=0&autoplay=1&rel=0>

Start with this….

https://www.youtube.com/embed/

<https://www.youtube.com/embed/arTheBZ94to?modestbranding=1&showinfo=0&autoplay=1&rel=0>

[](https://www.youtube.com/embed/arTheBZ94to?feature=oembed)

https://www.youtube.com/embed/tGEyy3zUS-w?modestbranding=1&showinfo=0&autoplay=1&rel=0

https://www.youtube.com/embed/tGEyy3zUS-w?modestbranding=1&showinfo=0&autoplay=1&rel=0

**Installation**

Sara

Open your browser to the VS Code download page.

Since I am the only user on this PC, I’ll choose User Installer 64 bit.

When it is finished downloading, I’ll double click the setup .exe file.

Read and accept the terms

Choose the default location and optionally click the Explorer integration options.

Sara

Click next and wait for the install to complete.

To install the NTT DATA VS Code Extension for COBOL, choose the extensions icon in vs code.

Click on the ellipsis icon, and choose Install from v s i x.

Guy

I always choose the Explorer integration options. It is such a great feature

Sara

Click Finish and wait for the install to complete.

Now that VS Code is installed, let’s add the VS Code extension for NTT DATA COBOL.

Click on the Extensions button and then on the ellipsis button. Choose Install from V S I X.

Navigate to your downloads folder and choose the V S I X file.

That’s it, installation is complete. Continue with the User Guide for details on using the extension.

This is the only software you will need to install.

There are no licenses or 3rd party software required.

**Server Connection**

Sara

Now that you have installed VS Code and the NTT DATA COBOL extension, we need to connect to a server.

In the Activity bar on the left side of the window, click on the server icon and then the connect to a remote server button.

Enter a friendly name for the server, the host or IP address, the port number and a username.

Next, enter either a password or a SSH private key file on your computer. Instructions on how to create public/private keypairs are discussed in the User Guide.

Finally, enter the initial directory to open once the connection has been established.

Back on the server tab, click on your new connection. Expand the Workspace section in Explorer, and you will see the contents of your folder.

Guy - low

Great work Sara, I would just like to add that when setting the initial directory, the user account should have read and write access to that folder. You should choose your home directory, or the directory where your projects are stored just as you showed.

Build Command

Now that we have a connection to our server we can start building COBOL projects. In my dev folder I have the Primer project that is included with UniKix TPE. I can see my source and other files in the explorer.

The Primer source also includes a make file which I use to build the sample.

Let’s explore building a project. Here I am at a command prompt on my server.

I can source my environment like this.

And then run the make command.

Hmmm, looks like there’s some errors.

Guy - low

Thanks. Let’s forget about those errors for now.

I’d like to show you how we can automate these steps with our VS Code extension and what new features become available.

Open VS Code and select the servers icon and connect to our dev server.

Back in Explorer, navigate to the Primer folder and right click on it.

Choose the Add/Edit Project folder option.

Here there will be many options for your Project, including a name, build and compile commands and more. We cover all these features in the User Guide.

For the build command, I’ll enter the exact commands that you used in the command line.

Since most of the features in your Project Folder will need to source the environment. We can move the command to source the environment to here so it will be used for building, debugging and running your programs.

Let me enter the Output and Source folders – we will need these later.

When you create a Project Folder, the Name of the folder is listed at the same level as your initial directory folder to make it easier to find your projects. The file structure has not been changed.

Right click the project folder and choose build. The output of the make command is shown in the terminal window. The errors are highlighted in red.

Also notice that the Problems tab now reports one problem. You can open the problems tab and click on the error which will open up the source and navigate to the line with the error.

Let's fix the error and retry the build. Note the red squiggly line has disappeared and no Problems have been reported.

We can also use the make utility to compile individual files. Back in the Project Folder dialog, I’ll paste these commands into the Compile section.

