

Modern Buildable Projects with IBM i Project Explorer and Bob

Edmund Reinhardt, IBM

Product Architect - IBM i App Dev & AI Solutions for IBM i

edmund.reinhardt@ca.ibm.com

Sanjula Ganepola, IBM

Software Developer – IBM i App Dev & AI Toolchain

Sanjula.Ganepola@ibm.com

IBM i



Agenda

- Challenges with Building on IBM i
- How does local development overcome this?
- What are IBM i Projects?
- What is Bob and how to use it?
- Ins and Outs of IBM i Project Explorer

Challenges with Building on IBM i

Building on IBM i is limiting...

- 1 SRC-PF
 - 10 char names
 - Fixed record length
 - Not accessible to open ecosystem, including Git and Make
 - Source of the same type stored in QxxxSRC to avoid name conflicts (member type does not disambiguate)
- 2 Libraries
 - Only 2 level hierarchy to organize, with only short 10 char names
- 3 Source control
 - None (sequence number dates)
 - Home grown
 - Proprietary IBM i systems
 - Cost
 - Smaller market = less investment
- 4 Build system
 - Individual CRTXXXMOD + CRTPGM
 - CL Scripts
 - A couple of vendors have dependency-based build

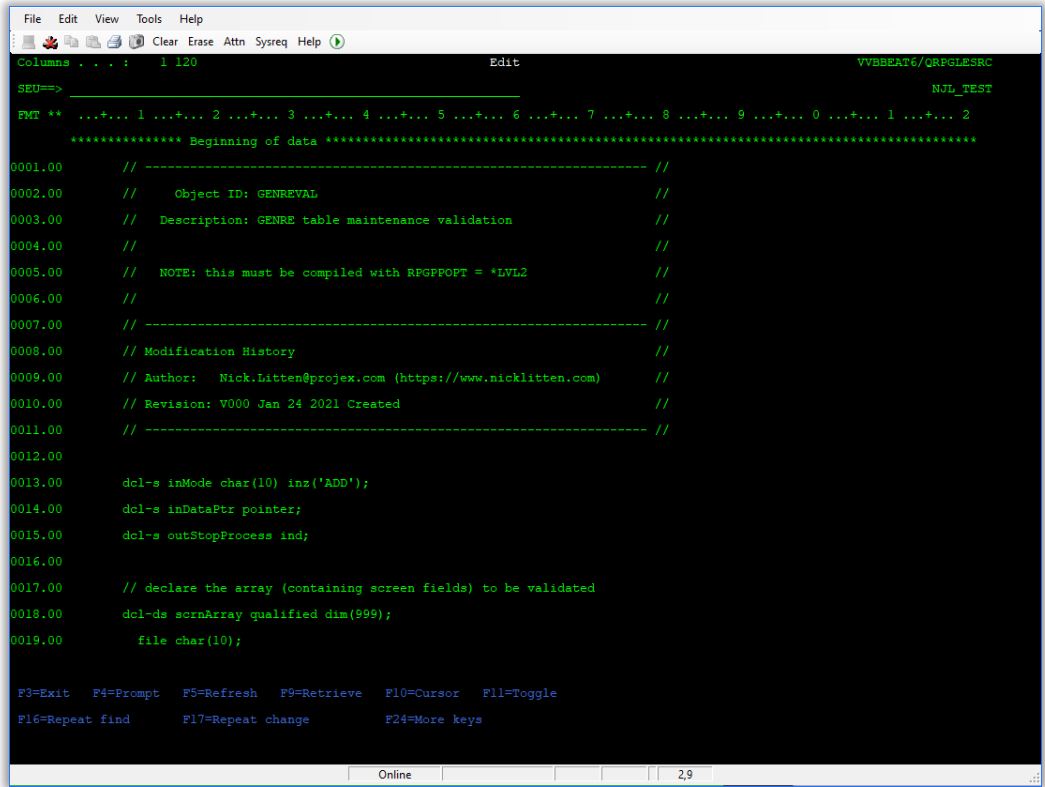
Let's look ahead to the future

New
Developers
Incoming!

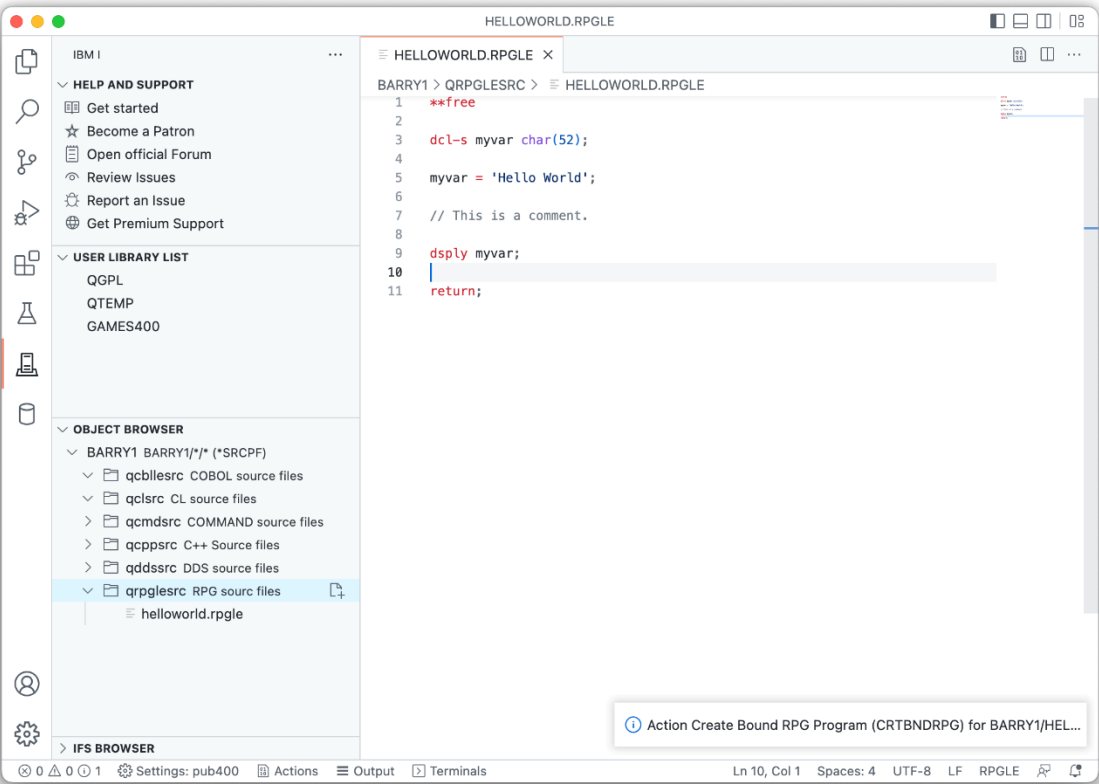
Need for Modernization is Urgent

New development tools/ecosystems
(ie. Code for IBM i/Merlin)

Modern development practices (ie. Git)



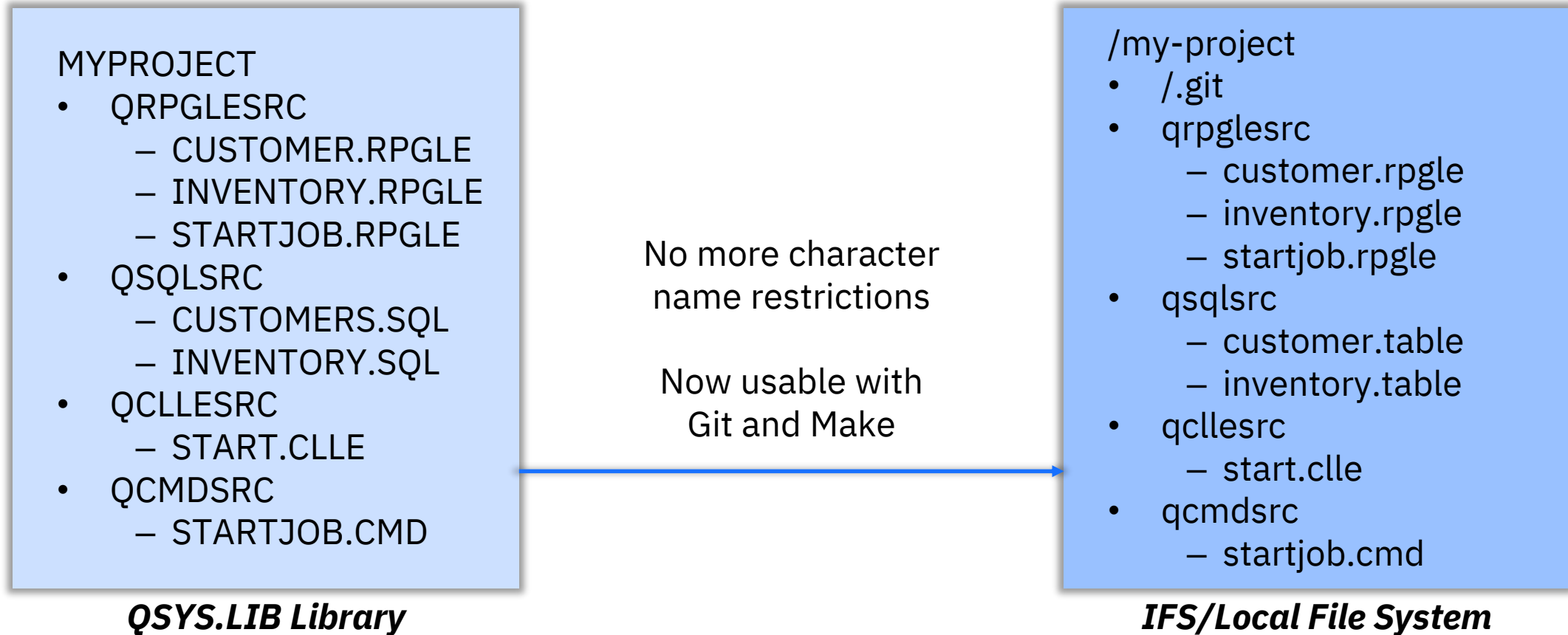
A screenshot of a legacy IBM i development environment. The interface is a green-on-black terminal window. At the top, there's a menu bar with 'File', 'Edit', 'View', 'Tools', and 'Help'. Below the menu, there's a status bar showing 'Columns: 1 120' and 'Edit'. The main area displays RPGLE code. The code starts with a comment '***** Beginning of data *****' followed by a series of comments and data declarations. The code is written in a monospaced font, with some lines highlighted in green. At the bottom, there's a status bar showing 'Online' and '2,9'.



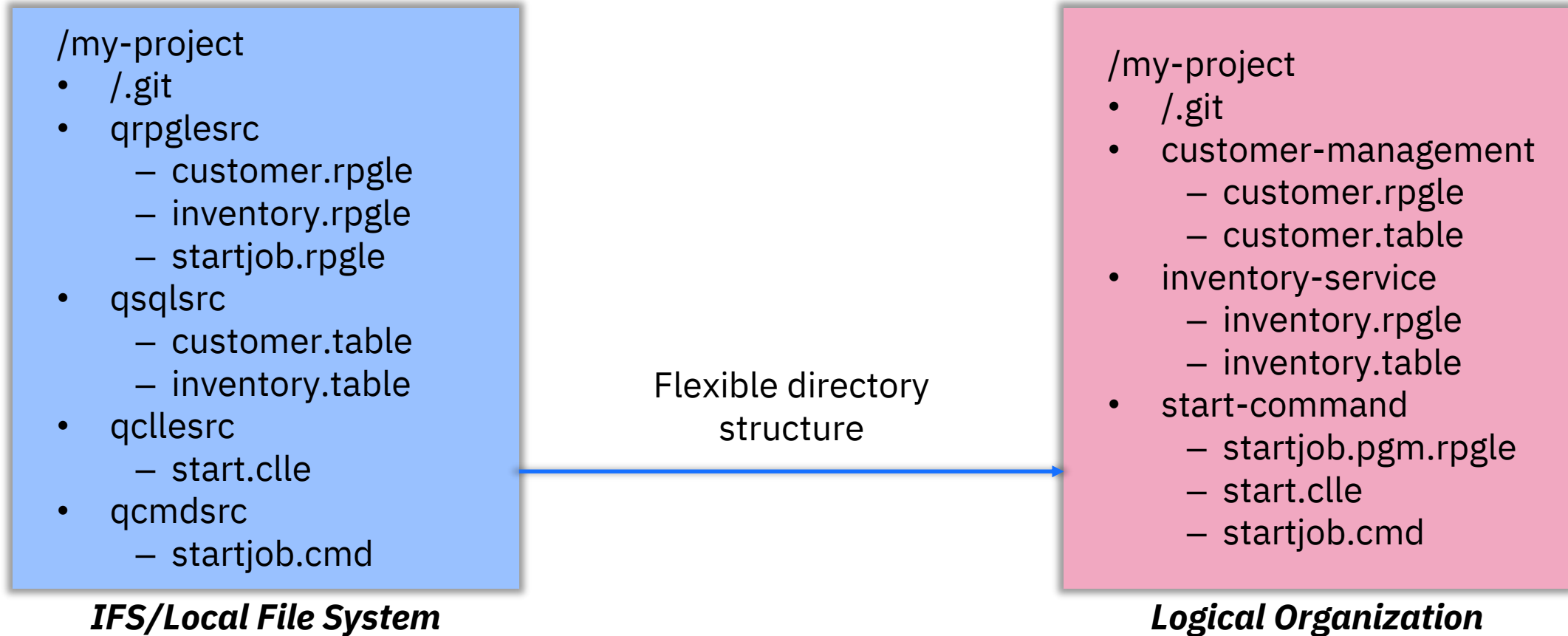
A screenshot of a modern IBM i development environment. The interface is a light-colored code editor with a sidebar on the left. The sidebar contains a 'HELP AND SUPPORT' section with links like 'Get started', 'Become a Patron', 'Open official Forum', 'Review Issues', 'Report an Issue', and 'Get Premium Support'. Below this is a 'USER LIBRARY LIST' section showing 'QGGL', 'QTEMP', and 'GAMES400'. At the bottom of the sidebar is an 'OBJECT BROWSER' section showing a tree view of objects, including 'BARRY1 BARRY1/* (*SRCPF)' and 'qrplesrc RPG src files'. The main area displays the code for 'HELLOWORLD.RPGLE'. The code is written in a monospaced font, with some lines highlighted in blue. At the bottom, there's a status bar showing 'Ln 10, Col 1', 'Spaces: 4', 'UTF-8', 'LF', 'RPGLE', and a button 'Action Create Bound RPG Program (CRTBNDRPG) for BARRY1/HEL...'.

