Raincode 360 User Guide

Version 4.0.0

14-06-2026

Raincode 360 topics

1. Developer Experience
2. Raincode Legacy Compilers
3. Raincode Console
4. Raincode Batch: JCL Interpreter
5. zBridge
6. CICS Emulation
7. Caching
8. IMSql
9. Cloud Native Deployment
10. Expert mode

RAINCODE 360
Advanced Showcase

What is Raincode 360?

- An advanced showcase
- A non-trivial application
  - Showcasing advanced features of the Raincode product suite
- Includes:
  - Full source code and development projects
  - Data
  - Tooling
  - This presentation
- Deployed as an Azure Resource Manager Template
- Demos are hosted in a GitLab repository (https://gitlab.phidani.be/raincodepublic/raincode-360.git)
- Raincode 360 demos are located at C:\Raincode360\Repositories\Demos
- Raincode Product Manuals are available at https://www.raincode.com/docs/

RAINCODE 360
Advanced Showcase

List of installed resources on Raincode 360

- Raincode product suite
  - Raincode Legacy Compilers
  - Raincode Visual Studio Integration
  - Raincode QIX Emulator
  - Raincode Batch
  - Raincode IMSql
  - Raincode RadaR
  - Raincode zTrieve
- System tools
  - PowerShell
  - GNU Make
- Development tools
  - Visual Studio 2026 Community Edition
  - Git for Windows
- Terminal emulators
  - wc3270
- PowerShell Modules
  - Az
  - SqlServer
  - VSSetup
- Analysis tools
  - Raincode Insight
  - DB Browser for SQLite
  - SQLite ODBC
  - HxD hex editor
  - ILSpy .NET decompiler
  - Meld
  - SQL Server Management Studio

RAINCODE 360 Advanced Showcase

Raincode 360 Licensing

- Raincode 360 is licensed under the Raincode 360 Software License Agreement.
  - The latest version of the license is available at: https://www.raincode.com/legal/TLA.pdf
  - A local copy is available on the VM at: C:\Raincode360\Cache\TLA.pdf

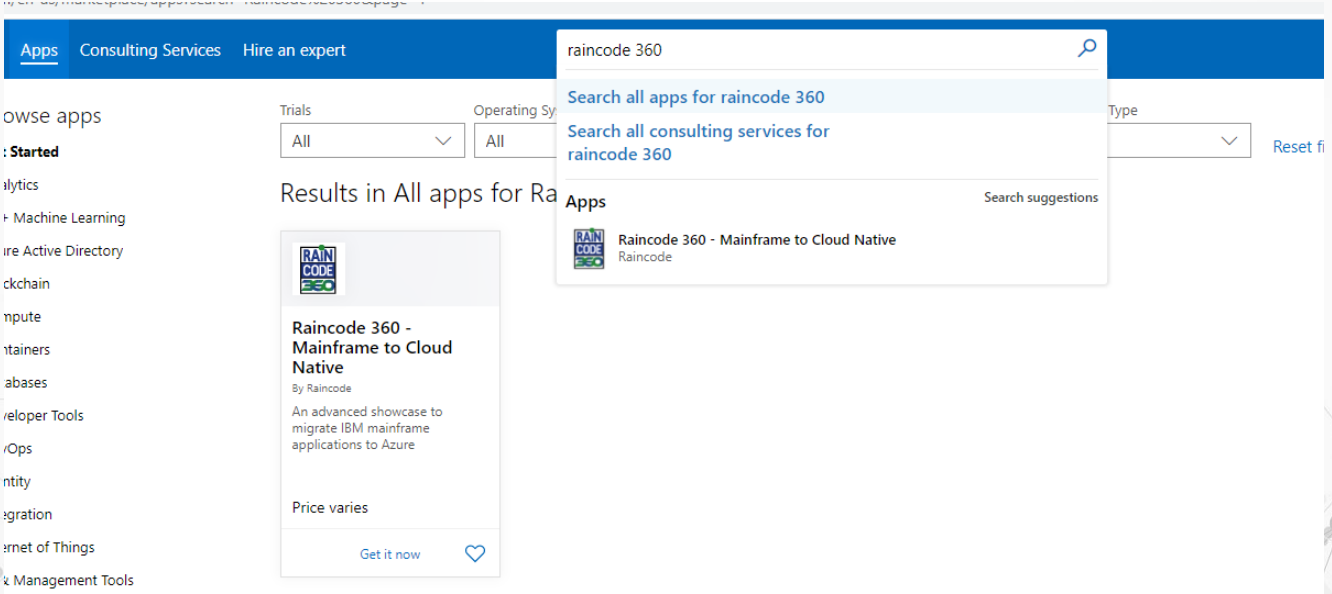
- Raincode 360 includes third-party products which are covered by their own respective licenses.
  - Third-party licenses are available on the VM at: C:\Raincode360\Cache\licenses.html

Raincode 360 on Azure Marketplace

RAINCODE 360
Advanced Showcase

Steps to download Raincode 360 from Marketplace

- Step 1: Login to the Azure Marketplace (https://azuremarketplace.microsoft.com/)
- Step 2: Search for **Raincode 360**

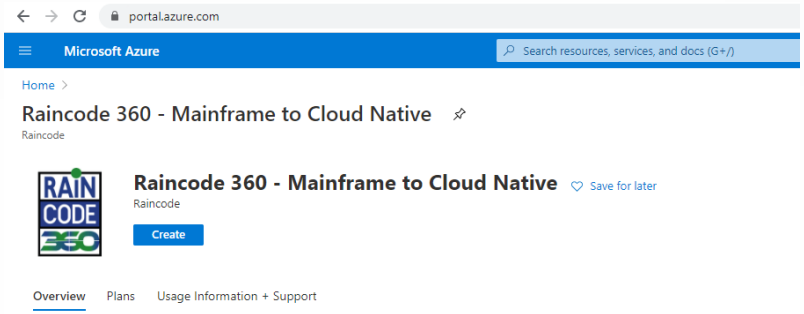


- Step 3: Click **Get it now** and fill in the required details. It will redirect you to the Azure Portal.

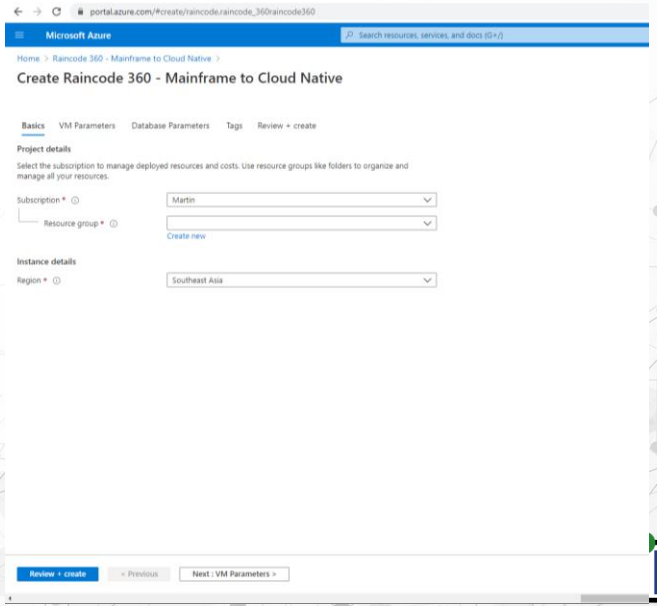
RAINCODE 360
Advanced Showcase

Steps to create Raincode 360 from Marketplace

- Step 4: Click Create



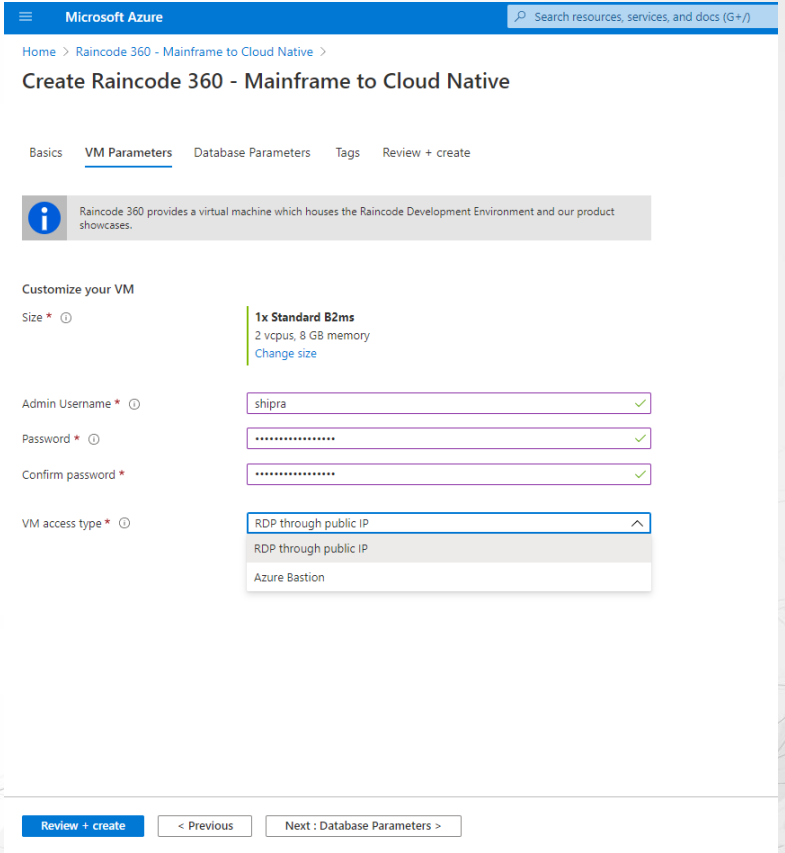
- Step 5: Enter Resource group name and choose Region



Advanced Showcase

Steps to create Raincode 360 from Marketplace

- Step 6: Enter the details:
  - Size
  - Username and password
  - VM access type:
    - RDP through public IP: the VM is made available on a public IP address with open access only for the RDP protocol (TCP port 3389).
    - Bastion: the VM is secured using Azure Bastion. The VM is not exposed on the public Internet and can only be accessed through the Azure portal. Allows fine-grained access control using Azure role assignments.

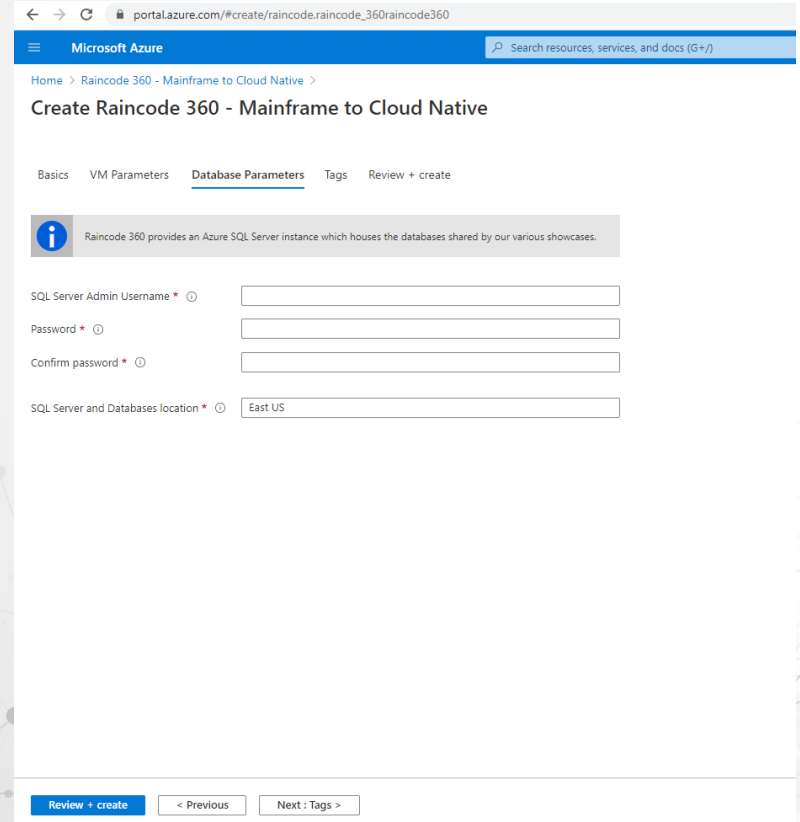


Note: Azure Bastion greatly improves security, but it also incurs supplementary resource costs. See Azure Bastion pricing for details.