The make utility uses two targets, the CL2 file and the CLASS file. Notice that there is a substitution variable named basename which is replaced with the filename being edited. substitution variables are discussed in the User Guide.

The compile command will use the Source environment commands that we entered earlier. And with the Compile commands set, we can also Enable syntax checking as we edit the file

Back in the Editor, let’s make that same typo again. Notice the red squiggly line appears and a Problem has been reported. After we correct the error the problems go away. Please review the User Guide for important information regarding syntax checking.

**TASKS2**

Sara,

I’ve been working with the Project Folder settings for Build, Compile, and BPE SUB commands. While these options work for most cases, my users have special tasks they need to perform – like running make clean, resetting the test database, or testing JCL scripts on multiple subsystems. I don’t want my users performing these tasks from the command line if possible.

Guy(low)

Sure, let’s talk about a very powerful feature in VS Code – called Tasks. Tasks can perform any task that you can execute from the command line and can contain customizable parameters.

Let’s create a VS Code task to do what you need . Let’s create a tasks.json file manually in our project's V S Code folder so it is specific to our project.

Always check the Problems tab to make sure we don’t have any typos.

Now, press F1 and choose Run task. Our new task appears in the list. Select Run BPE SUB program and press ENTER.

Jon (low)

Let’s talk about a very powerful feature in VS Code – called Tasks. Here is a very simple COBOL program that is executed from a JCL batch script. Here is the source which displays some output. And here is the JCL that executes that program.

Let’s create a VS Code task to execute my job. VS Code will want to create the task at the topmost folder in our workspace named UNI kix. So, we will create the tasks.json file manually in our simple project's V S Code folder.

In the tasks.json file, I’ll paste in the sample json text from the User Guide and modify it for our JCL job.

Let’s change the label and the commands to be executed.

Always check the Problems tab to make sure we don’t have any typos.

Now, press F1 and choose Run task. Our new task appears in the list. Select Run BPE SUB program and press ENTER.

In a moment, our job will run and in the terminal window we can see the output of our job.

Sara,

But what if we don’t want to hard code the subsystem and job name?

Jon,

Right. VS Code has a feature called inputs, where the user can enter text string or pick text from a list. Let’s create an input for the subsystem. I’ll enter an ID for the input, the input type which is prompt string, the description that will appear in prompt for the subsystem, and a default for the subsystem.

I’ll replace the hard coded sub1 string with the user’s input. Note the dollar brace syntax for substitution strings.

No problems. So, now press F1 and Run task. When I select our task, notice that we are now prompted for the subsystem. We can either choose the default or enter a new subsystem and press Enter. Or, I can press escape to cancel the task because we are not yet finished modifying it.

In my project folder, I have a Build command that copies the output to the appropriate areas for my BPE environment.

Let’s create a VS Code task to execute my job. VS Code will want to create the task at the topmost folder in our workspace named UNI kix. So, we will create the tasks.json file manually in our simple project's V S Code folder.

In the tasks.json file, I’ll paste in the sample json text from the User Guide and modify it for our JCL job.

Let’s change the label and the commands to be executed.

Always check the Problems tab to make sure we don’t have any typos.

Now, press F1 and choose Run task. Our new task appears in the list. Select Run BPE SUB program and press ENTER.

In the terminal window we can see the output of our job.

SARA

But what if we don’t want to hard code the subsystem and job name?

Right. VS Code has a feature called inputs, where the user can enter text string or pick text from a list. Let’s create an input for the subsystem. I’ll enter an ID for the input, the input type which is prompt string, the description that will appear in prompt for the subsystem, and a default for the subsystem.

I’ll replace the hard coded sub1 string with the user’s input. Note the dollar brace syntax for substitution strings.

Now, press F1 and Run task. When I select our task, notice that we are now prompted for the subsystem. We can either choose the default or enter a new subsystem and press Enter. Or, I can press escape to cancel the task because we are not yet finished modifying it.