How does local development overcome this?

Let's use a different (but similar) file system

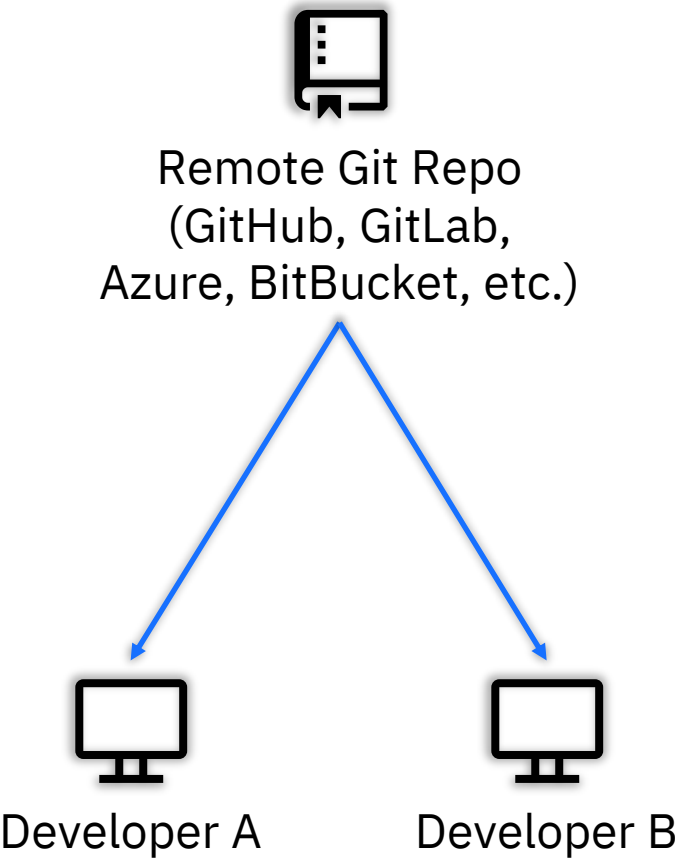


Let's go one step further!

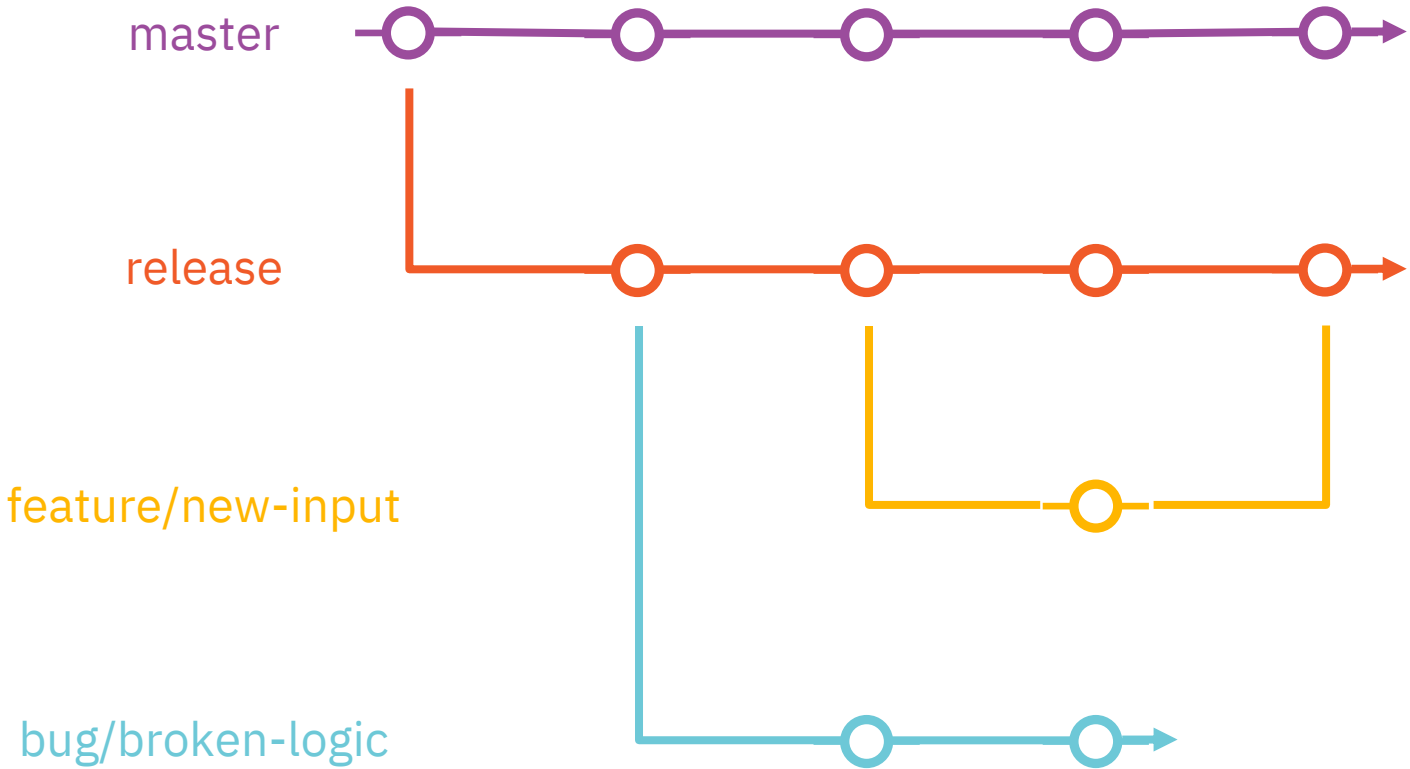


Unlocking source control with Git

Distributed Development



Version Control and Git Workflow



Did we solve our problems?

1 ~~SRC-PF~~

- ~~— 10 char names~~
- ~~— Fixed record length~~
- ~~— Not accessible to open ecosystem, including Git and Make~~
- ~~— Source of the same type stored in QxxxSRC to avoid name conflicts (member type does not disambiguate)~~



2 Libraries

- ~~— Only 2 level hierarchy to organize, with only short 10 char names~~



3 ~~Source control~~

- ~~— None (sequence number dates)~~
- ~~— Home grown~~
- ~~— Proprietary IBM i systems~~
 - ~~• Cost~~
 - ~~• Smaller market = less investment~~



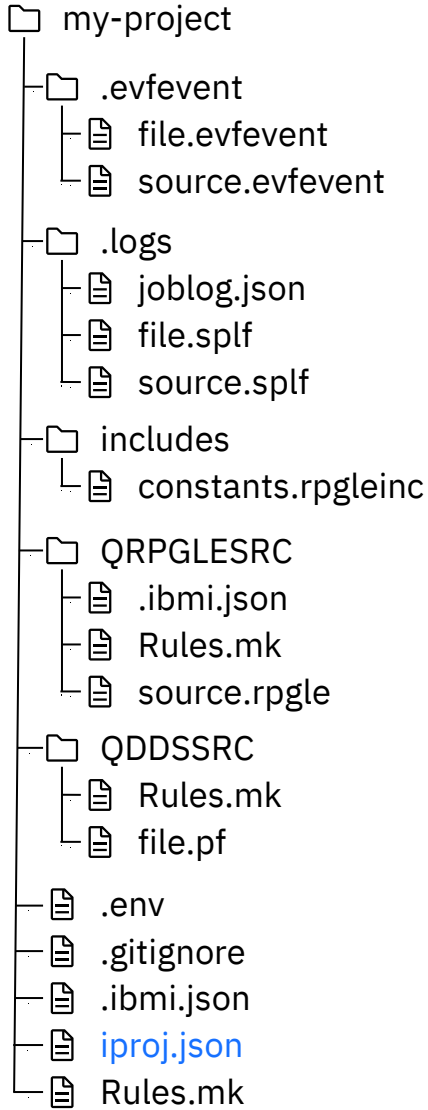
4 Build system

- Individual CRTXXXMOD + CRTPGM
- CL Scripts
- A couple of vendors have dependency-based build



What are IBM i Projects?

Projects that self-describe how to build themselves!?



Project Information

Configure library list

Configure build/compile environment

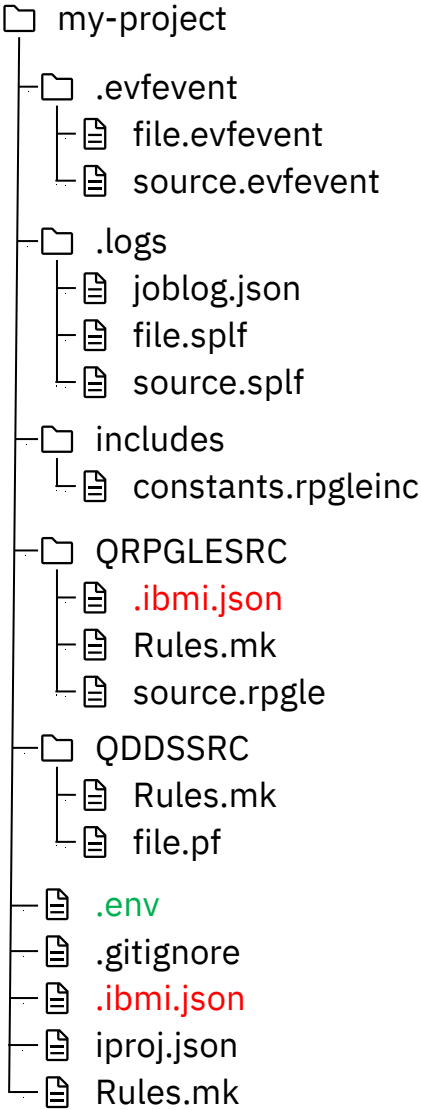
```
{ } iproj.json x
{ } iproj.json > ...
1 {
2   "version": "0.0.1",
3   "description": "SAMPLE PROJECT",
4   "repository": "https://github.com/edmundreinhardt/bob-recursive-example.git",
5   "license": "Apache 2.0",
6   "objlib": "&CURLIB",
7   "curlib": "&CURLIB",
8   "includePath": [
9     "includes",
10    "QPROTOSRC"
11  ],
12  "preUsrlib1": [
13    "&lib1"
14  ],
15  "postUsrlib1": [
16    "&lib2"
17  ],
18  "setIBMiEnvCmd": [],
19  "compileCommand": "makei c -f {filename}",
20  "buildCommand": "makei build"
21 }
```

Standardized metadata format with variables (&...)

Set build/compile command

iproj.json in project root

Flexible subdirectories and build customization



```
{ } .ibmi.json X
{ } .ibmi.json > ...
1 {
2   "version": "0.0.1",
3   "build": {
4     "tgtCcsid": "273",
5     "objlib": "&lib3"
6   }
7 }
```

.ibmi.json in project root or subdirectories

EBCDIC encoding for compiler

Target object library for directory

```
gear .env X
gear .env
1 LIBL=QGPL QTEMP QDEVELOP QBLDSYS QBLDSYSR
2 CURLIB=SANJULA
3 lib1=MYLIB
4 lib2=ABCLIB
5 lib3=APILIB
```

.env in project root

Custom variable values so that each developer can customize build

What is Bob and how to use it?

What is Bob?