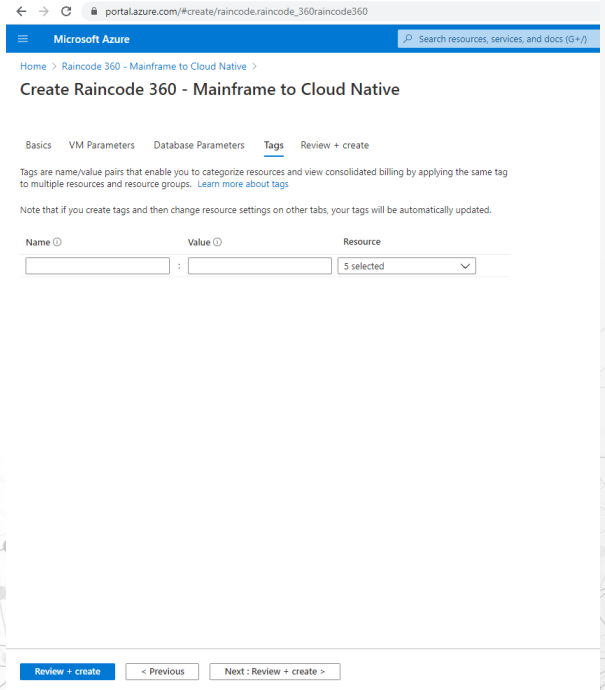
RAINCODE 360 Advanced Showcase

Steps to create Raincode 360 from Marketplace

- Step 7: Enter details:
  - SQL server admin username and password



- Step 8: Enter tags (optional):
  - Name
  - Value
  - Resource types



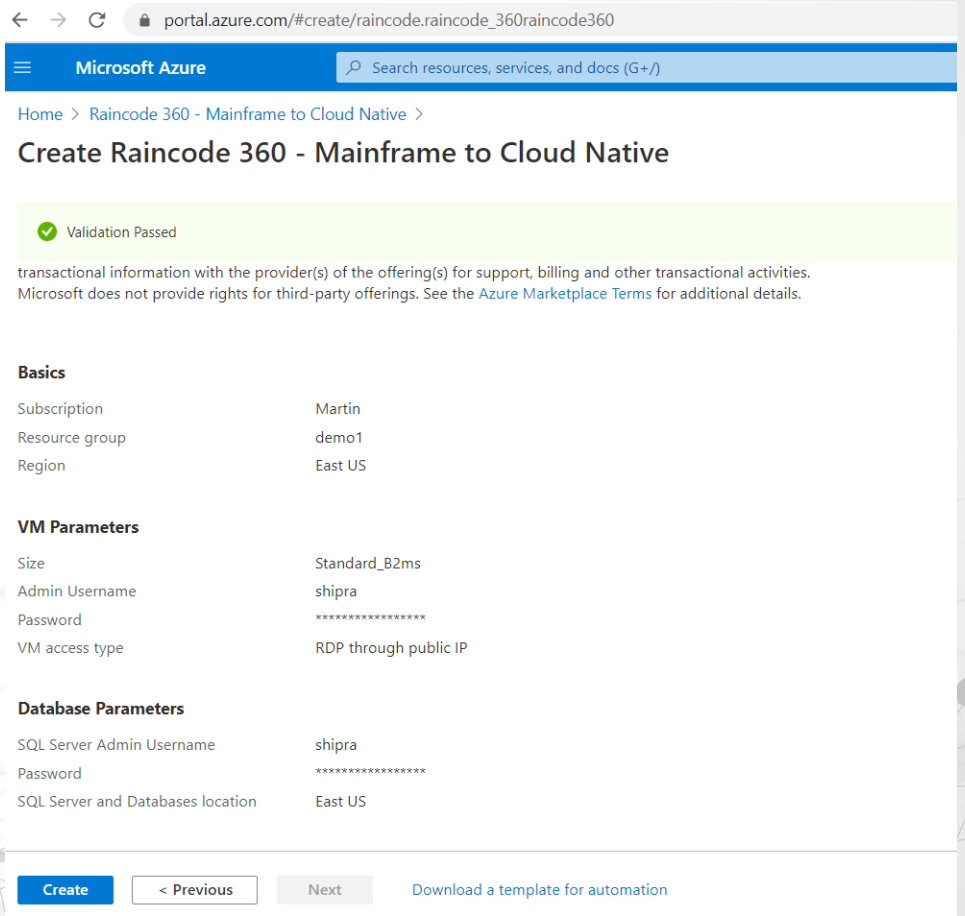
RAINCODE 360
Advanced Showcase

Steps to create Raincode 360 from Marketplace

- Step 9: Click Review + Create

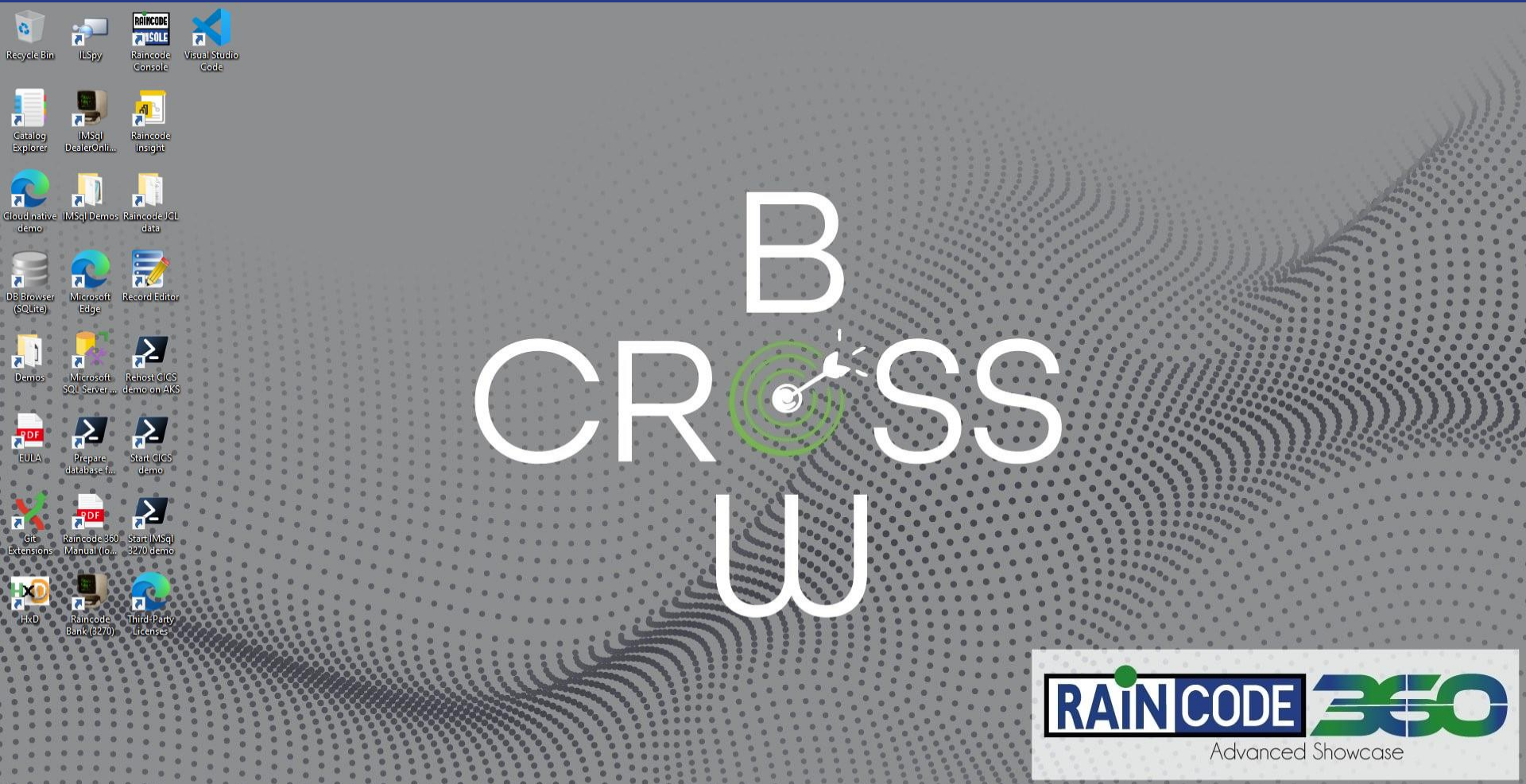
- Step 10: Once the validation is passed, click Create

After the successful creation, VM will appear under Virtual Machines (Azure Services)



RAINCODE 360 Advanced Showcase

Welcome to Raincode 360!

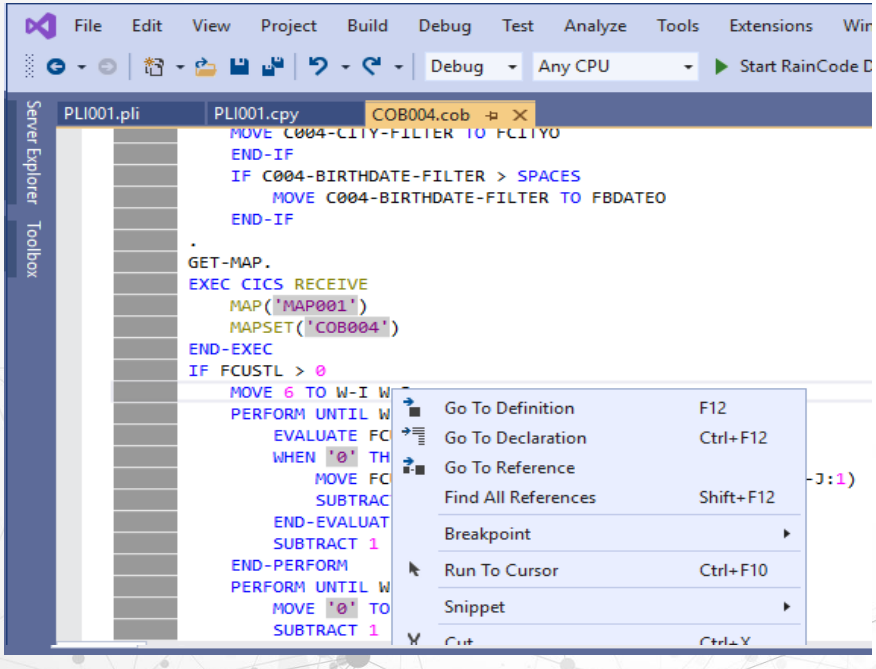


Developer Experience

- Demos:
  - Visual Studio Integration
  - Repository Database
  - Raincode Insight
  - Raincode Insight – Call Graph Tools

Visual Studio Integration

- Raincode products are fully integrated with the Microsoft Visual Studio development environment and provide developer productivity tools, such as:



Some more features to name:

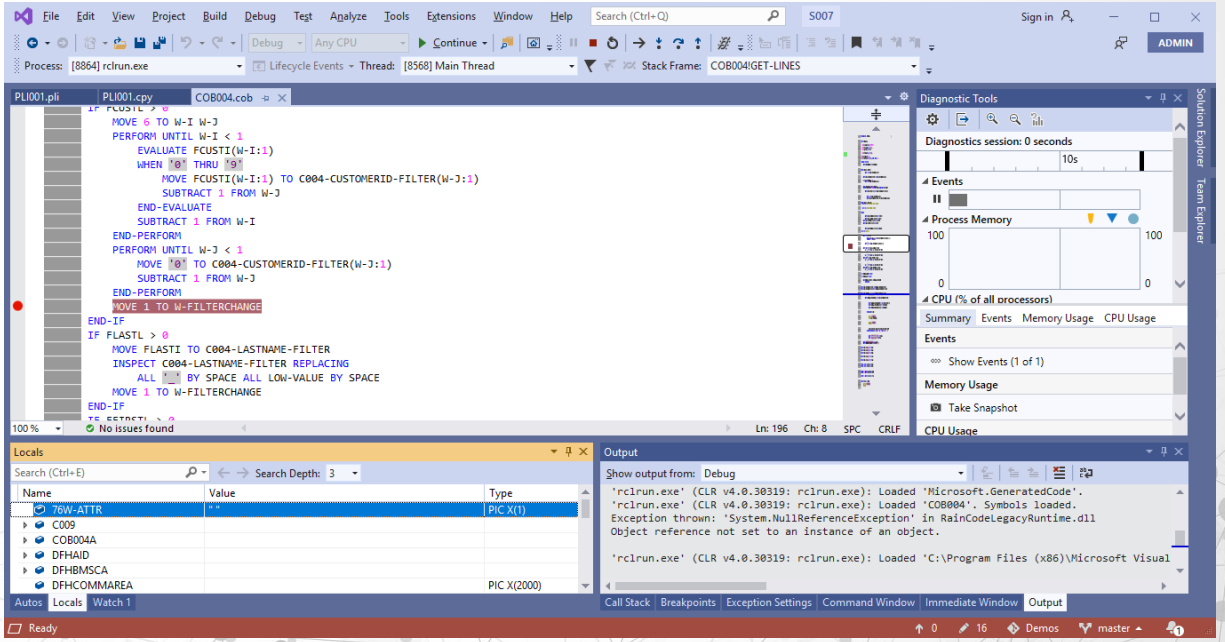
- Visuals:
  - Color coding
- Assistance in editing:
  - Code folding
  - IntelliSense
  - Automatic completion (variables, labels)
- Navigation
  - Declaration (jump to definition)

Hovering over SQL statements shows the converted SQL statement sent to the database.