Let’s replace the hard coded JOB name with a drop down list of jobs. Just like the prompt string, I enter an ID for the input, a description, and use the type pick string to create a drop down list.

The list uses an options array of strings. I’ll enter some sample JOBS here. And then choose the default option.

Like before, we will replace the hard coded JOB name with a substitution string.

Great, no problems reported, let’s try our task.

Now, we can enter the subsystem and the JOB to run. And you’ll see the output of the job in the terminal window.

SARA

Great, but my project has hundreds of jobs. Is there another way to choose which job to run?

I’ll keep the default sub1 for the subsystem, and choose PROGRAM1 for the job to run. In the terminal, we can see the output of our JCL job.

Thanks. Please see the VS Code documentation on Tasks. You can use things like global variables

SARA

Impressive, but my project has hundreds of jobs. Is there another way to choose which job to run?

Sure, we can replace the drop down list with the currently selected file in the editor using the basename substitution string.

When I go back to my project and open a JCL file, that JCL filename will be used in our BPE SUB command.

SARA,

Perfect, this will help to minimize the use of the command line for my users. Please see the VS Code documentation on Tasks. You can use things like global variables and more.

Problem matchers

Here I have a sample COBOL program opened in VS Code. It’s part of my Demo projects that consist of over ten thousand files.

The COBOL program references a number of copy books. Let’s say I want to see which programs, or other copy books also reference this file.

Double click the copybook filename and press CONTROL SHIFT F.

This will open up the Find in Files pane in VS Code.

Here you see it found 391 results in 41 files. The folder that was recursively search is also listed.

EMMA

But it also lists similar copybook filenames like D P O R L 2 and D P O R L 3

ANDREW

We can fix that by choosing a option in the search window. Here you can select to Match Case and Whole Word.

Choosing Whole Word now lists only the filename we clicked on.

And you aren’t limited to copy book file names. You can search for any text.

Here we will search for all references to POLICY NUMBER.

Double click and press CONTROL SHIFT F.

Now it shows 16 thousand results in three hundred and 15 files.

If I collapse the list, I can see there are a variety of file extensions

Clicking on the ellipsis I can see the file extensions that I want to include or want to exclude. Since copy books might not have an extension, I’ll add extensions to the exclude option.

Now you see that c b l and list files are not included in my results.

I can click on any result and the file will be opened to the line containing our search term.

Thanks for watching.

Debug Port Tunnel

ANDREW EMMA

Debugging COBOL programs use a number of different ports. For example, debugging COBOL programs directly, use default port 9 9 9 9 . Batch programs use a dynamic port starting with 9 9 0 0 plus the job number. C E D F has no default port number.

Now, these ports must be accessible through your server’s firewall. Here I will open up the configuration for a virtual machine running in Microsoft Azure.

We are using Azure simply because it is easy to see a firewall configuration in a graphical interface. The same principles apply when using a local area network, your corporate network or data centers in the cloud.

Here you see I have ports opened for S S H, T P E Manager and 32 70. No COBOL ports are opened.

If I go back and try to debug a COBOL program.

I will eventually get a connection error because the port is blocked.

There are ways around this. You can ask the system administrator to open up ports for COBOL, Online and Batch. And if you are working with a team of developers, each developer might require a different port. That’s a lot of work and a potential security risk.

Or, you can run a S S H tunnel from the command line. It might look like this.

Another way is to use a third party program like putty and configure S S H tunnel options.

You are responsible for entering all the port numbers and IP addresses. All this requires running a separate program and login. It can be confusing to some users.

But our extension now supports automatic tunneling for Online, Batch and direct debugging of COBOL programs.

Under the Servers option select your connection and place a checkmark on the Tunnel Debug Ports.

You do not need to enter any IP address or port number. Everything is managed for you.

Now, when I try to debug my program. You’ll see the connection is successful and my debugging commands work perfectly.

One thing to remember when debugging online programs using C E D F, is that you need to enter local host as the destination address for the connection and NOT your workstation address.