Free and open-source build system to build QSYS objects on IBM i

- 🚀 Speed: Compile objects that need recompiling (new or changed source code)
- ⚙️ Reliability: If an item changes, then it and everything it depending on it will be rebuilt
- 💎 Industry standard: Object dependencies are specified using standard makefile syntax
- 🛠️ Flexibility: Override compile parameters, write custom recipes (*If you can code it, you can build it!*)
- ☀️ Ease of use: Build with a single command or a single button in an IDE (*IBM i Project Explorer in VS Code*)

```
PROBLEMS 74 IBM i COMMENTS OUTPUT DEBUG CONSOLE TERMINAL PORTS IBM i JOB LOG 1

-bash-5.2$ makei b
> /QOpenSys/pkgs/bin/make -k BUILDVARSMKPATH="/tmp/tmp1uix3o3" -k BOB="/QOpenSys/pkgs/lib/bob" -f "/QOpenSys/pkgs/lib/bob/src/mk/Makefile" all
make: Warning: File '/home/SANJULA/builds/ibmi-company_system/qrpglesrc/employees.pgm.sqlrpgle' has modification time 14354 s in the future
=== Creating SQL TABLE MERTESTBLD/DEPARTMENT from Sql statement [department.table]
RUNSQLSTM srcstmf('/home/SANJULA/builds/ibmi-company_system/qddssrc/department.table') DBGVIEW(*SOURCE) TGTRLS() OUTPUT(*PRINT) MARGINS(1024) COMMIT(*NONE )
✓ DEPARTMENT.FILE was created successfully!

=== Creating DSPF [depts.dspf] in MERTESTBLD
/QOpenSys/pkgs/lib/bob/src/scripts/crtfrmstmf --ccsid *JOB -f /home/SANJULA/builds/ibmi-company_system/qddssrc/depts.dspf -o DEPTS -l MERTESTBLD -c CRTDSPF -p ENHDSP(*YES) RSTDSP(*YES) DFRWRT(*YES) AUT() OPTION(*EVENTF *SRC *LIST) TEXT('')
✓ DEPTS.FILE was created successfully!

=== Create Bound SQLRPGLE Program [DEPTS] in MERTESTBLD
CRTSQLRPGI srcstmf('/home/SANJULA/builds/ibmi-company_system/qrpglesrc/depts.pgm.sqlrpgle') OBJ(MERTESTBLD/DEPTS) COMMIT(*NONE ) OBJTYPE(*PGM) OPTION(*EVENTF) OUTPUT(*PRINT) TEXT('') TGTRLS() DBGVIEW(*SOURCE) RPGPOPT(*LVL2) COMPILEOPT(*TGTCSSID(*JOB) OPTIMIZE() INCDIR('qrpglef'))
✓ DEPTS.PGM was created successfully!

=== Creating SQL TABLE MERTESTBLD/EMPLOYEE from Sql statement [employee.table]
RUNSQLSTM srcstmf('/home/SANJULA/builds/ibmi-company_system/qddssrc/employee.table') DBGVIEW(*SOURCE) TGTRLS() OUTPUT(*PRINT) MARGINS(1024) COMMIT(*NONE )
✗ Failed to create EMPLOYEE.FILE!

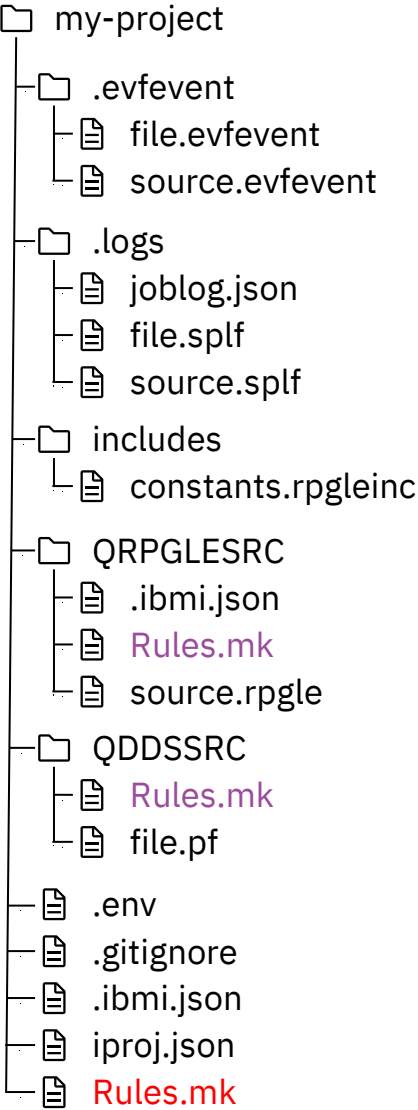
=== Creating DSPF [emps.dspf] in MERTESTBLD
/QOpenSys/pkgs/lib/bob/src/scripts/crtfrmstmf --ccsid *JOB -f /home/SANJULA/builds/ibmi-company_system/qddssrc/emps.dspf -o EMPS -l MERTESTBLD -c CRTDSPF -p ENHDSP(*YES) RSTDSP(*YES) DFRWRT(*YES) AUT() OPTION(*EVENTF *SRC *LIST) TEXT('')
✓ EMPS.FILE was created successfully!

=== Create Bound SQLRPGLE Program [EMPLOYEES] in MERTESTBLD
CRTSQLRPGI srcstmf('/home/SANJULA/builds/ibmi-company_system/qrpglesrc/employees.pgm.sqlrpgle') OBJ(MERTESTBLD/EMPLOYEES) COMMIT(*NONE ) OBJTYPE(*PGM) OPTION(*EVENTF) OUTPUT(*PRINT) TEXT('') TGTRLS() DBGVIEW(*SOURCE) RPGPOPT(*LVL2) COMPILEOPT(*TGTCSSID(*JOB) OPTIMIZE() INCDIR('qrpglef'))
✗ Failed to create EMPLOYEES.PGM!

=== Create Bound RPG Program [MYPGM] in MERTESTBLD
CRTBNDRPG srcstmf('/home/SANJULA/builds/ibmi-company_system/qrpglesrc/myrpgm.pgm.rpgle') PGM(MERTESTBLD/MYPGM) TGTCSSID(*JOB) DBGVIEW(*ALL) OPTION(*EVENTF) TEXT('') INCDIR('qrpglef')
✓ MYPGM.PGM was created successfully!

make: warning: Clock skew detected. Your build may be incomplete.
Objects:      2 failed 5 succeed 7 total
> Failed objects:  EMPLOYEE.FILE EMPLOYEES.PGM
Build Completed!
```

Defining targets with Rules.mk



```
M Rules.mk  X
M Rules.mk
1  SUBDIRS = qrpglesrc qddssrc
```

Rules.mk in project root

Declare subdirectories to be built

```
M Rules.mk  X
M Rules.mk
1  FVAT.SRVPGM: fvat.bnd VAT300.MODULE
2  FVAT.SRVPGM: TEXT = Functions VAT
3  FVAT.SRVPGM: private TEXT = Functions VAT
4
5  VAT300.MODULE: vat300.rpgle QPROTOSRC/vat.rpgleinc VATDEF.FILE
6  VAT300.MODULE: private TEXT := bound into FVAT.SRVPGM
7  VAT300.MODULE: private DBGVIEW ::= *SOURCE
8
9  VATDEF.FILE: vatdef.pf SAMREF.FILE
```

Rules.mk in subdirectories

Makefile with list of objects to be built and from which source files

Using variables in Rules.mk

Declare variables to override compile settings for multiple targets

M Rules.mk U X

SAMPLE > M Rules.mk

```
1  PRO200.MODULE: private TGTRLS := *PRV
2  PRO200.MODULE: PRO200.RPGLE
3
4  VAT.MODULE: private TGTRLS := *PRV
5  VAT.MODULE: VAT.RPGLE
```

M Rules.mk U X

SAMPLE > M Rules.mk

```
1  PROJECT_TGTRLS := *PRV
2
3  PRO200.MODULE: private TGTRLS := $(PROJECT_TGTRLS)
4  PRO200.MODULE: PRO200.RPGLE
5
6  VAT.MODULE: private TGTRLS := $(PROJECT_TGTRLS)
7  VAT.MODULE: VAT.RPGLE
```

Wildcarding in Rules.mk

Use wildcarding when you have multiple sources that create objects of the same type

M Rules.mk U X

SAMPLE > M Rules.mk

```
1  FILE_TEXT := SAMPLE1
2  MOD_TEXT  := SAMPLE2
3
4  TEST1.FILE: private TEXT := $(FILE_TEXT)
5  TEST1.FILE: TEST1.TABLE DEP1.FILE DEP2.FILE
6
7  TEST2.FILE: private TEXT := $(FILE_TEXT)
8  TEST2.FILE: TEST2.TABLE DEP1.FILE DEP2.FILE
9
10 TEST1.MODULE: private TEXT := $(MOD_TEXT)
11 TEST1.MODULE: TEST1.RPGLE
```

M Rules.mk U X

SAMPLE > M Rules.mk

```
1  FILE_TEXT := SAMPLE1
2  MOD_TEXT  := SAMPLE2
3
4  %.FILE: private TEXT := $(FILE_TEXT)
5  %.MODULE: private TEXT := $(MOD_TEXT)
6
7  %.FILE: %.TABLE DEP1.FILE DEP2.FILE
8  %.MODULE: %.RPGLE
```

Build and Compile Process

Initialization and Migration

Command	Description
makei init	Create iproj.json
makei cvtsrcpf	Convert QSYS members to Unicode IFS stream files

Building

Command	Description
makei build	Build the entire project
makei b -t <object>	Build target object
makei b -d <directory>	Build all objects in the specified directory (based on Rules.mk)

Compiling

Command	Description
makei compile -f <stream file>	Compile target object of specified stream file
makei compile -files file1: file2: ...	Compile target objects of all specified stream files

Ins and Outs of IBM i Project Explorer

Overview

The ultimate tool for local development on IBM i!