RAINCODE BEO Advanced Showcase

Visual Studio Integration – Symbolic Debugging

- Raincode compilers are fully integrated with the development environment debugger.
- It allows users to:
  - set breakpoints in the source code
  - perform step-by-step debugging



RAINCODE BEO
Advanced Showcase

Repository Database

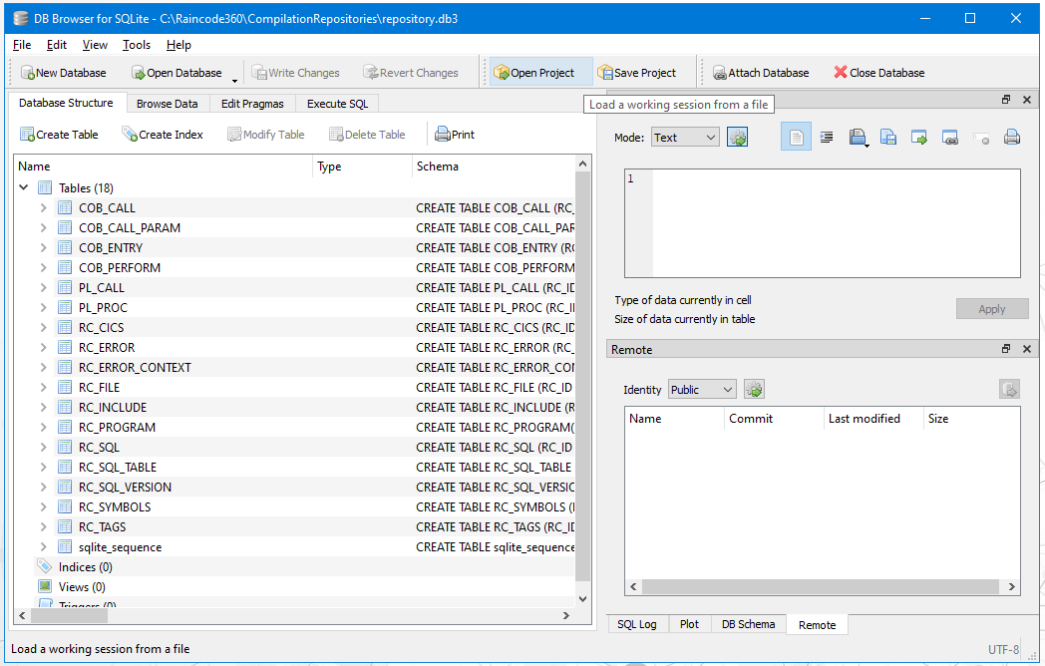
RAIN CODE 360
Advanced Showcase

Repository Database

- The Raincode Legacy compilers can populate a repository database when compiling
- This repository describes various artifacts found in the source code:
  - Programs, including size statistics, compilation time, compiler version, and compiler return code.
  - Procedures
  - CICS statements
  - SQL statements
  - Compilation errors
  - Call graph, with discrimination between local and cross-program calls
  - Etc.
- Essential asset for any non-trivial migration projects

Repository Database – How to demonstrate

- Step 1: Open the solution from folder Qix Emulator – Bank Application
- Step 2: Select Build -> Rebuild Solution
- Step 3: Open **repository.db3** to browse the repository, placed at C:\Raincode360\CompilationRepositories



We are using DB Browser for browsing the SQLite database. This program is also provided with Raincode 360.

https://sqlitebrowser.org/dl/

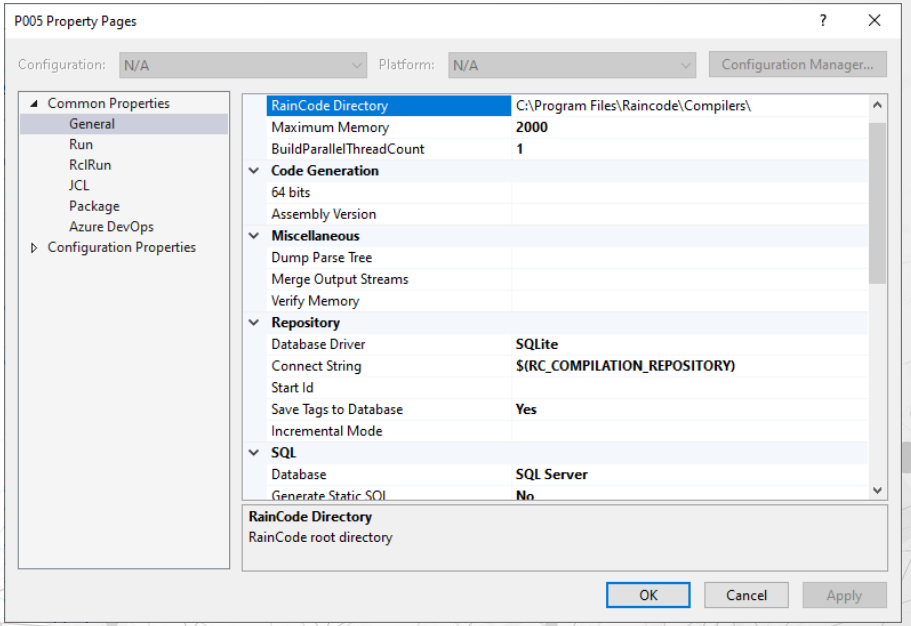
RAINCODE 360
Advanced Showcase

Repository Database – How to demonstrate

- The repository database made by compiling all the demos within Raincode 360, located at C:\Raincode360\CompilationRepositories\repository.db3 is already prepared and can be used directly

- If you want to configure the compilation repository, right-click on P005 in the Solution Explorer and select Properties to access the Compilation Repository configuration.

- Under Repository Configuration, you can:
  - Configure Database Driver: SQLite/ODBC. SQLite is easier (SQL Server requires an external database to be set up)
  - Set Connect String

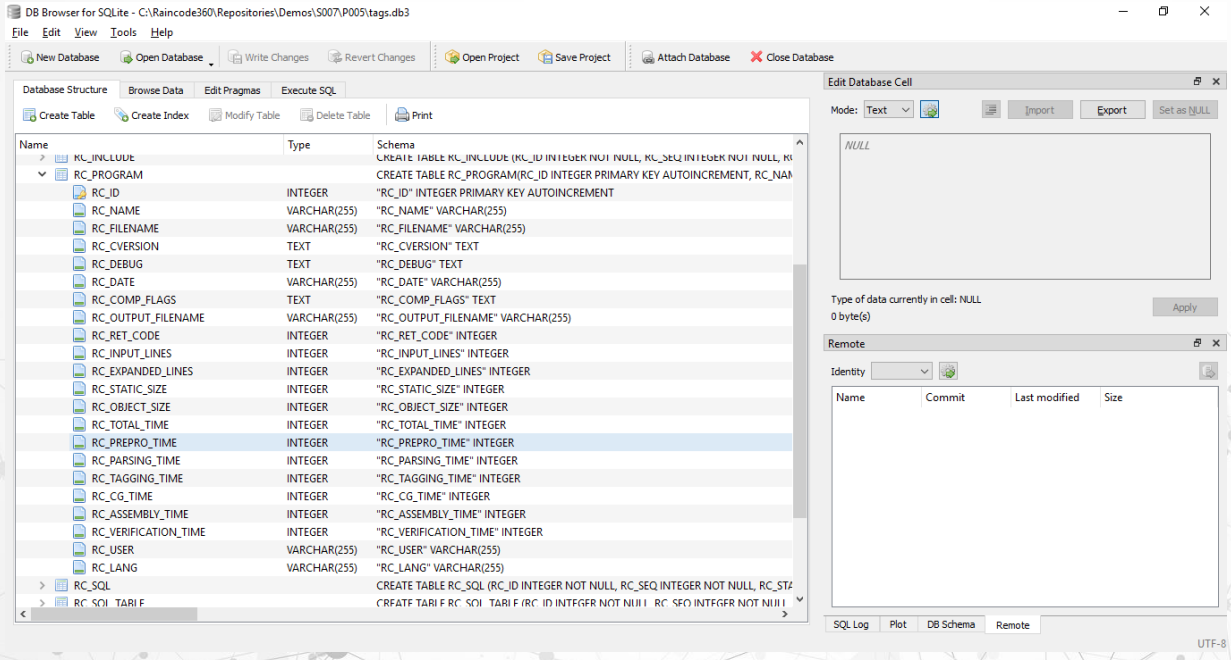


There are several tables created in the compilation repository. For details on the compiler artifacts repository, see Artifacts Repository

RAINCODE 360 Advanced Showcase

Browse Table

- A few tables explained:
  - **RC_PROGRAM:** The RC_PROGRAM table holds information about program files analyzed by the Raincode Legacy Compiler

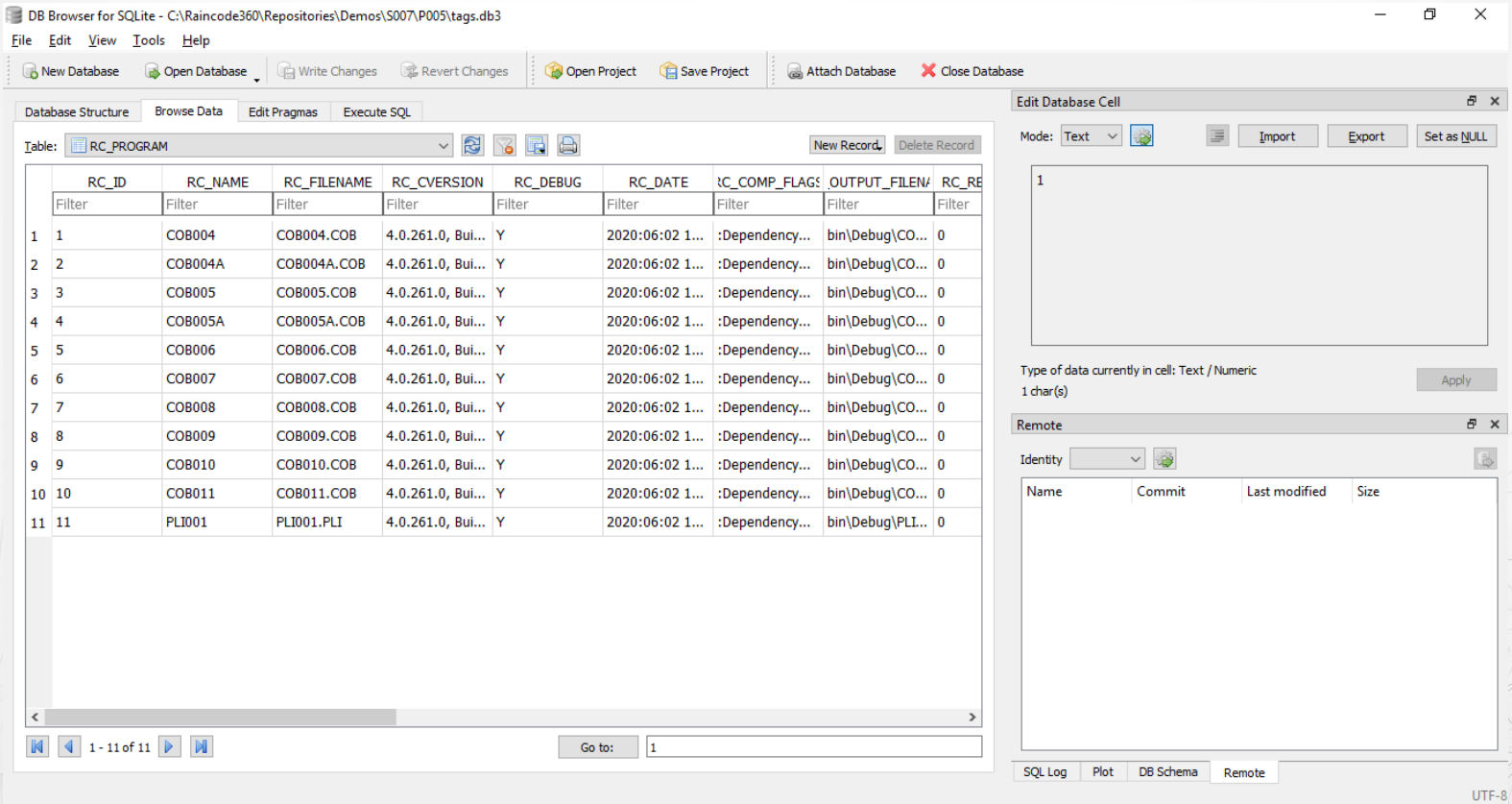


It includes the program name, the compiler version used to build it; compiled in debug mode, compilation date, compilation options used, compilation return code, etc.

RAINCODE 360 Advanced Showcase

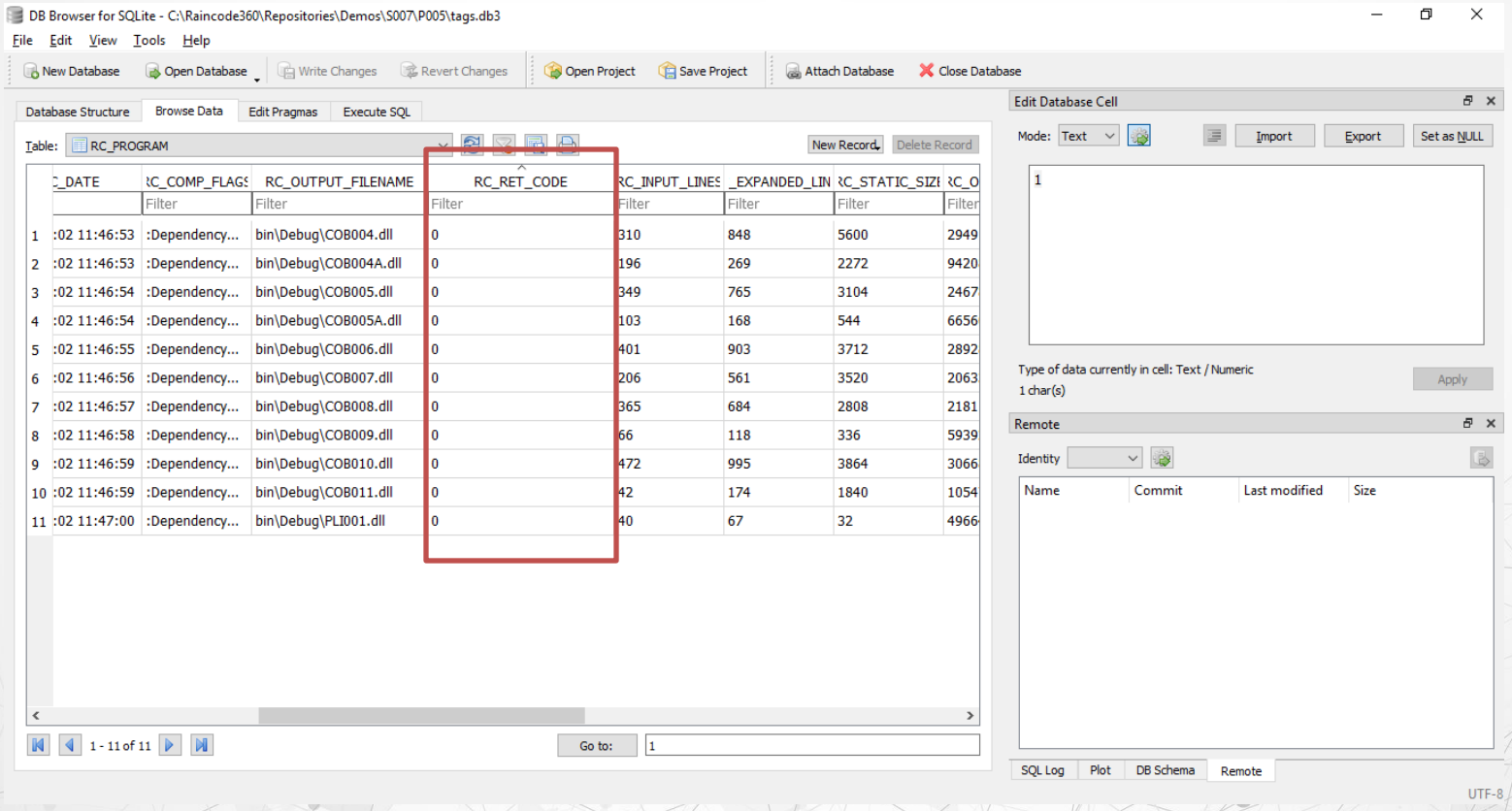
Browse Table

- In the **RC_PROGRAM** table, each row represents a COBOL, or PL/I program in the solution:



Browse Table

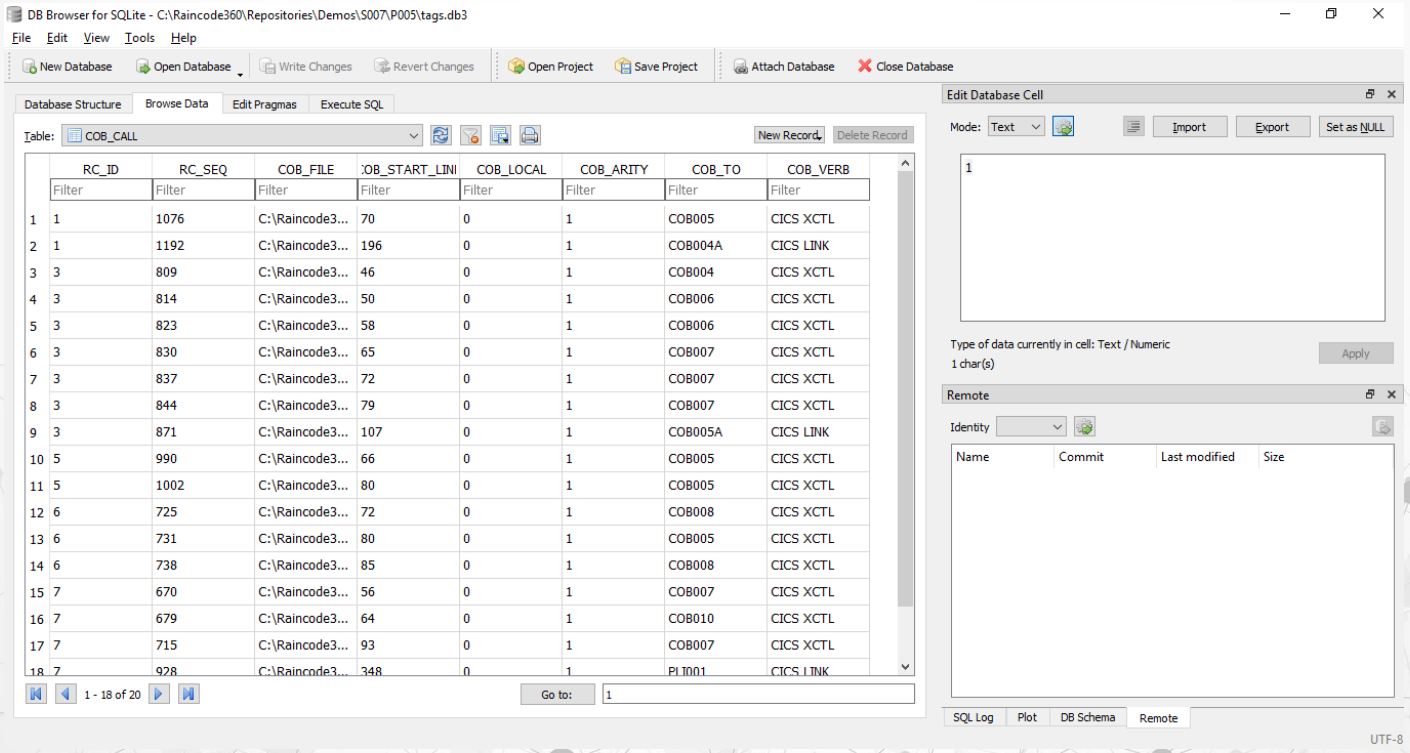
- **RC_RET_CODE=0** indicates that compilation was successful



RC_RET_CODE gives the Raincode Legacy Compiler's return code

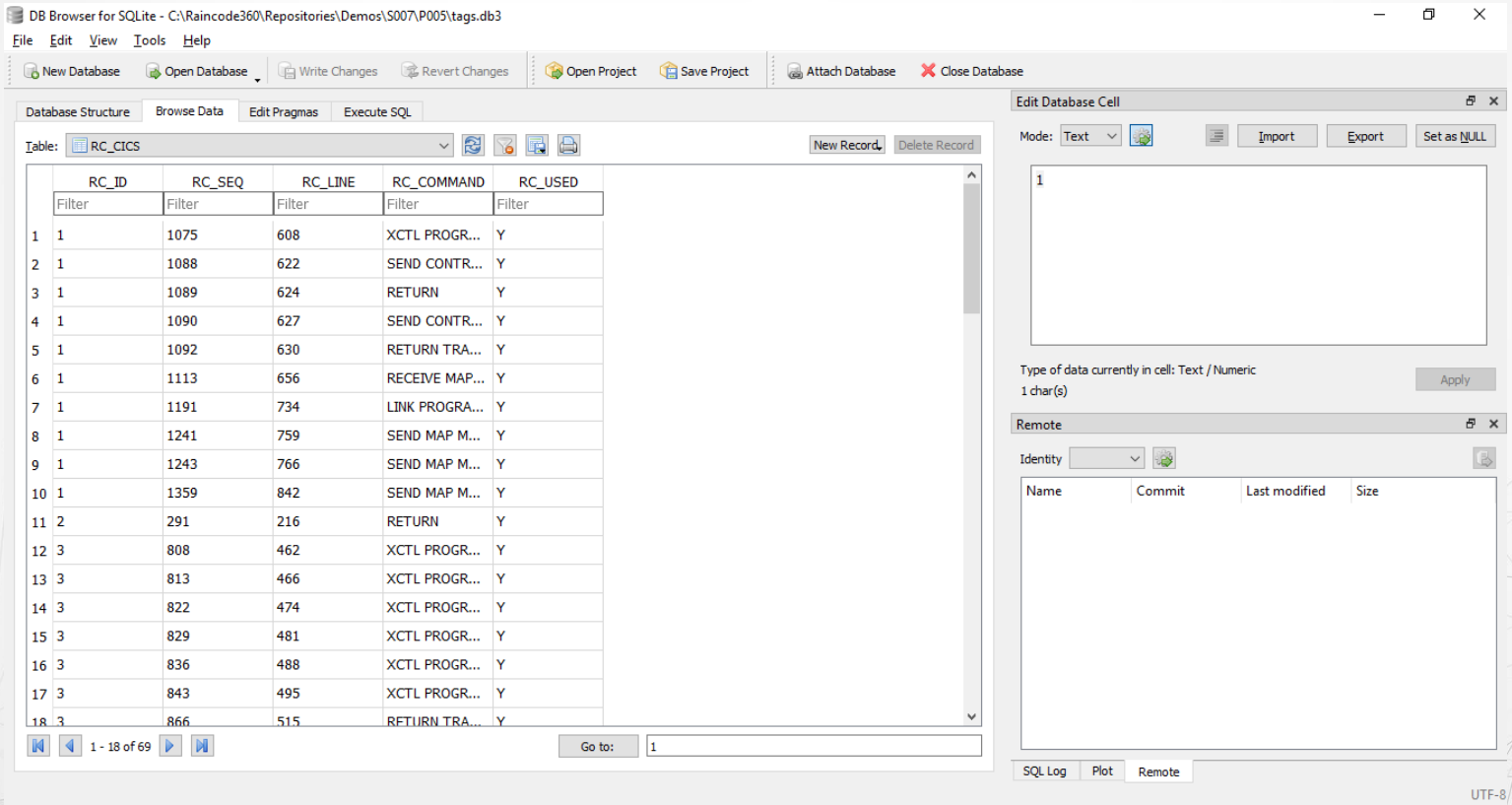
Browse Table

- **COB_CALL:** keeps track of all the CALL and EXEC CICS CALL and XCTL statements found when compiling COBOL source programs. CALL statements can be static, as in CALL “MYPRG”, or dynamic, referring to a variable that holds the program’s name to call, as in CALL PRG-NAME (COBOL only).