Set variables

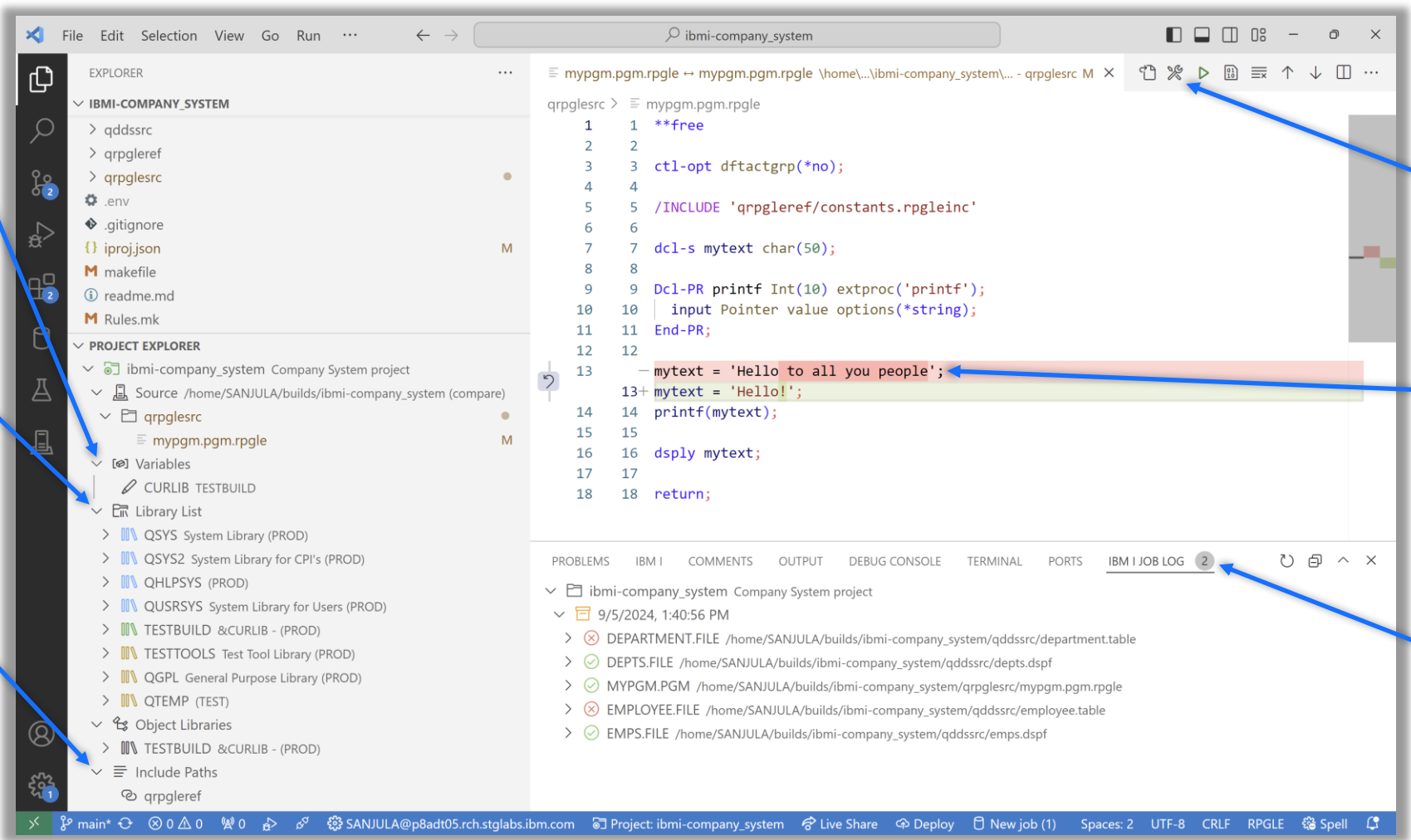
Manage library list

Modify include paths

Build and Compile

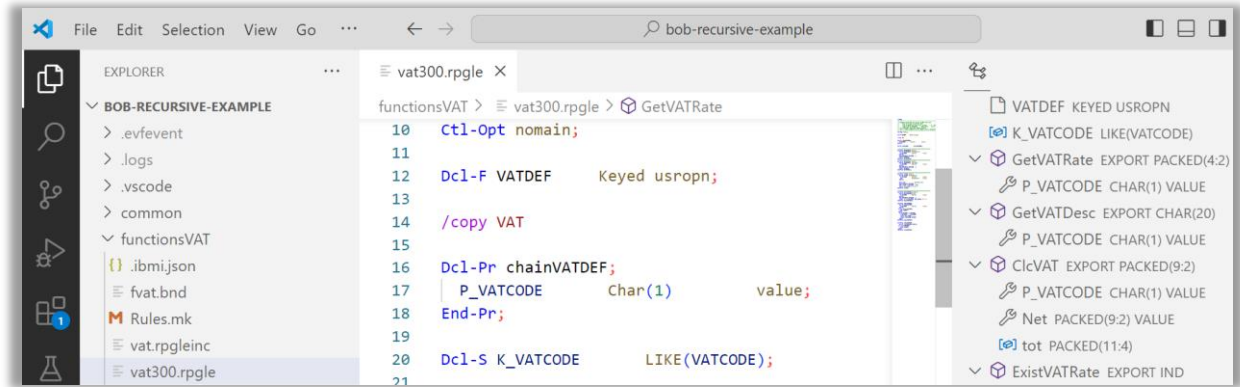
Local source vs. IFS source

View job logs

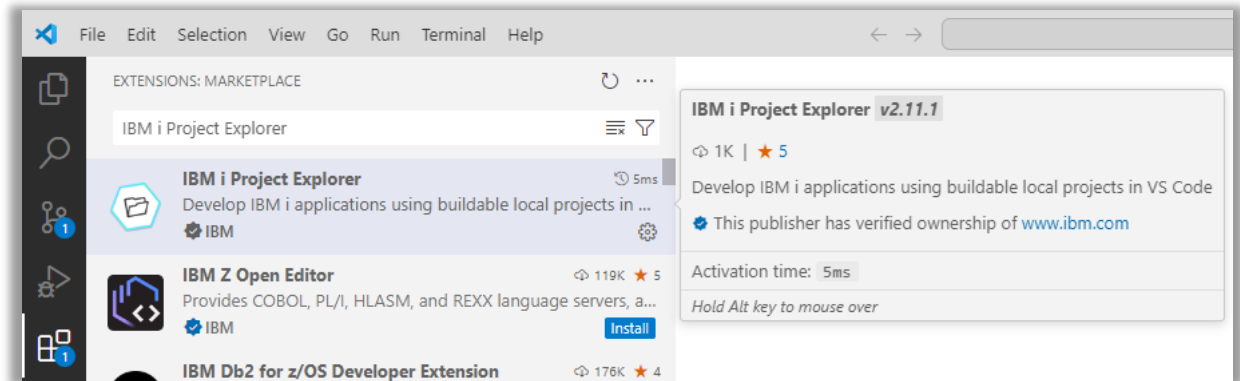


Installation

1 *Download
Visual Studio Code*



2 *Download VS Code extensions
IBM i Project Explorer,
Source Orbit and Code for IBM i*

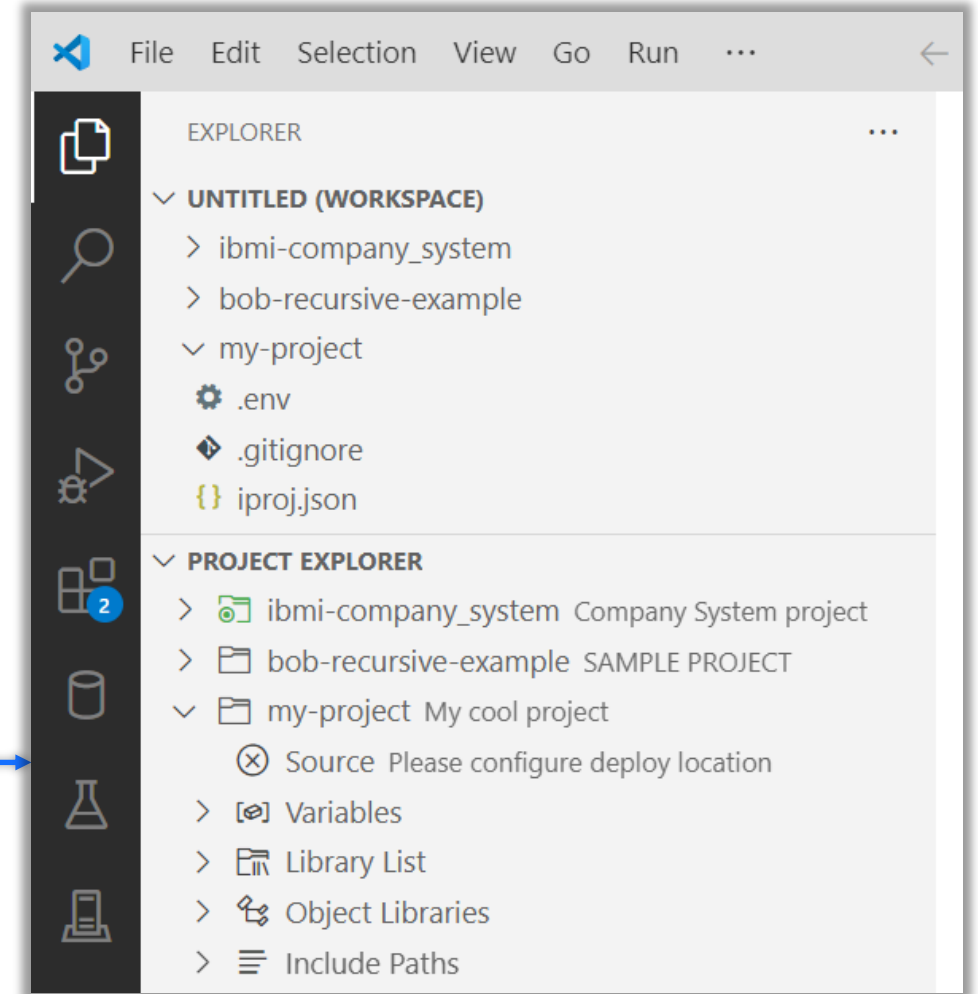
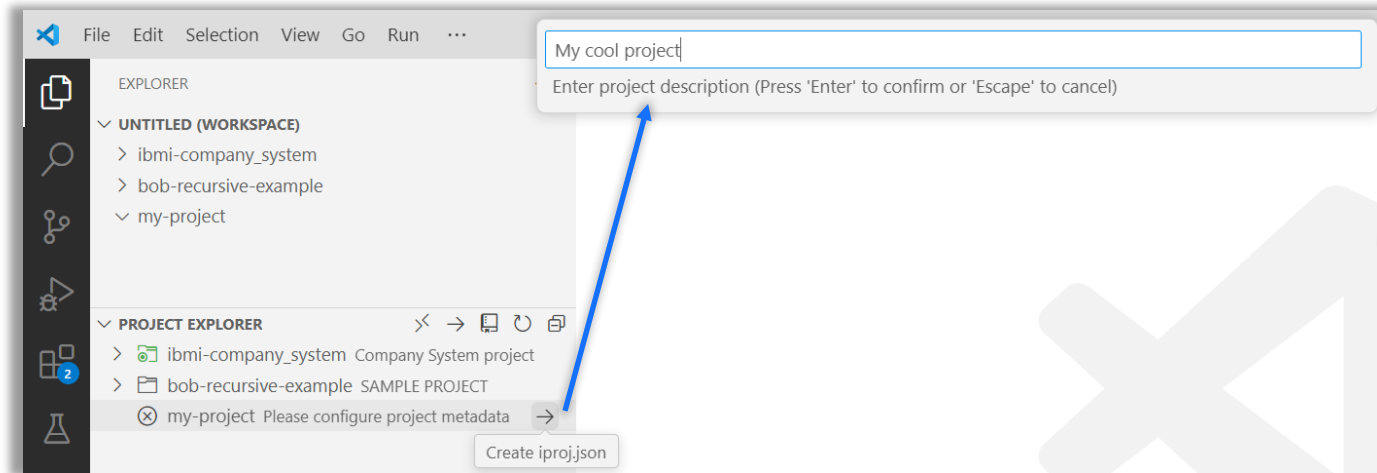


3 *Run
yum install bob
on IBM i*



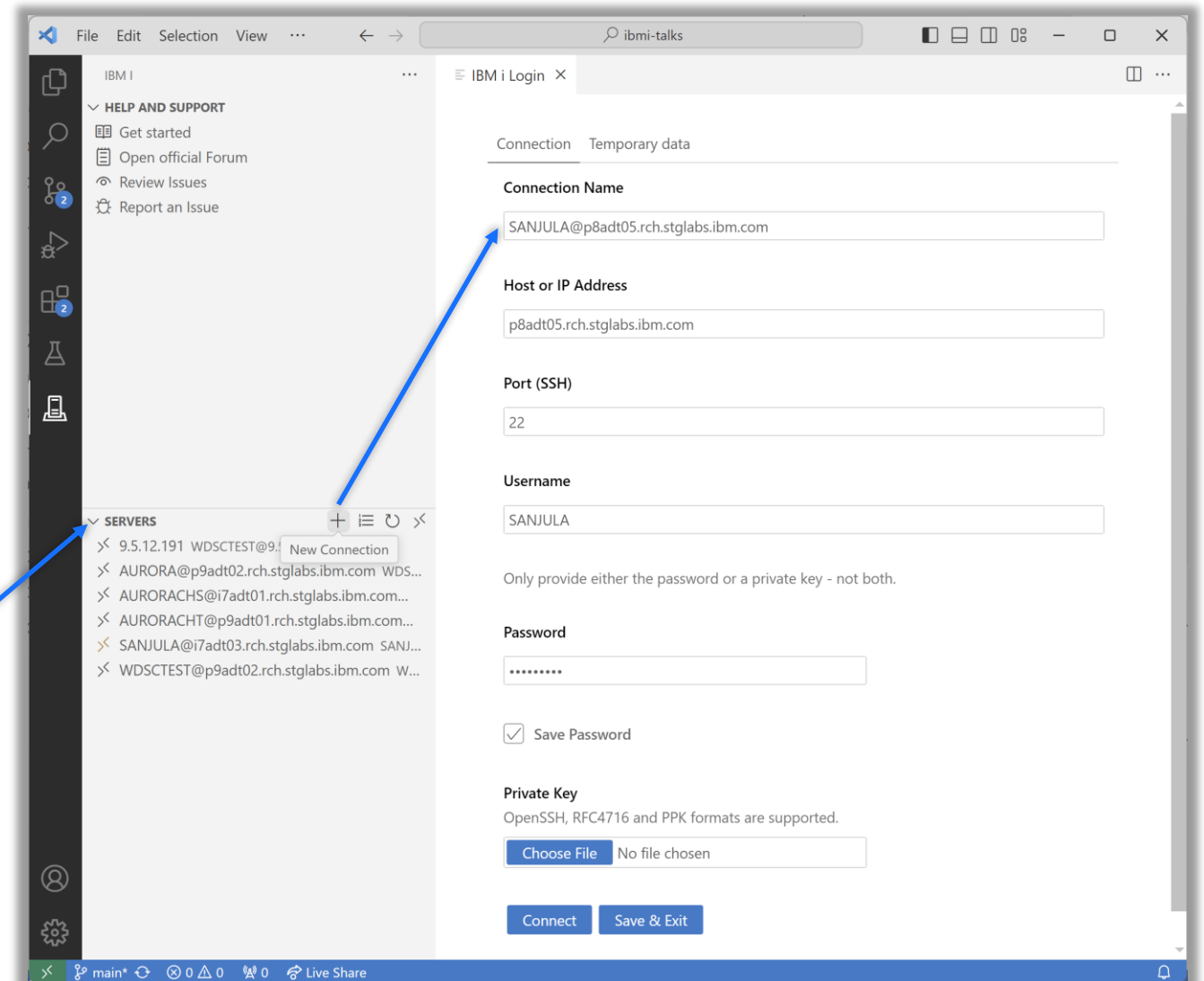
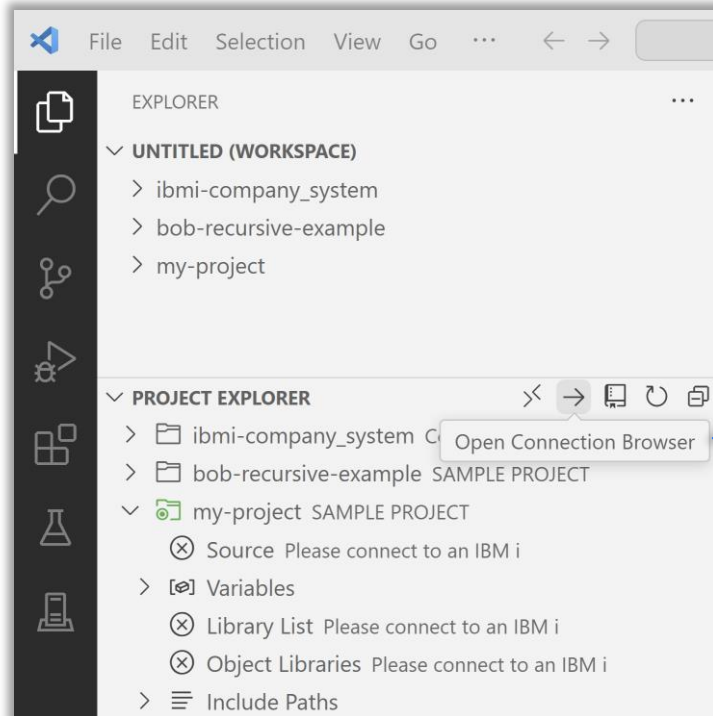
Create a New Project

- Create and open a folder for your project
- Create an `iproj.json`
- Set the project description