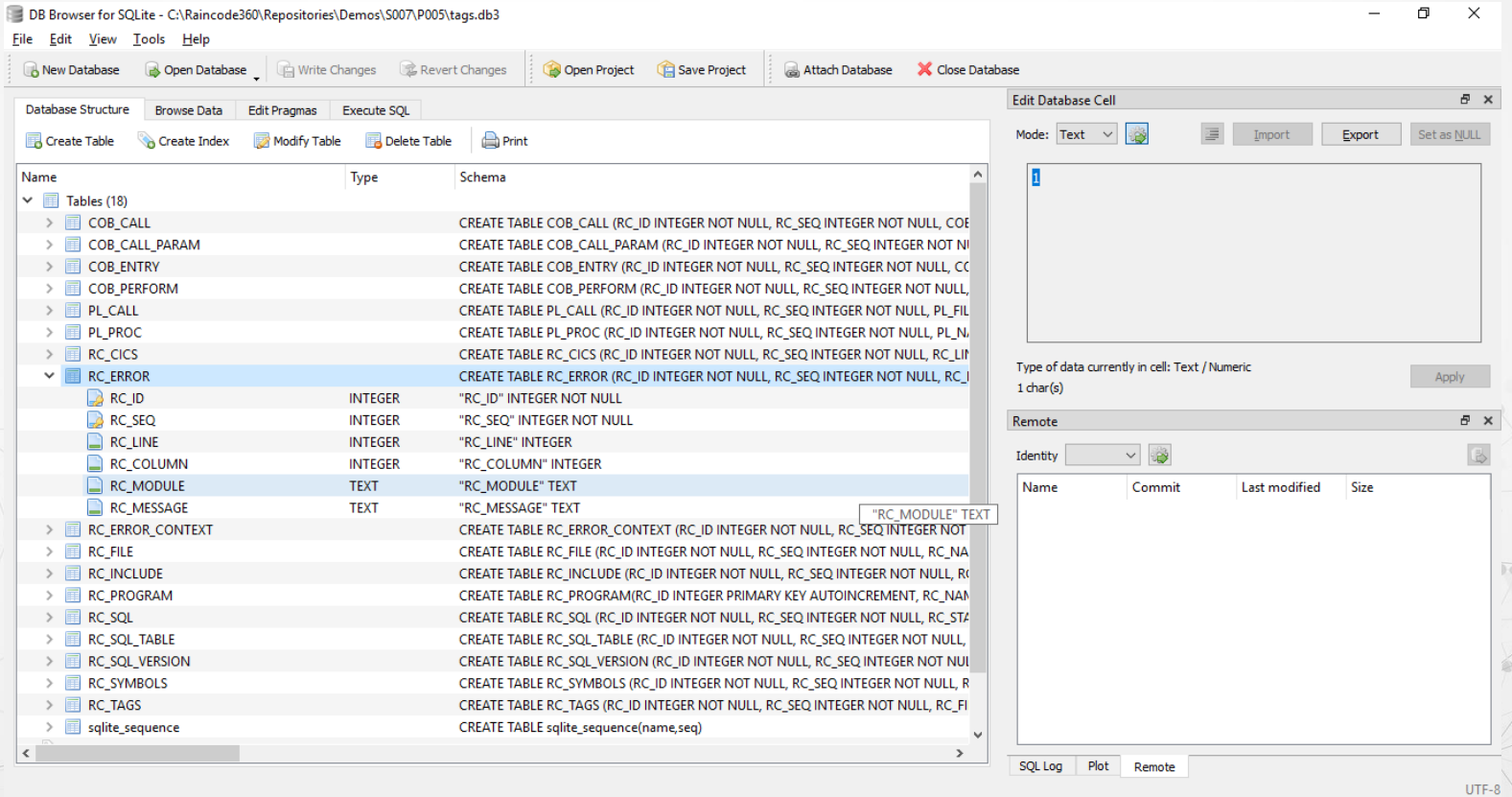
Browse Table

- **RC_CICS:** lists all the CICS statements encountered in the portfolio



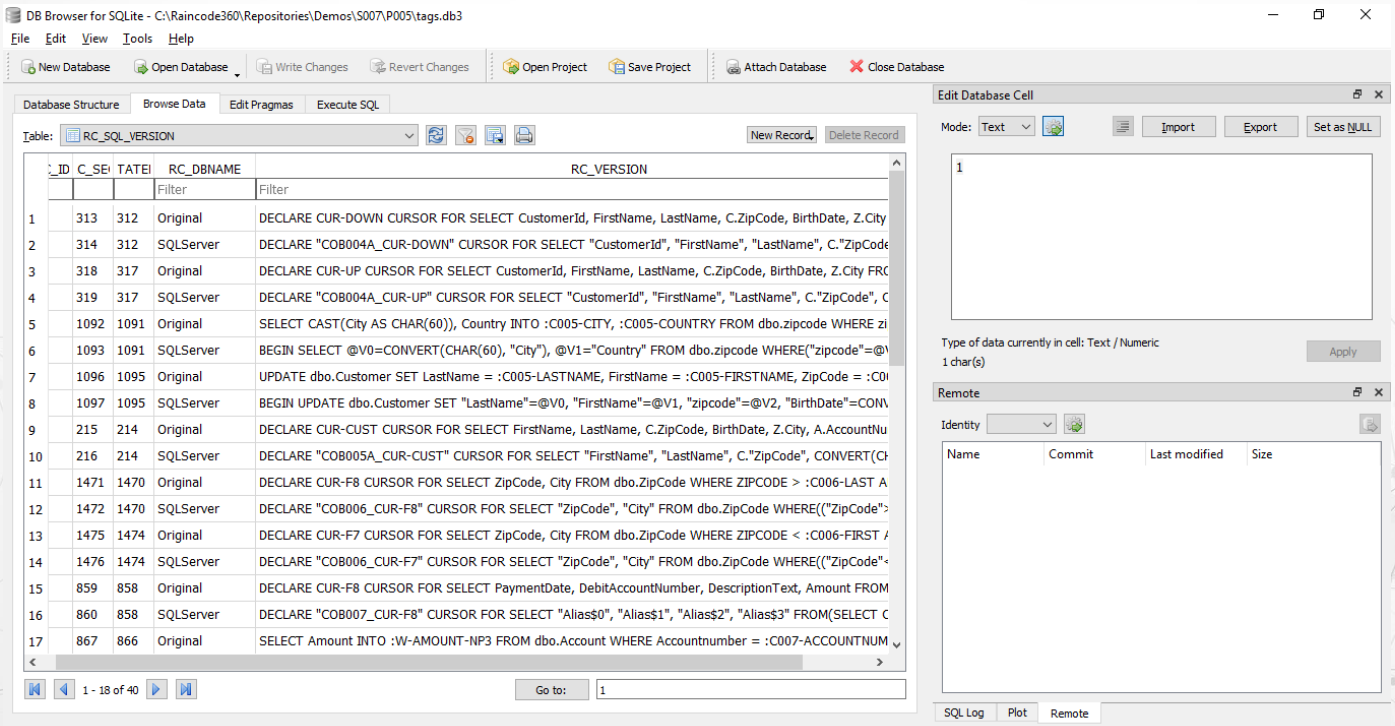
Browse Table

- **RC_ERROR:** summarizes the errors encountered while compiling the program



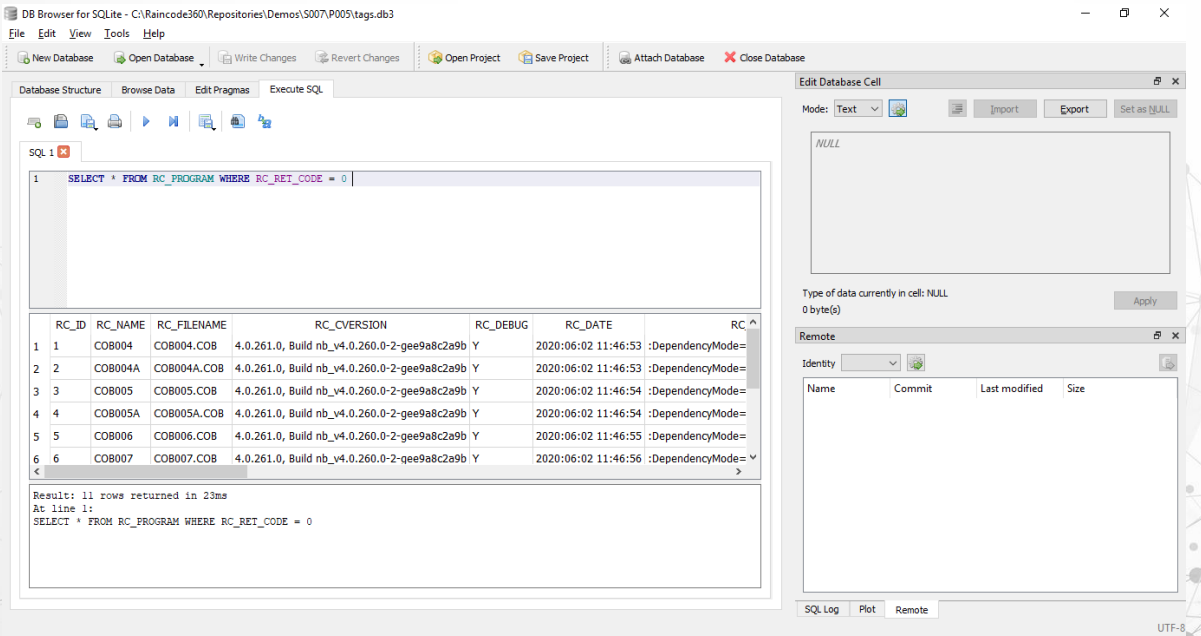
Browse Table

- **RC_SQL_VERSION:** This table holds the information about the automatic SQL translation process. If a translation process is applied, an entry holds the **original** statement (DB2), whereas another entry holds the **translated** statement (SQL Server).



Repository Database – SQL Editor

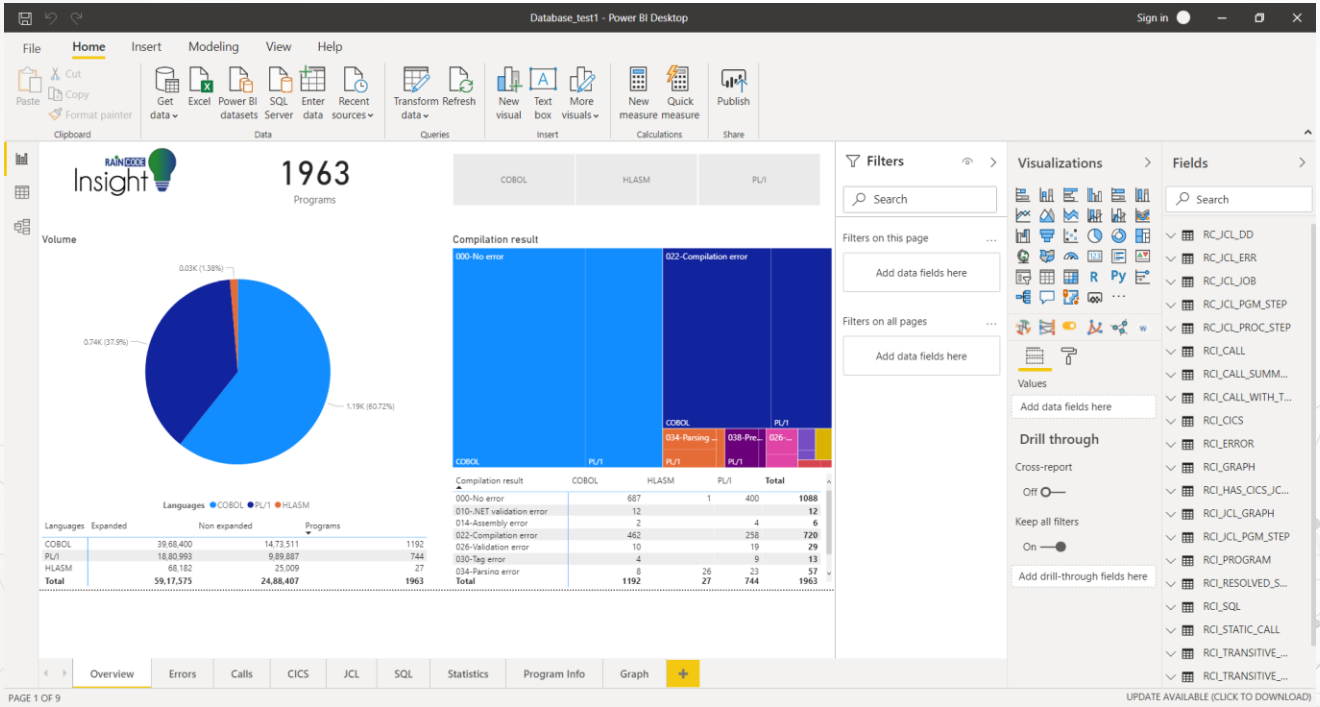
- **In Execute SQL Tab:**
- You can Execute SQL Statement to explore the repository
  - For example, the query **SELECT * FROM RC_PROGRAM WHERE RC_RET_CODE = 0** will yield all successful compilations.



Raincode Insight

Raincode Insight – Introduction

- **Raincode Insight** is a Power BI desktop-based data visualization and analysis tool.

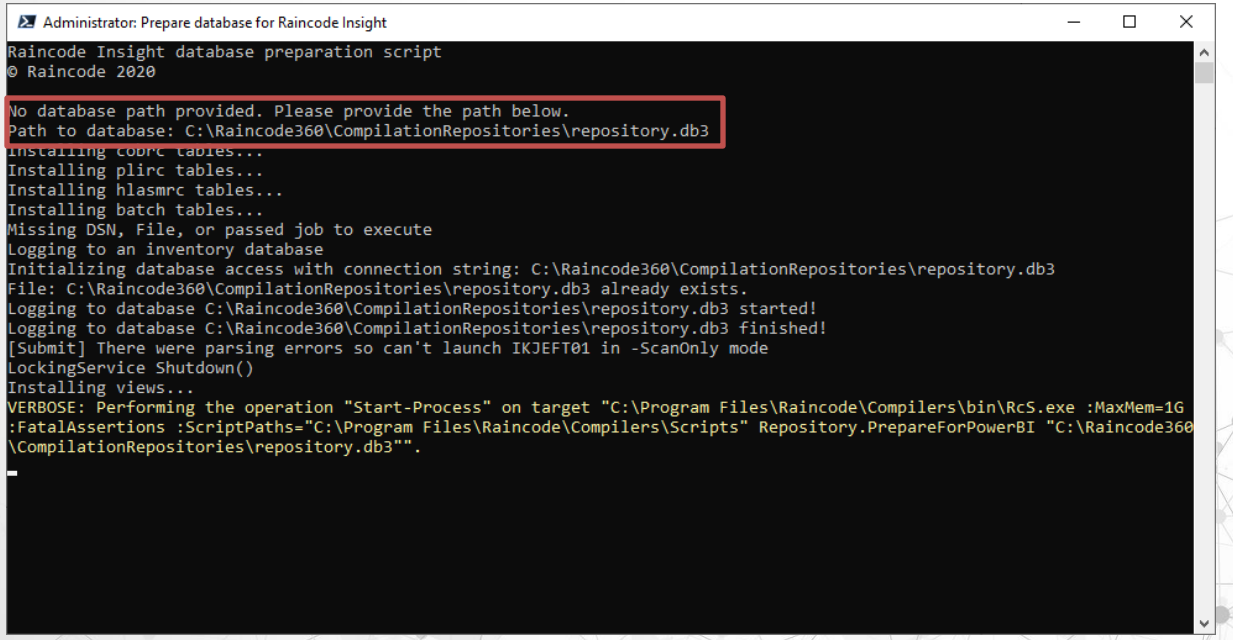


- It provides users with an interactive dashboard on top of the [Raincode Legacy Compilers repository database](#).

RAINCODE B2O Advanced Showcase

Raincode Insight – Install Power BI Desktop and prepare repository

- Step 1: The “Prepare Database for Raincode Insight” shortcut must be executed once to prepare a given compilation repository for Raincode Insight use. This is a one-shot operation that must be done once on a given repository database. The program will ask for the path to the repository database that should be prepared for Raincode Insight.

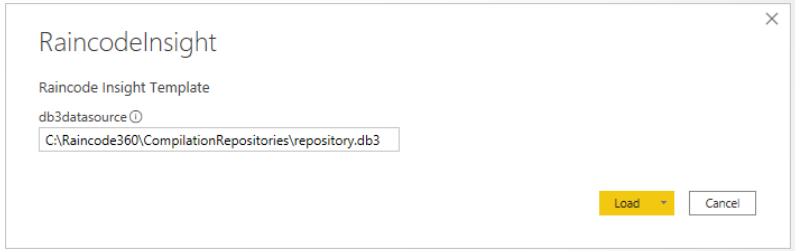


Note: the repository database made by compiling all the demos within Raincode 360, located at C:\Raincode360\CompilationRepositories\repository.db3 is already prepared and can be used directly in Raincode Insight.

RAINCODE 360 Advanced Showcase

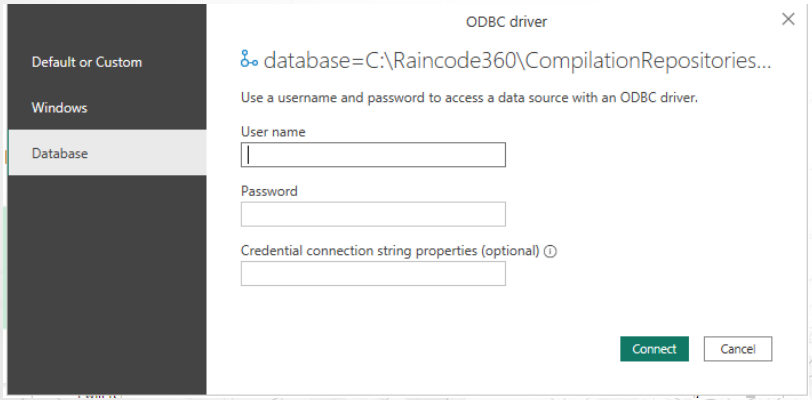
Raincode Insight – How to load database

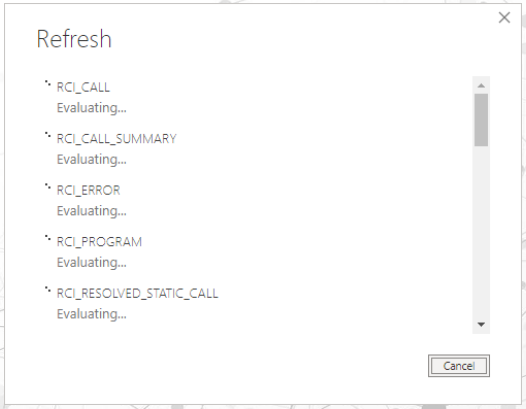
- Step 2: Once the repository database is prepared for Raincode Insight, double click on the “Raincode Insight” shortcut placed at the desktop. Power BI will launch, and a pop-up will appear on the main window



- Step 3: Provide the path to the repository database in the “db3datasource” text box and click on Load.

- Step 4: Provide a username and press Connect. A pop-up progress window with the status of database loading will appear.

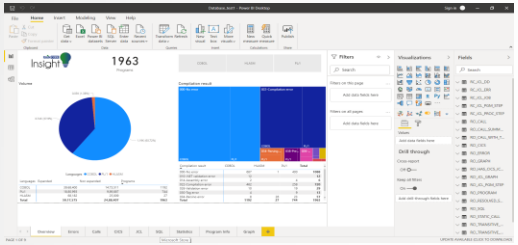




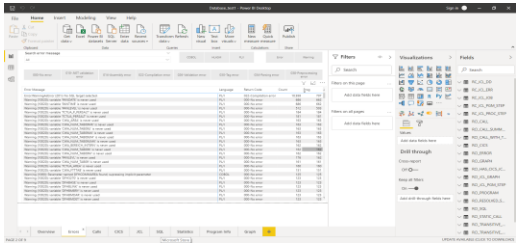
RAINCODE 360
Advanced Showcase

Raincode Insight – Dashboard

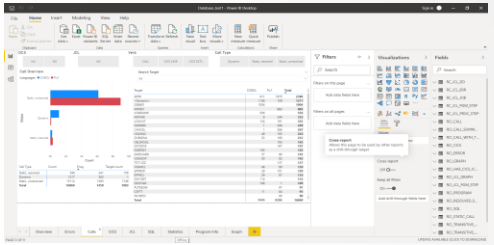
- Once the database is loaded, the Raincode Insight dashboard (9 tabs) is available to the user.



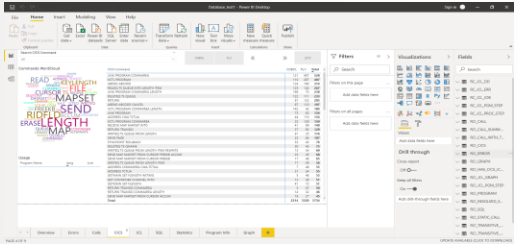
Overview



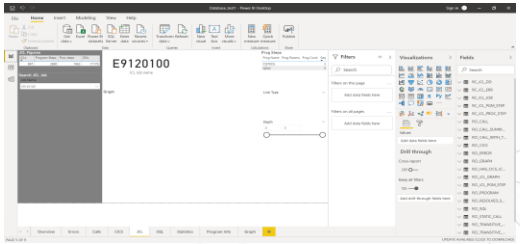
Errors



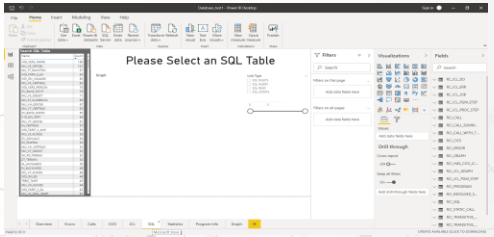
Calls



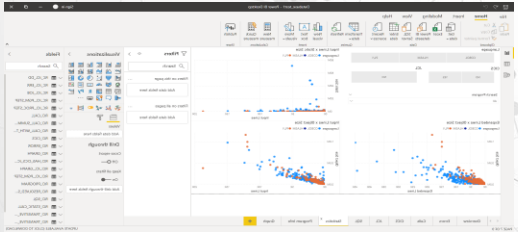
CICS



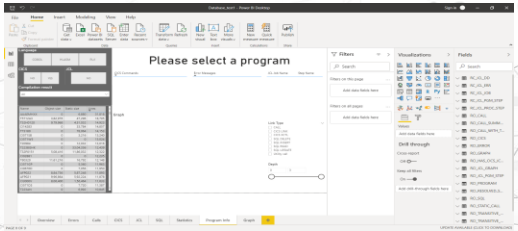
JCL



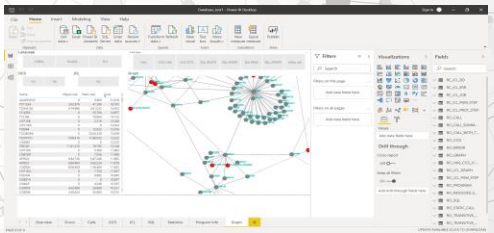
SQL



Statistics



Program Info



Graph

For details on the Raincode Insight dashboard, refer to the Raincode Insight documentation.

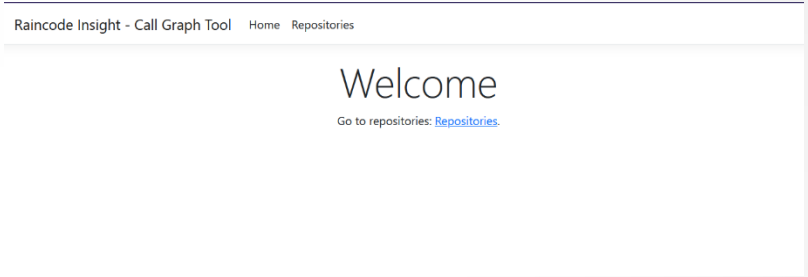
RAINCODE 360 Advanced Showcase

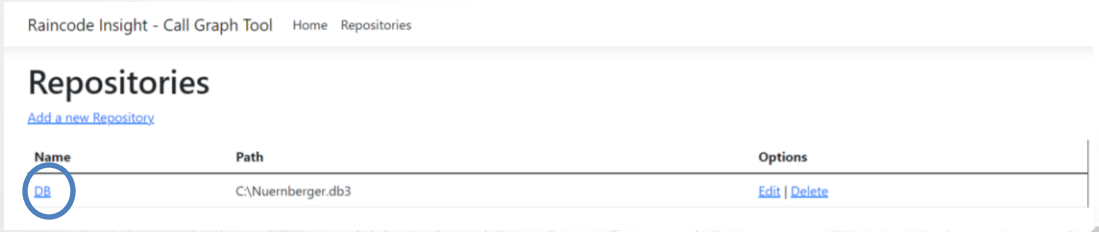
Raincode Insight – Call Graph Tool

RAINCODE 360
Advanced Showcase

Raincode Insight – Call Graph Tool

Raincode Insight -Call Graph Tool is a C# project. This tool explores call graphs that are generated from the repositories created by the Raincode compilers





You can choose to Go to repositories; if a repository has been added previously, you can see a list of the repositories. In case you are using the Call Graph Tool for the first time, you have to Add a new Repository

Click on the repository name to go into details of the call graph

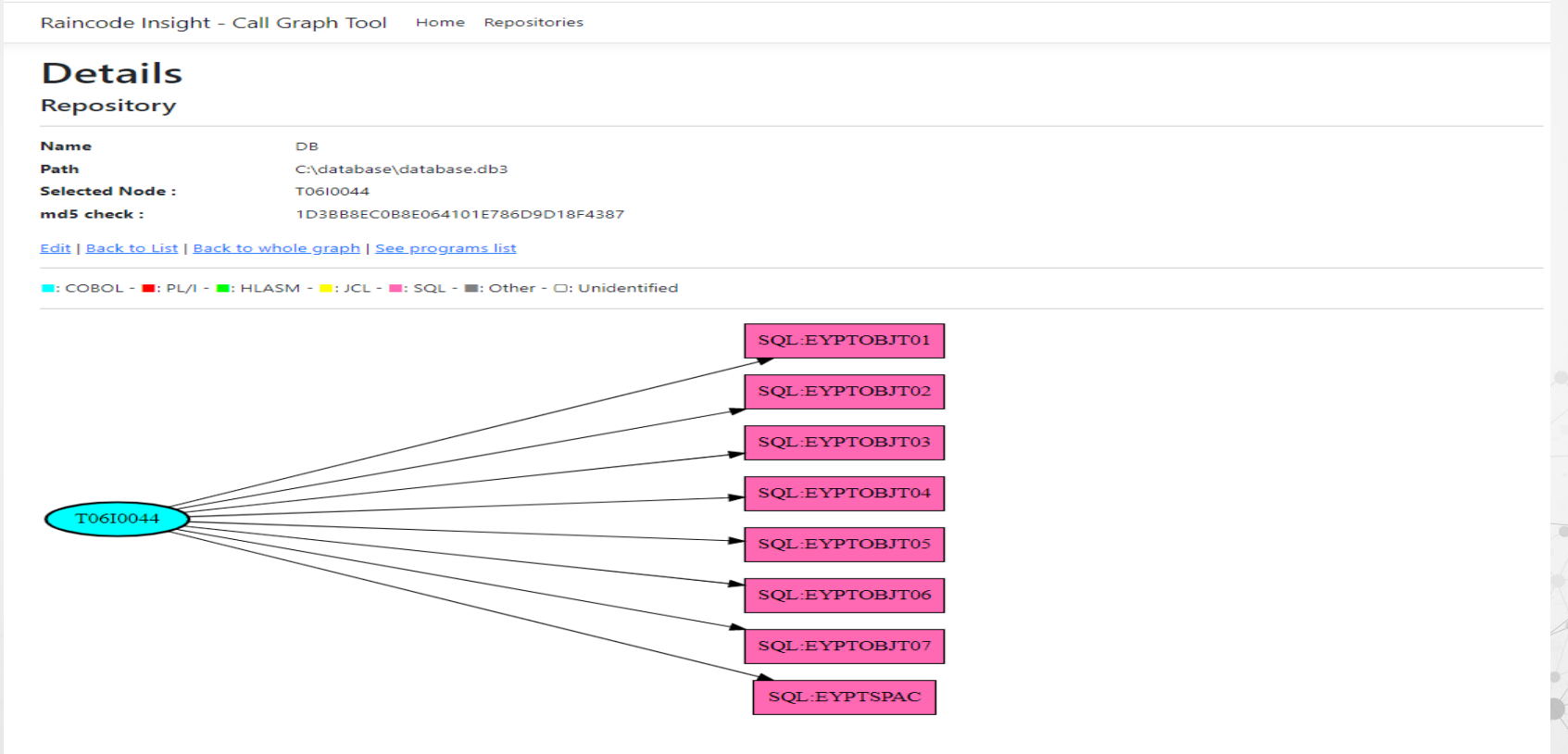
RAINCODE 360 Advanced Showcase

Raincode Insight – Call Graph Tool

The Call Graph Tool details page shows you the calling relationships between subroutines and provides you with the options of Edit, Back to List, Back to the whole graph, and See program list. It also shows you the different color coding of the language used in the program.