Connect to an IBM i

- Open the *Connection Browser* from Project Explorer
- Create new IBM i connection from the *Server* view



Migrate Source from QSYS

CVTSRCPF
from BOB



QSYS members in
source physical files



Properly encoded,
terminated, and named
source files in an IFS
directory



Download to local
project

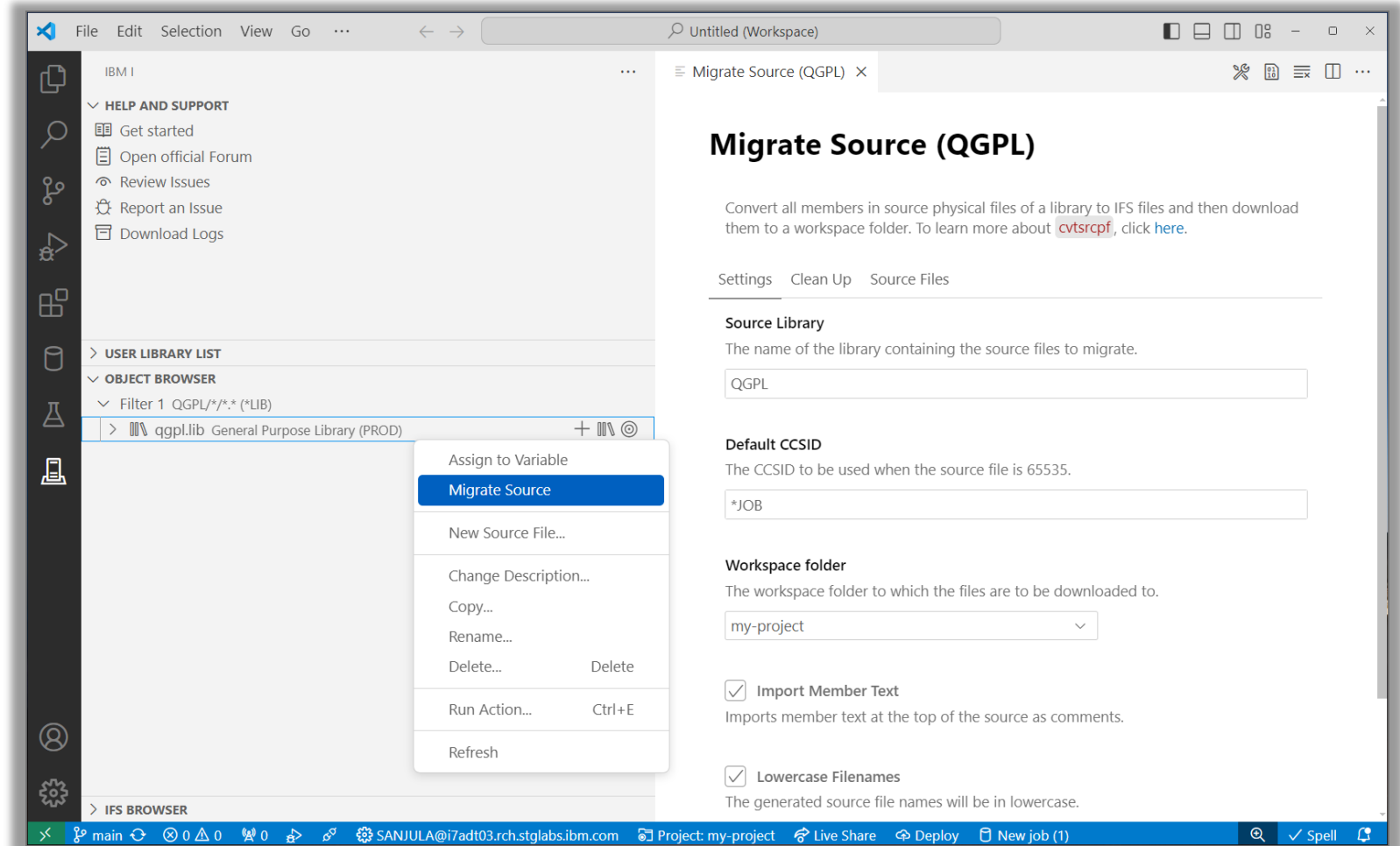


Rename extensions



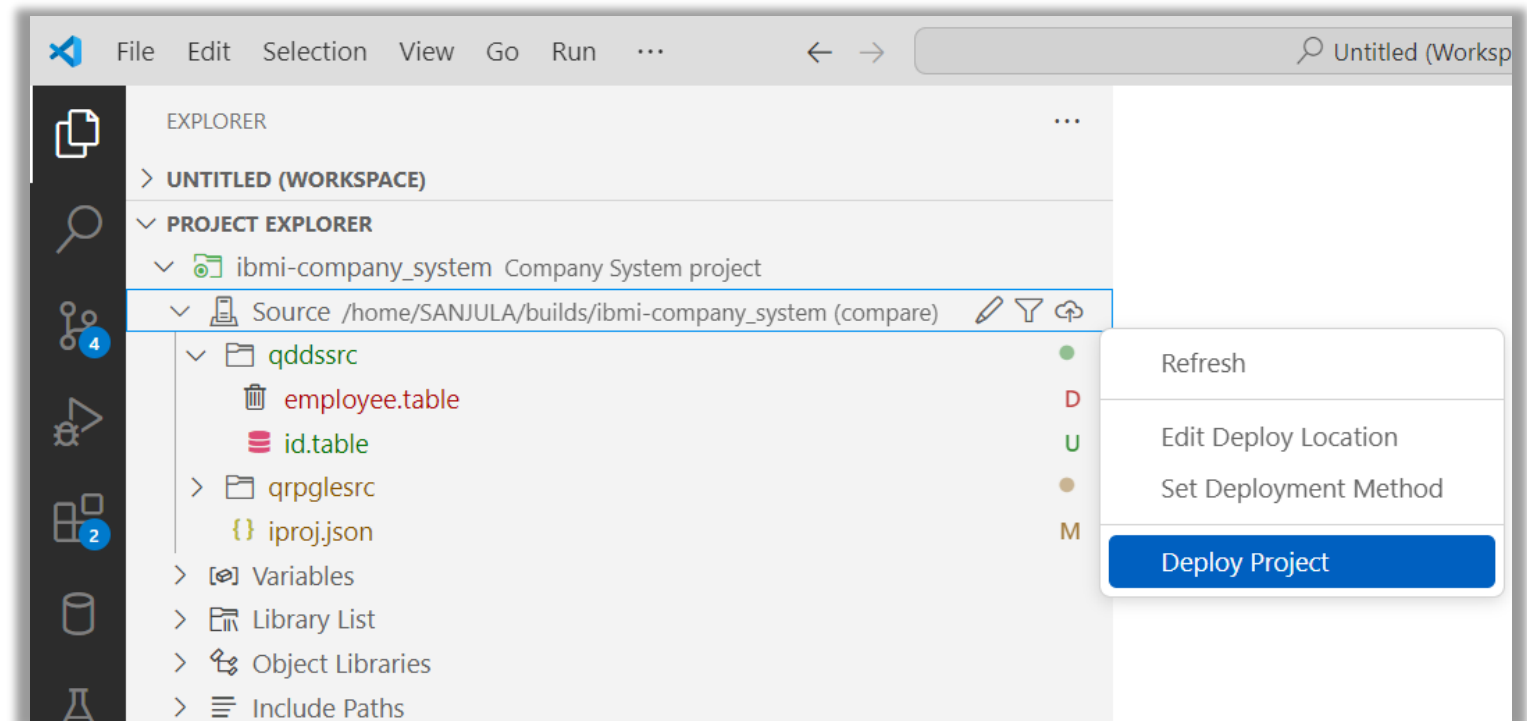
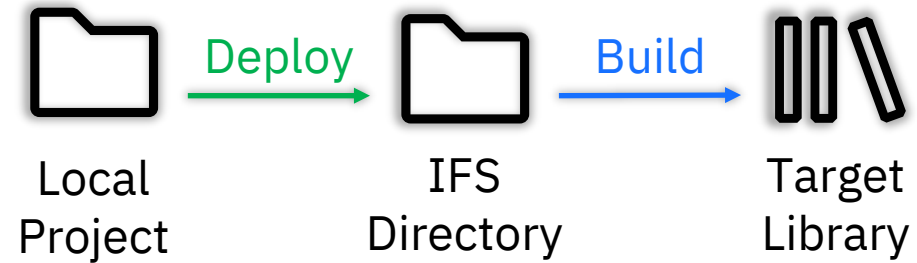
Convert includes/copy
directives to Unix style
paths

Source Orbit



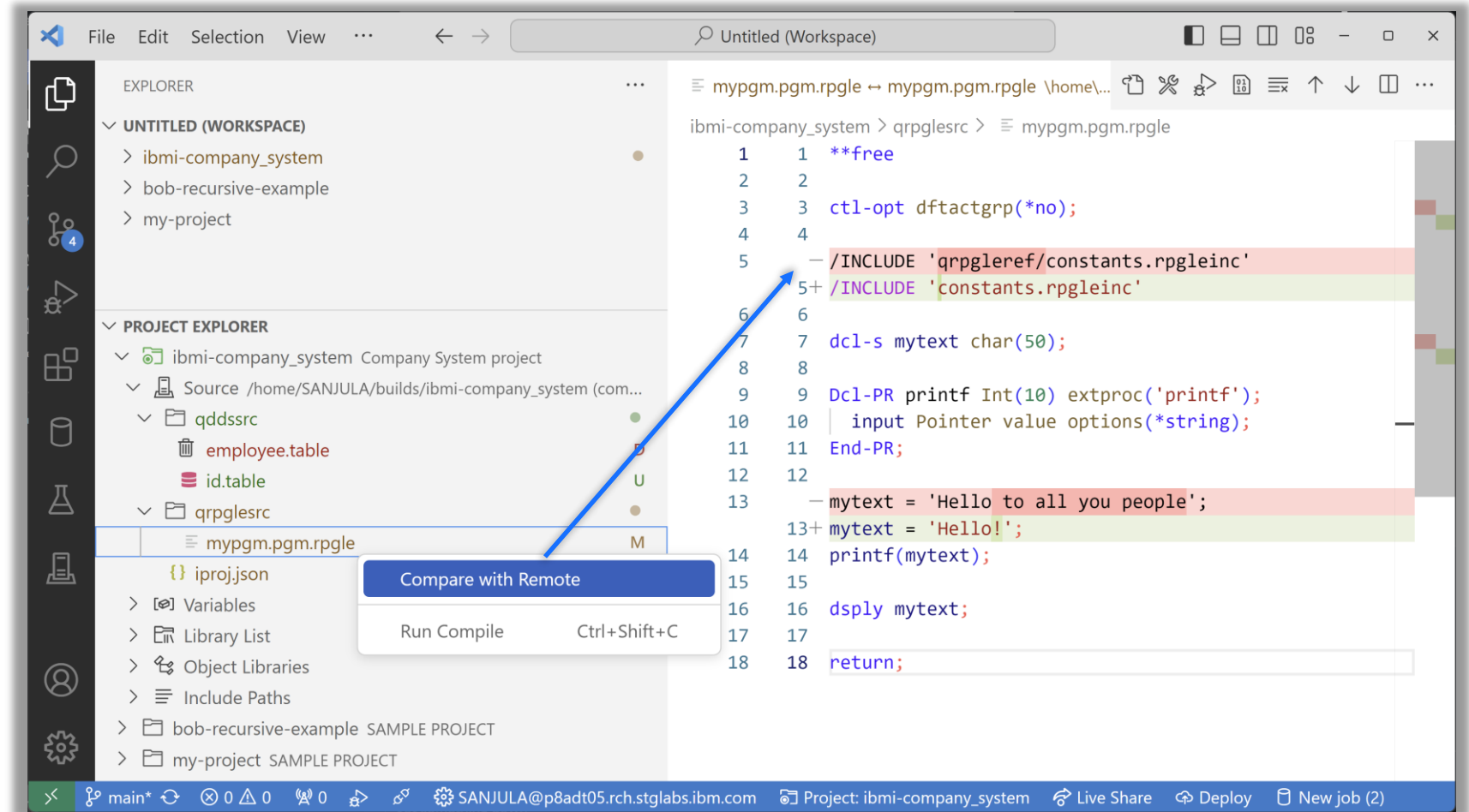
Deployment

- Set deploy location
 - Where source gets uploaded to
 - Typically set one
 - Each developer gets a unique location
 - Each repository gets a unique location
- Set deployment method
 - Compare (typically the safest)
 - Changes (typically the fastest)
 - Working Changes
 - Staged Changes
 - All
- Deploy project
 - Moves files to deploy location based on deployment method



Visualize Local vs. Remote Source Files

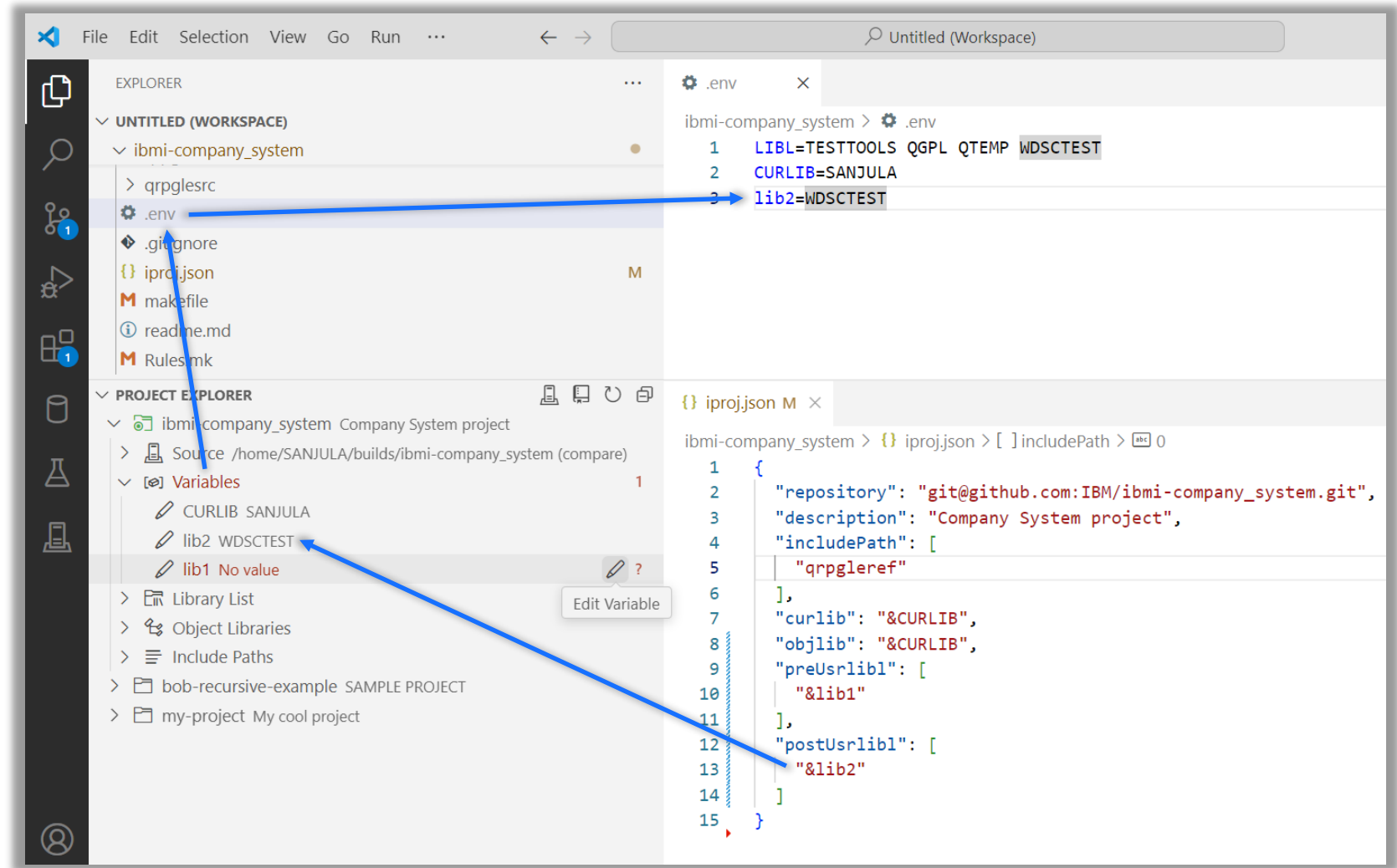
- Visualize, compare, and deploy your local source files to the deploy location in the IFS
- Track file changes (added, modified, deleted, etc.)
- Compare local file content with remote IFS



Work with Variables

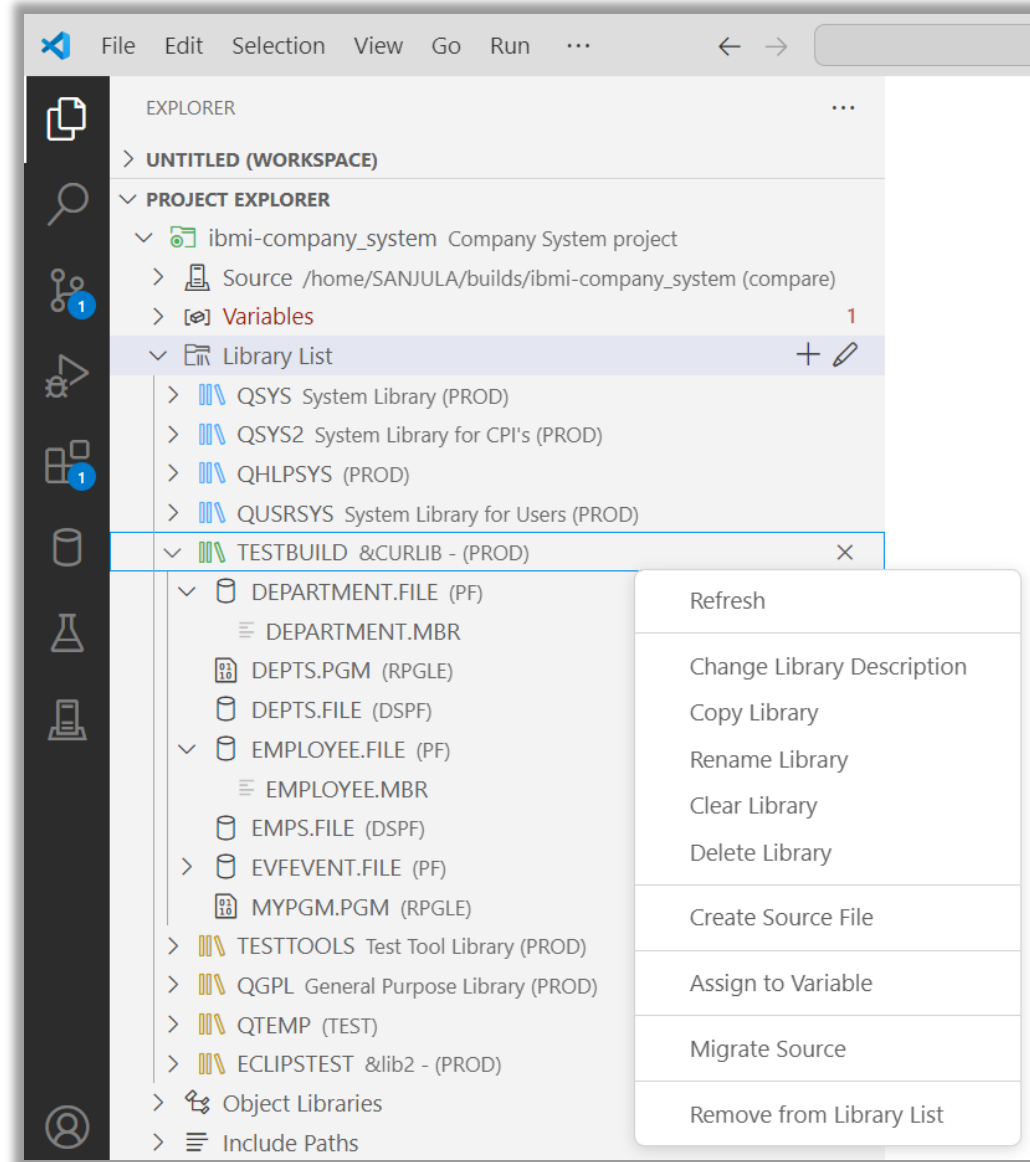
- Reusable project definition that can be used by multiple developers or in automated builds
- View and set variables (for libraries, include paths, or build/compile commands)
- Browse for libraries and assign values to variables
- Configure hardcoded values as variables

Do not push .env file to Git!



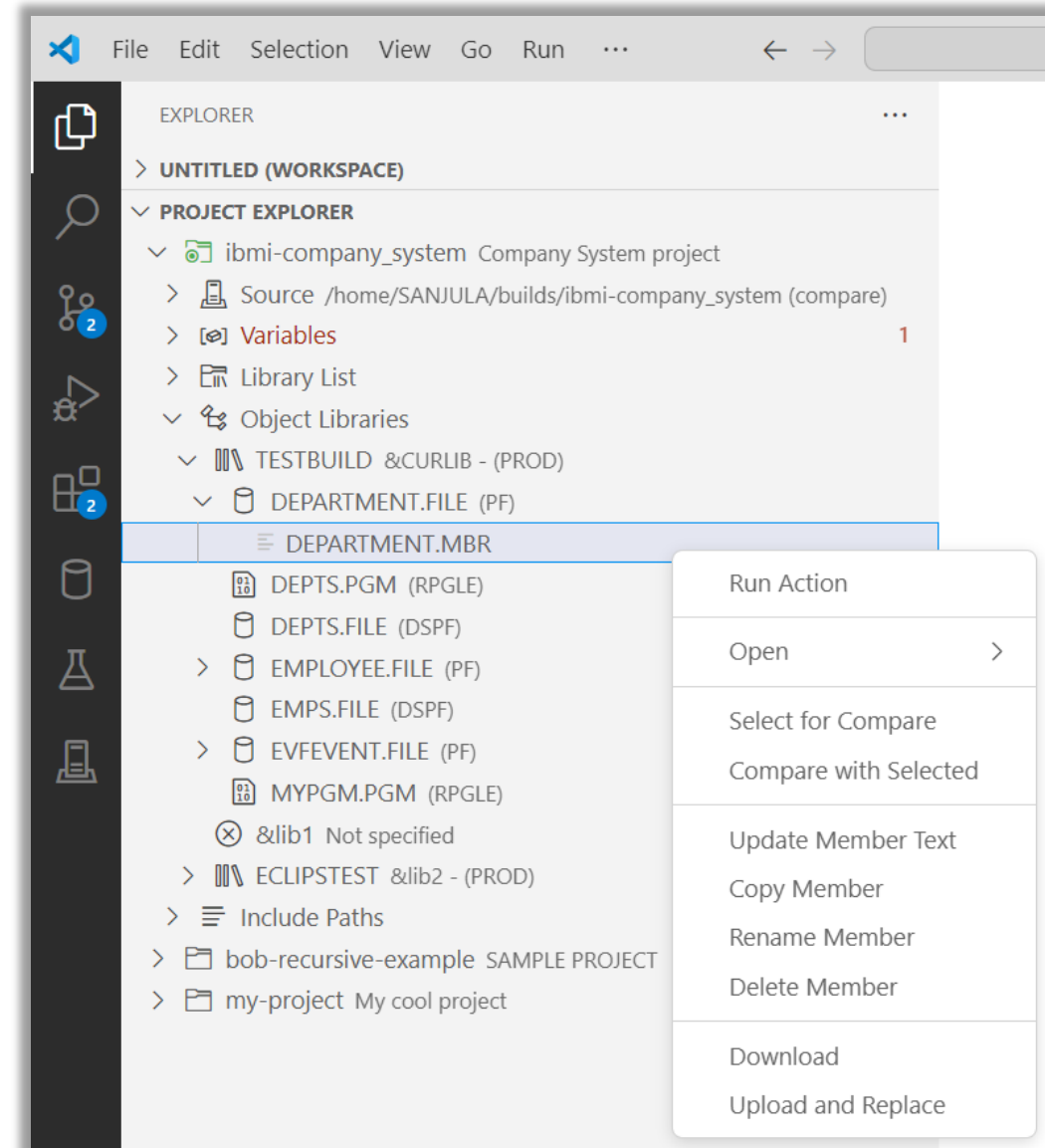
Manage the Library List

- Project's library list is a composition of your user profile's library list (from JOBD) + set of project specific libraries
- Add to beginning/end of library list (*preUsrlibl* and *postUsrlibl*) and set current library (*curlib* in *iproj.json*)
- Reorder library list
- Browse objects and members
- Manage libraries, objects, and members