Raincode Insight – Call Graph Tool

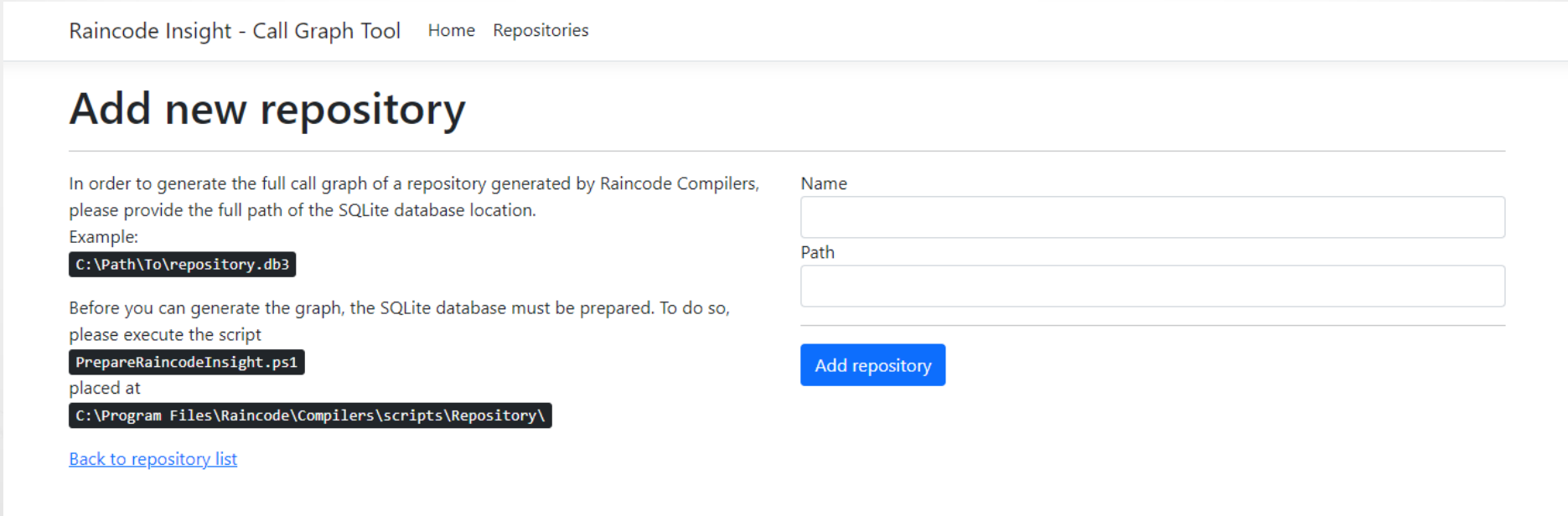
Once you click on any of the program, it shows the relationships of the subroutines of that program.



Raincode Insight – Call Graph Tool – Add a new repository

Step 1: To add a new repository, click on Add a new Repository

Step 2: Provide the name and full path of the SQLite database location and click Add repository



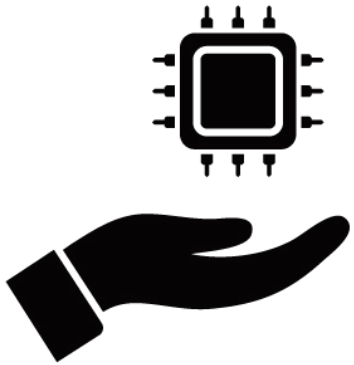
Raincode Legacy Compilers

- Demos:
  - An Introduction to HLASM
  - Legacy calling Assembler
  - SQL Rewriting
  - Dump Processor
  - Custom logger
  - Legacy with database access
  - Standalone SQL Conversion
  - File Access via FileConfig
  - National Character Output on Console
  - User Defined Functions
  - Timeshift
  - RadaR

An introduction to HLASM

A true compiler for ● ASM 370

Not a translator
Not a converter
Not an emulator



Comprehensive coverage

Self-modifying code
CICS verbs
Macro processing
Most commonly used system macros

Seamless integration

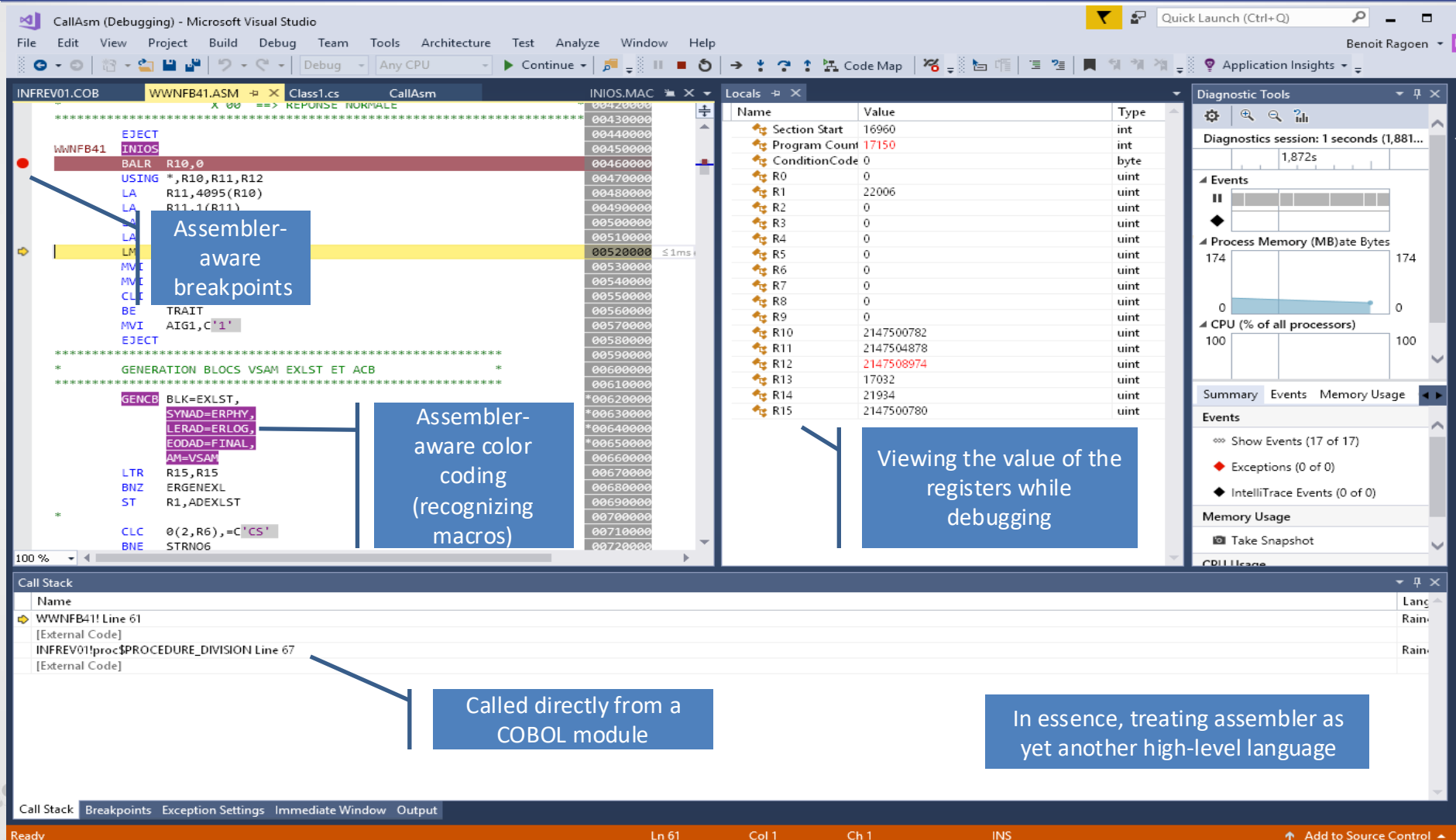
Called from ● COBOL and ● PL/I
Visual Studio (Debugging)

Single source:
execute the same ASM code on .NET and mainframe

Optimized modes available

RAINCODE 360
Advanced Showcase

HLASM – Visual Studio Plugin



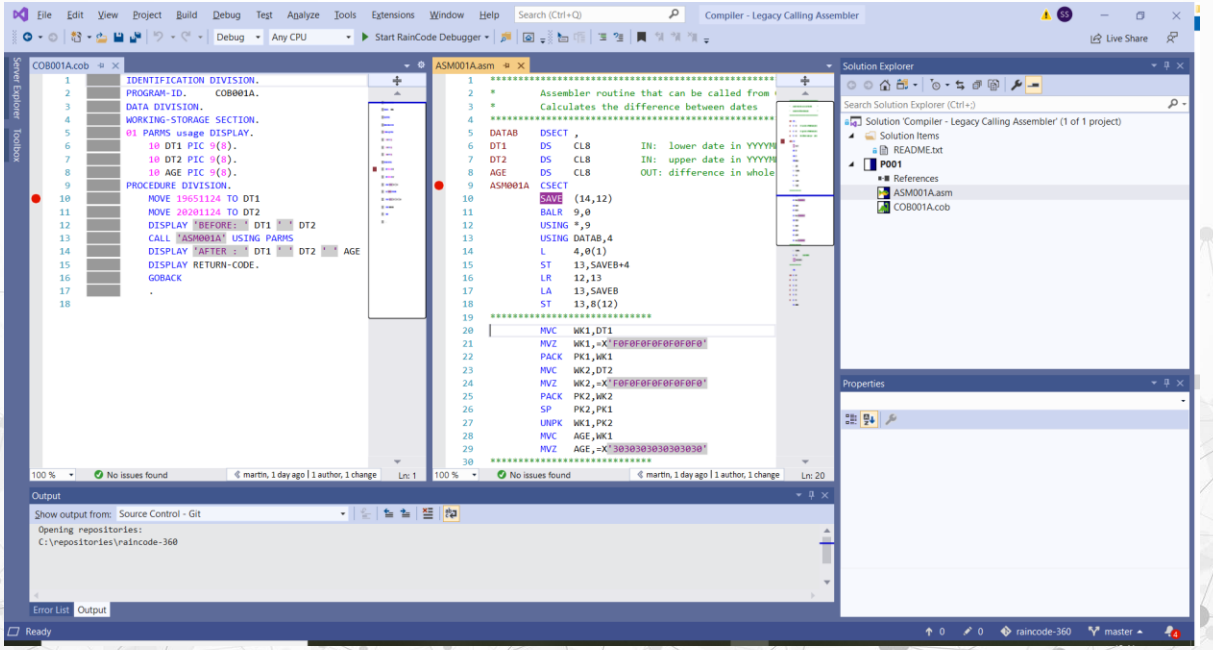
Legacy Calling Assembler

Legacy Calling Assembler – Introduction

- An example of how to run a COBOL program with the rclrun (the Raincode runner) from the Visual Studio
- Demonstrates how to call a subroutine written in the assembler
- The assembler routine calculates the difference in years between two dates, that are passed as parameters by the COBOL main program

Legacy Calling Assembler – How to demonstrate

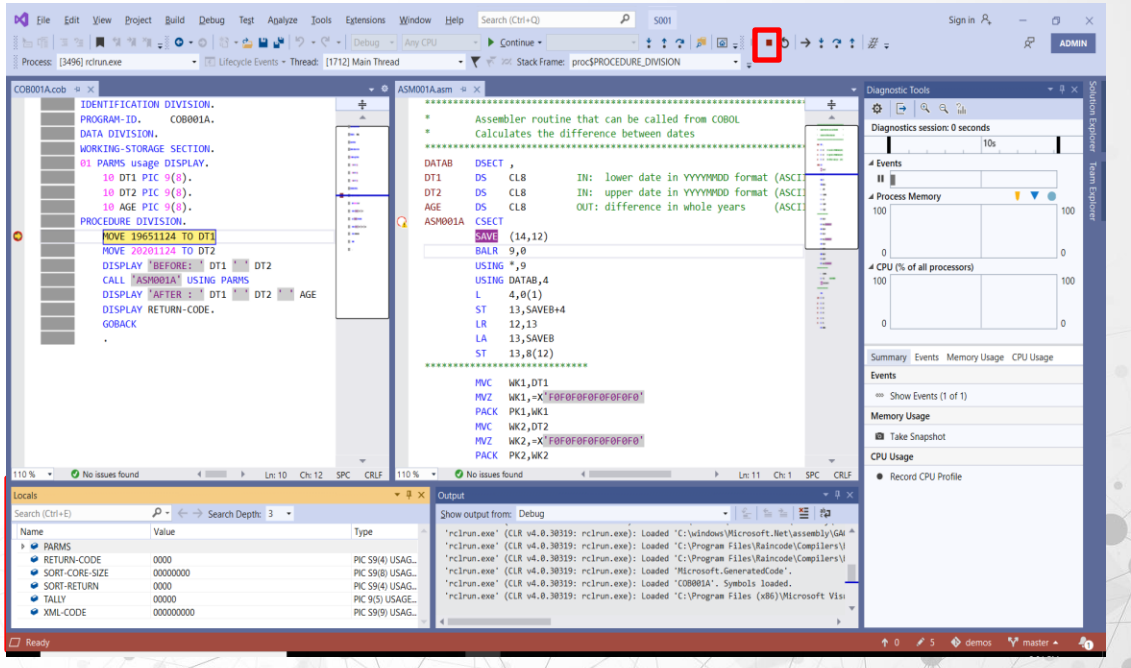
- Step 1: Open the solution from the folder Compiler - Legacy Calling Assembler
- Step 2: Put breakpoints in COB001A.cob program and ASM001A.asm