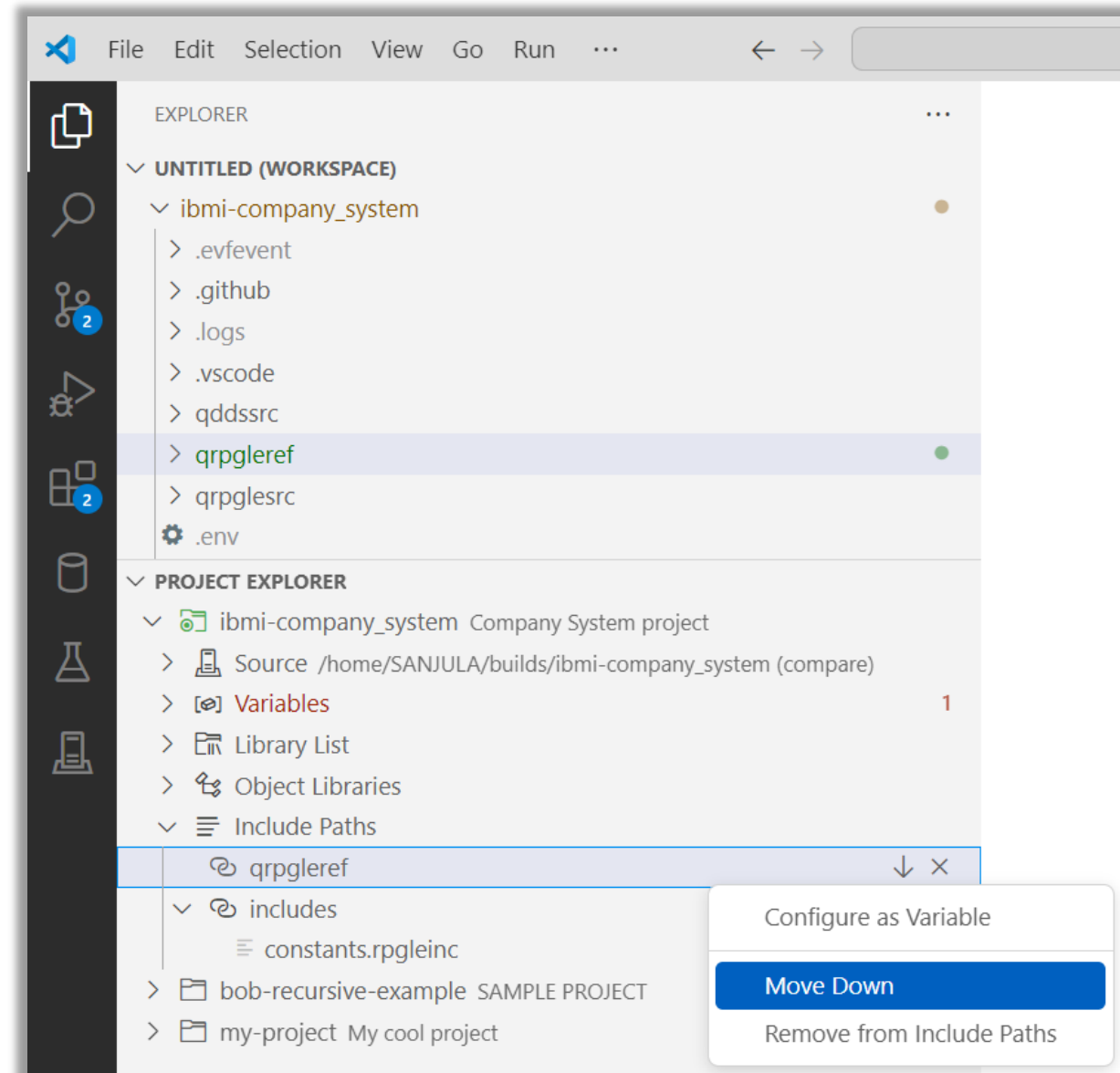
Browse Object Libraries

- The place for developers to easily see, debug, and manipulate the results of your build
- Another place to manage libraries in iproj.json (*curlib*, *objlib*, *preUsrlibl*, *postUsrlibl*)
- Manage libraries, objects, and members



Manage Include Paths

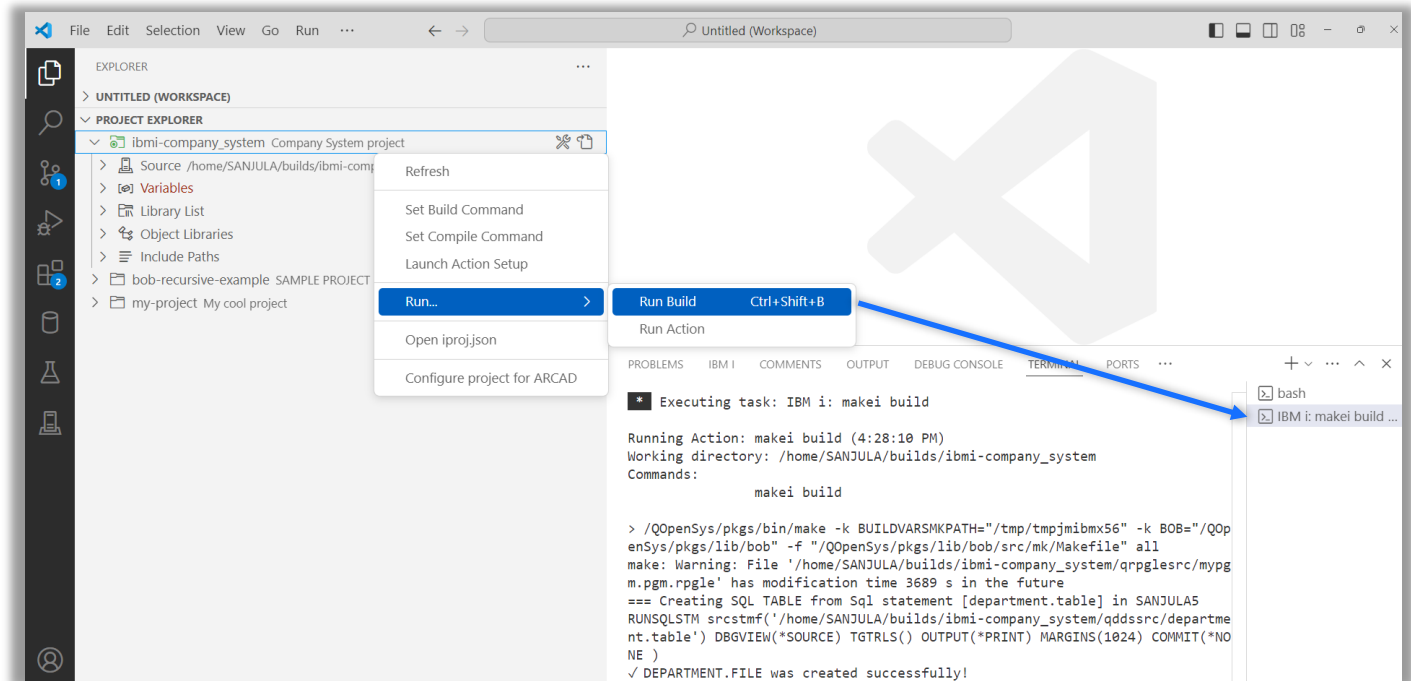
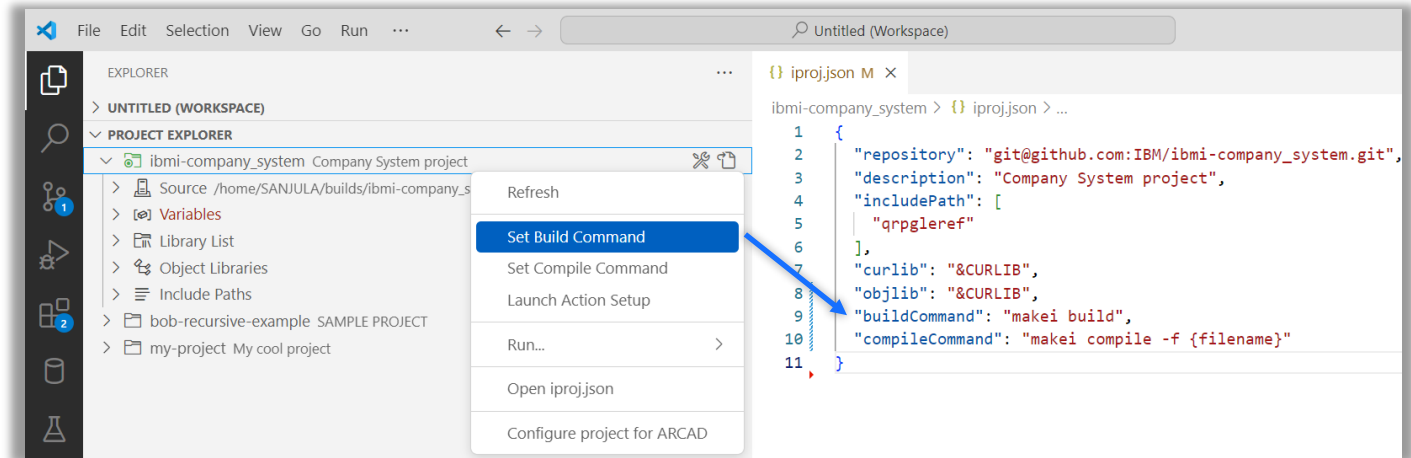
- Self-contained projects should know where to find includes within the project
- Add, remove, and reorder include paths
- Visualize if includes resolve locally or to remote IFS



Build and Compile

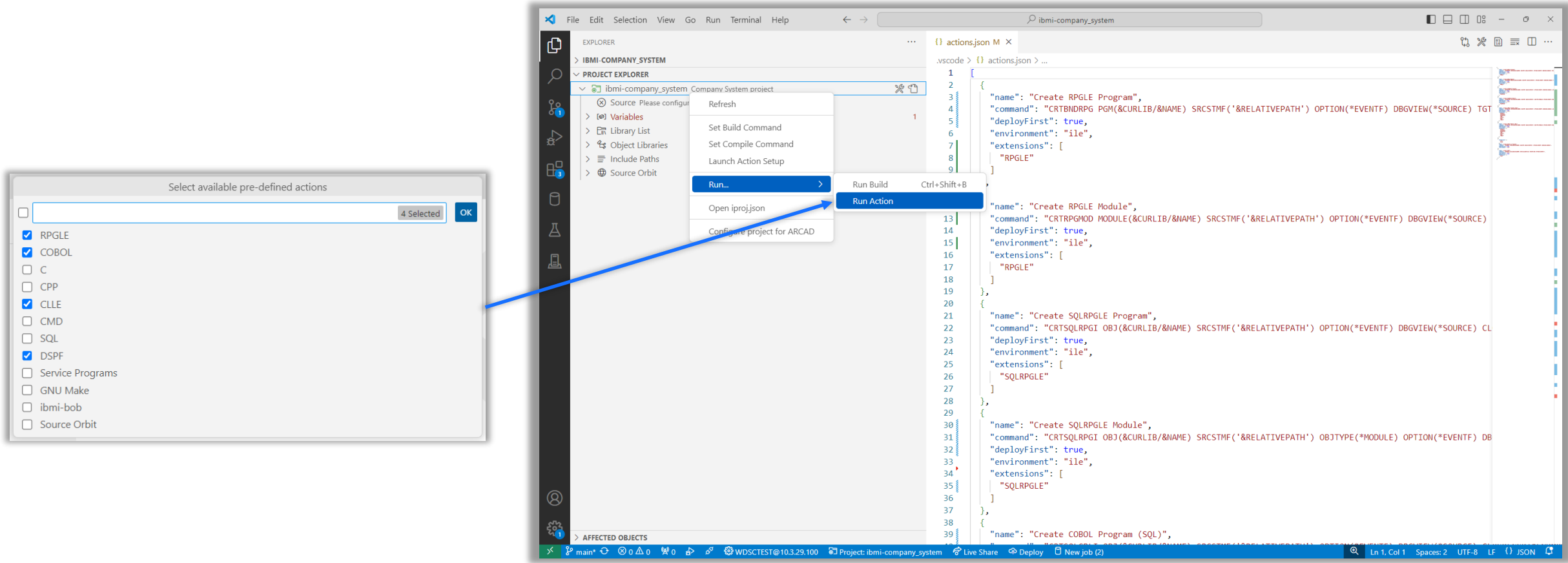
- 1 **Deploy**
↓
- 2 **Run build or compile command**
(any build framework)
↓
- 3 **Download logs and event files**

- Building
 - Set build command
 - Run Build (*Ctrl+Shift+b* or *Cmd+Shift+b*)
- Compiling
 - Set compile command
 - Run compile (*Ctrl+Shift+c* or *Cmd+Shift+c*)
 - On active editor
 - On file or directory in File Explorer
 - On file or directory in Source



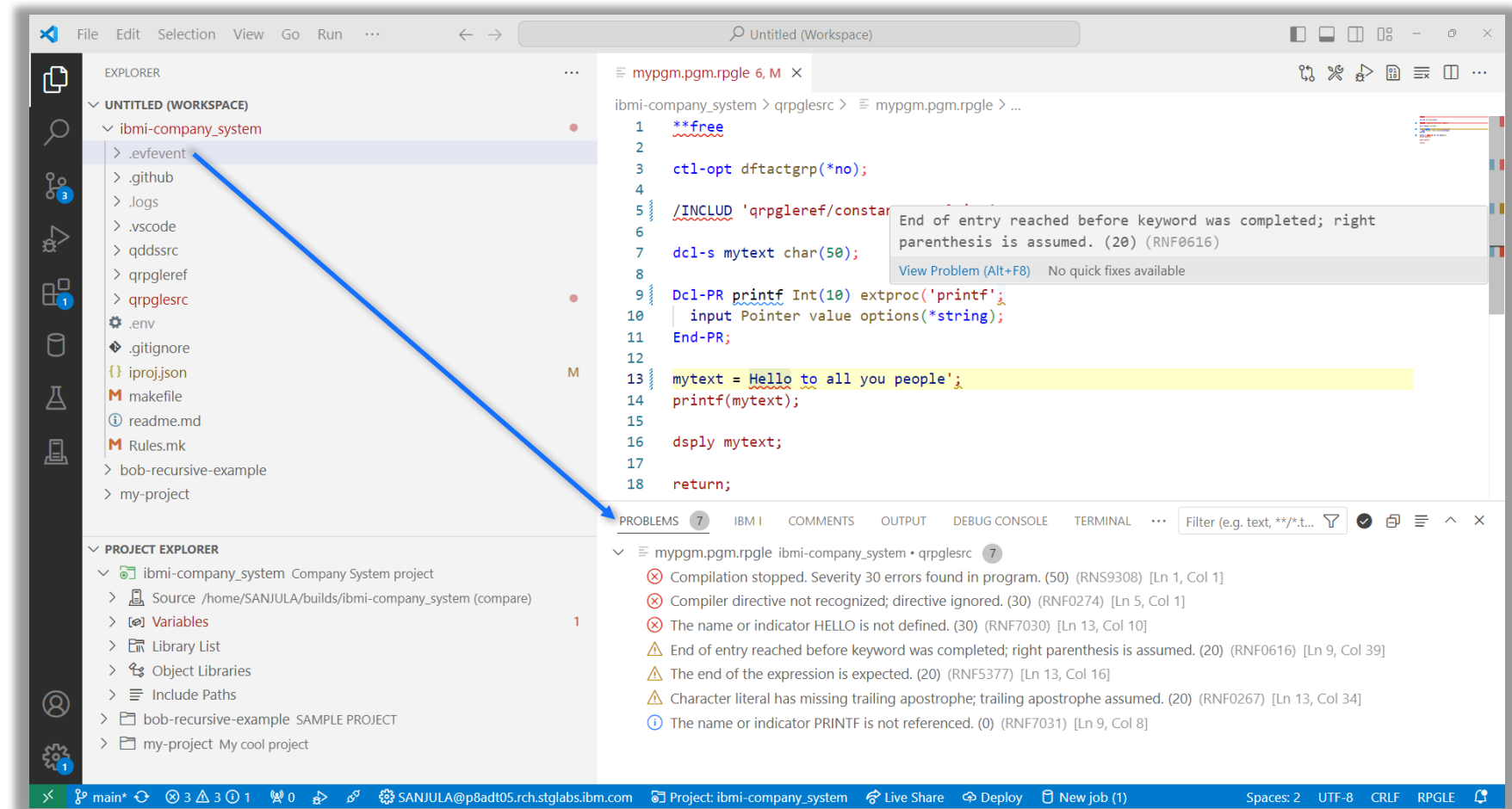
Run Actions

IBM i Project Explorer also supports running Code for IBM i's custom workspace actions



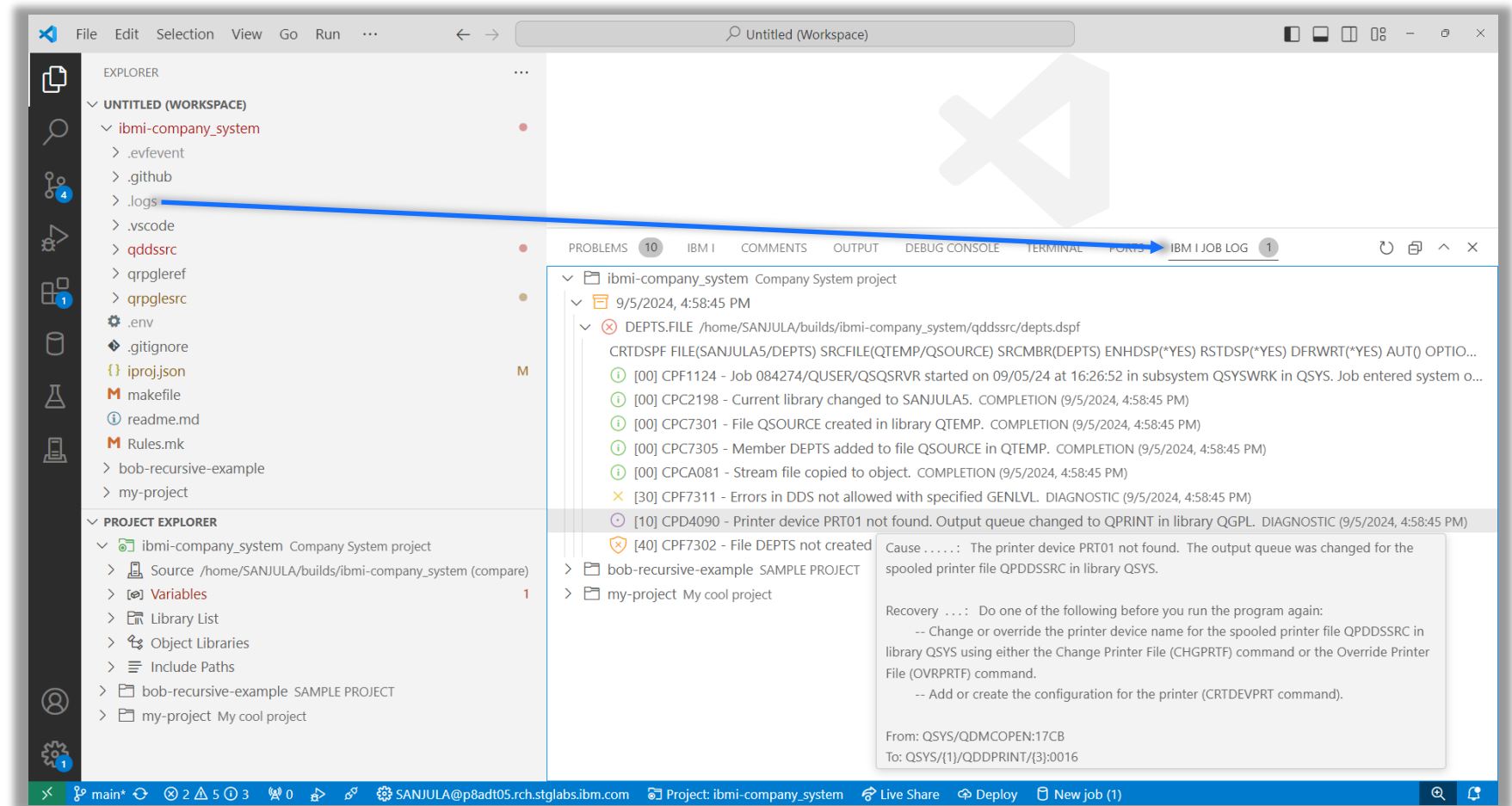
View Diagnostics

- Visualize build or compile diagnostics in the Problems view
- Evfevent file diagnostics are dumped in a .evfevent directory after a build or compile
- Diagnostics are also rendered inline in the source file



View Job Logs

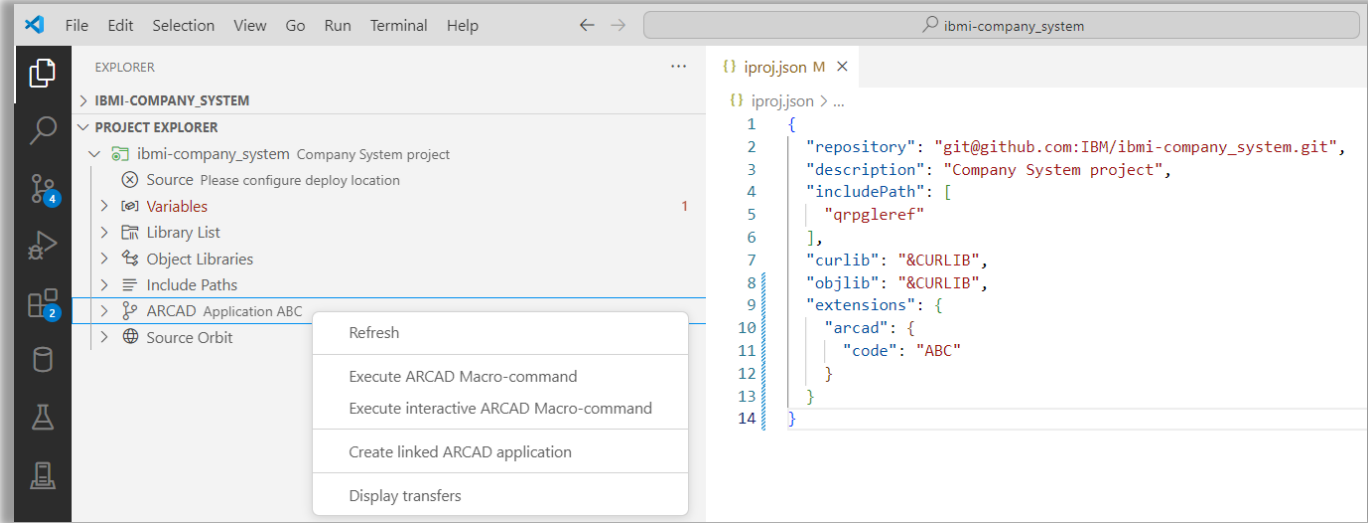
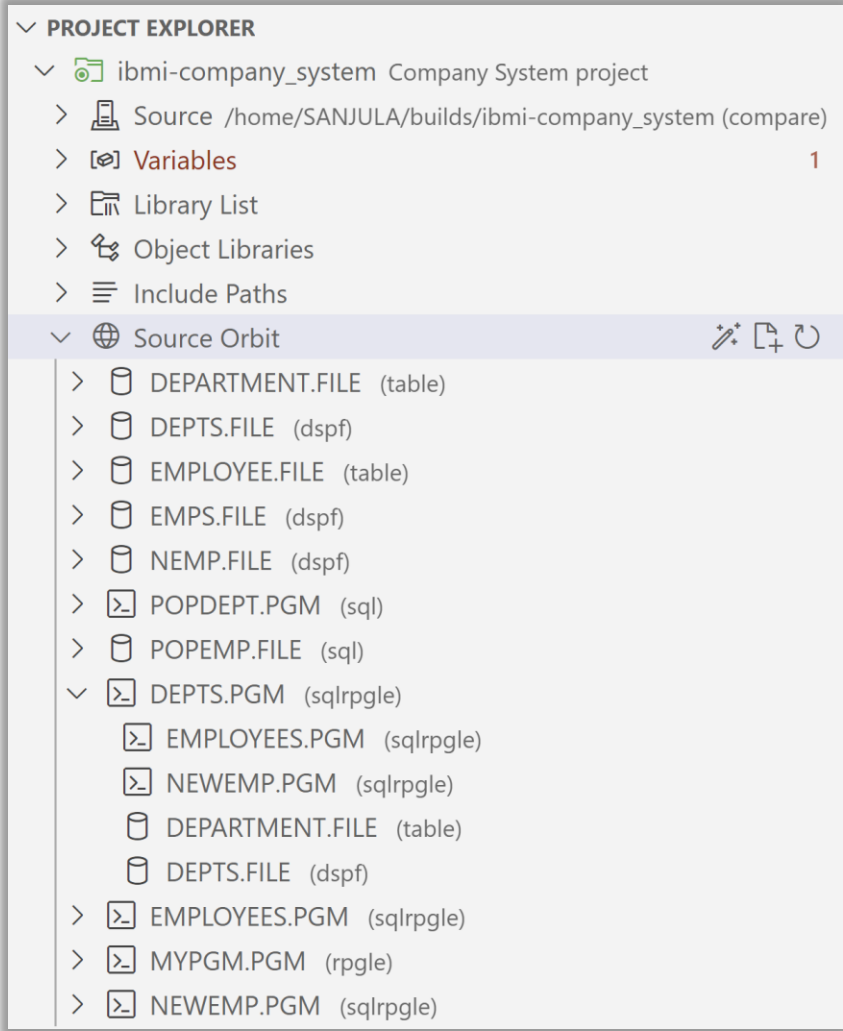
- Visualize and manage anything that could be seen in an IBM i job log including second level help
- Job log and spool files are dumped in .logs directory after a build or compile
- Track up to 10 of the previous logs in memory
- Organized by the ILE objects being built
- Filter by failed objects or severity



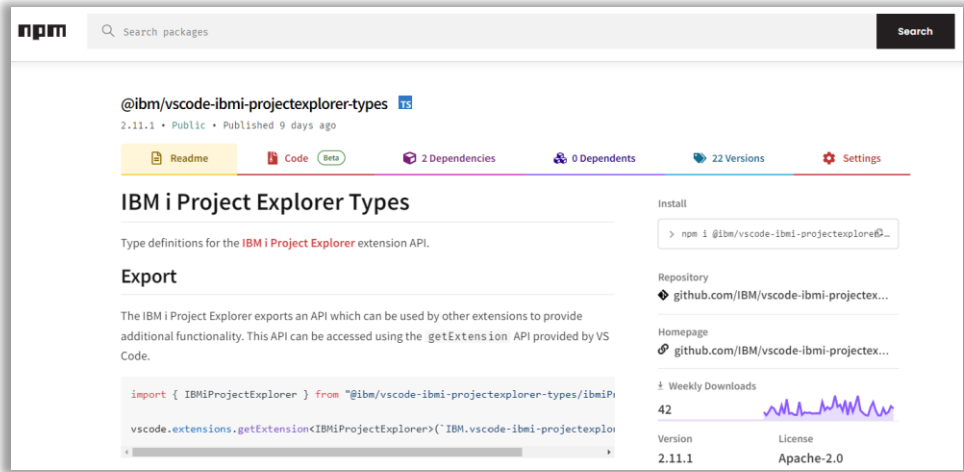
Integration



ARCAD-Elias



*What can you
integrate with
IBM i Project
Explorer's API?*



Any Questions?

Important Links

IBM i Project Explorer

- VS Code Marketplace <https://marketplace.visualstudio.com/items?itemName=IBM.vscode-ibmi-projectexplorer>
- Documentation <https://ibm.github.io/vscode-ibmi-projectexplorer/#/>
- GitHub Repository <https://github.com/IBM/vscode-ibmi-projectexplorer>
- API <https://www.npmjs.com/package/@ibm/vscode-ibmi-projectexplorer-types>

Bob

- Documentation <https://ibm.github.io/ibmi-bob/#/>
- GitHub Repository <https://github.com/IBM/ibmi-bob>

Code for IBM i

- VS Code Marketplace <https://marketplace.visualstudio.com/items?itemName=HalcyonTechLtd.code-for-ibmi>
- Documentation <https://codefori.github.io/docs/#/>
- GitHub Repository <https://github.com/codefori/vscode-ibmi>
- API <https://www.npmjs.com/package/@halcyontech/vscode-ibmi-types>

IBM i