RAINCODE 360 Advanced Showcase

Legacy Calling Assembler – How to demonstrate

- Step 3: Click on Start Raincode Debugger
  - It activates solution in the debugging mode, and execution will stop at the breakpoint
  - Use the step buttons to step through the program
  - Check the locals windows to see a list of local values and their corresponding values
  - Stop debugging by pressing the stop icon in the debug toolbar



RAINCODE BEO Advanced Showcase

SQL Rewriting

Compile-time DB2 to SQL Server translation

Original DB/2:

```
EXEC SQL DECLARE C1 CURSOR FOR
       SELECT CONCAT(CONCAT(STRIP(UPPER(LastName),
              BOTH), ', '), STRIP(FirstName, TRAILING))
       FROM People WHERE BirthDate + 30 YEARS > :Today
       WITH UR
       FETCH FIRST ROW ONLY;
```

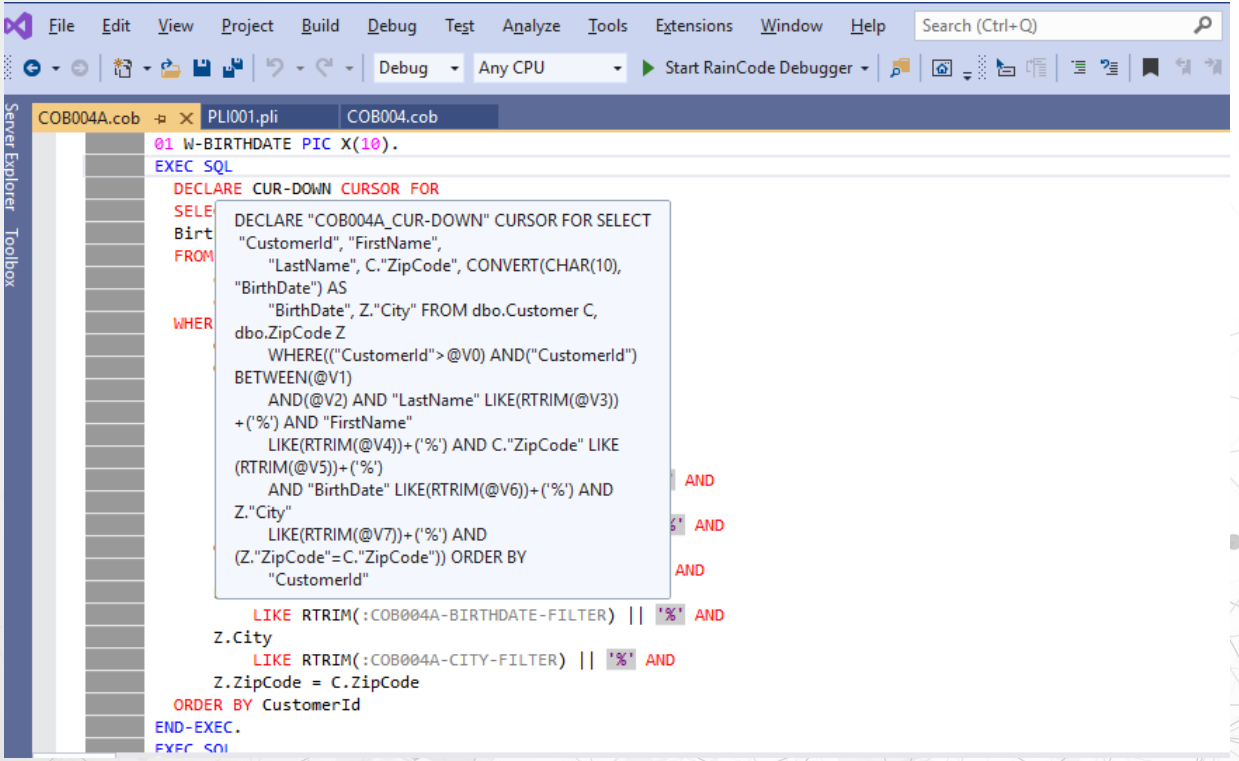
Converted SQL Server:

```
SELECT TOP(1)
       ((dbo.STRIP(UPPER(LastName), 'BOTH', DEFAULT))+
          (', '))+(dbo.STRIP(FirstName, 'TRAILING', DEFAULT))
       FROM People WITH(READUNCOMMITTED)
       WHERE(DATEADD(YEAR, 30,
             BirthDate)> CONVERT(DATE, @V0))
```

RAINCODE 360 Advanced Showcase

SQL Rewriting

- SQL statements get transformed during the compilation process, but the source code remains intact.



The Visual Studio plug-in allows you to see the transformed SQL statement by hovering the mouse on the original statement.

RAINCODE BEO
Advanced Showcase

Dump Processor

Dump Processor – Introduction

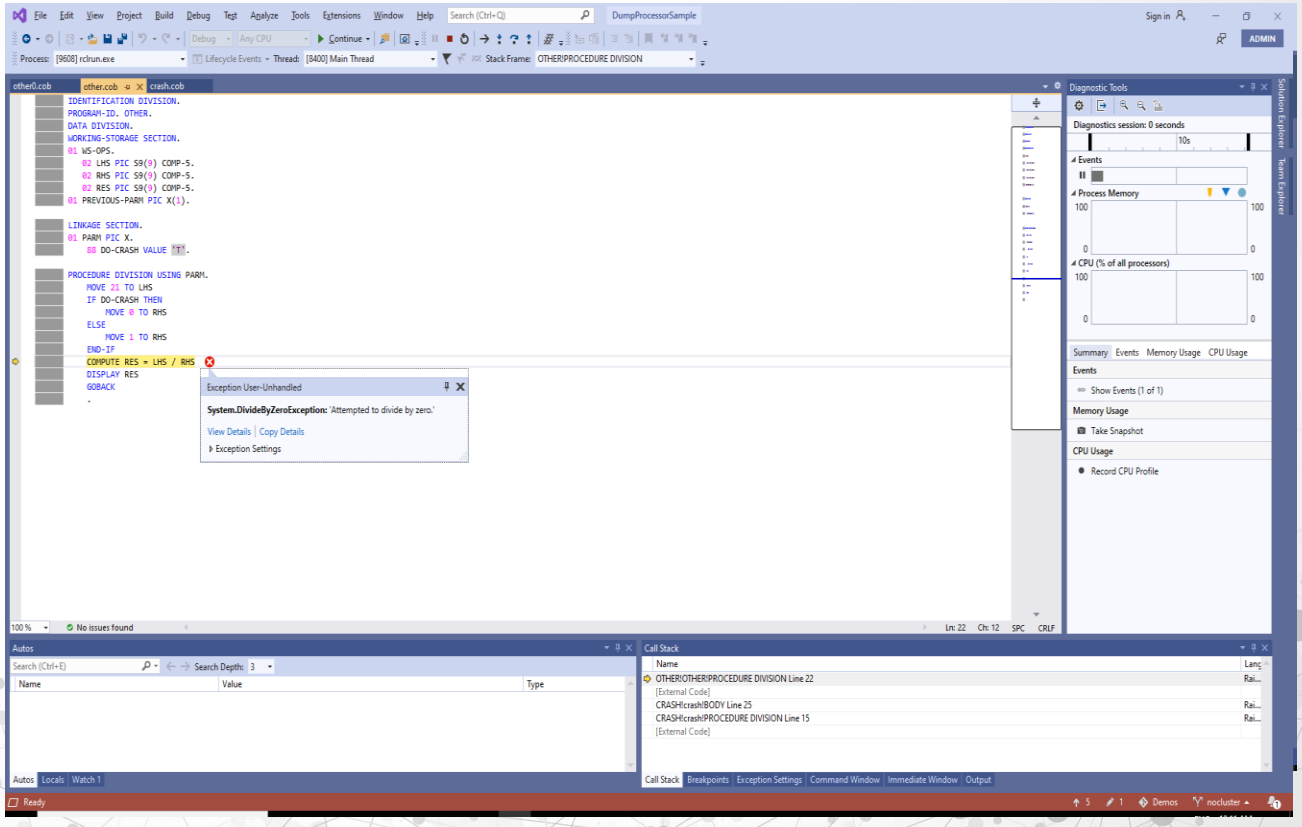
- When a program crashes, it is possible to obtain a report with the following information:
  - The .NET exception that stopped the execution
  - A COBOL stack trace, with the file and the line number
  - the list of variables in all loaded modules and their value
- Dump Processor consists of the following demos:
  - Plugin
  - CSharp Main
  - Plugin-JSON
- Installs a dump processor based on the abstract BaseDumpProcessor class.
- The Dump Processor will produce a formatted report when a program crash.
- Refer to Dump Processor for more details.

Dump Processor – Plugin: How to demonstrate

- Step 1: Open the solution from the folder Compiler - Dump Processor - Plugin

- Step 2: Start the Raincode debugger

Execution will hit a breakpoint in other.cob caused by a DivideByZeroException.



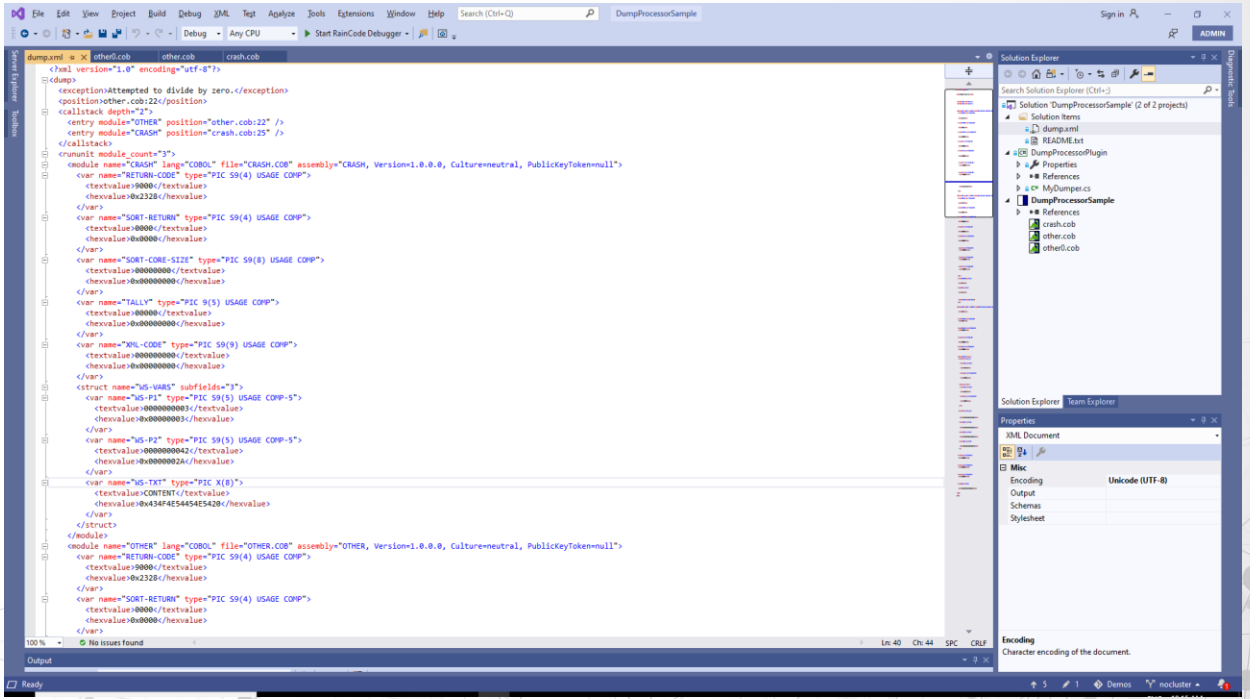
- Step 3: Click continue or close the application window to exit the debugger

RAINCODE BEO Advanced Showcase

Dump Processor – Plugin: How to see output

- Open the dump.xml from solution items

- The XML contains the working storage of the three COBOL programs involved at the time of the crash.

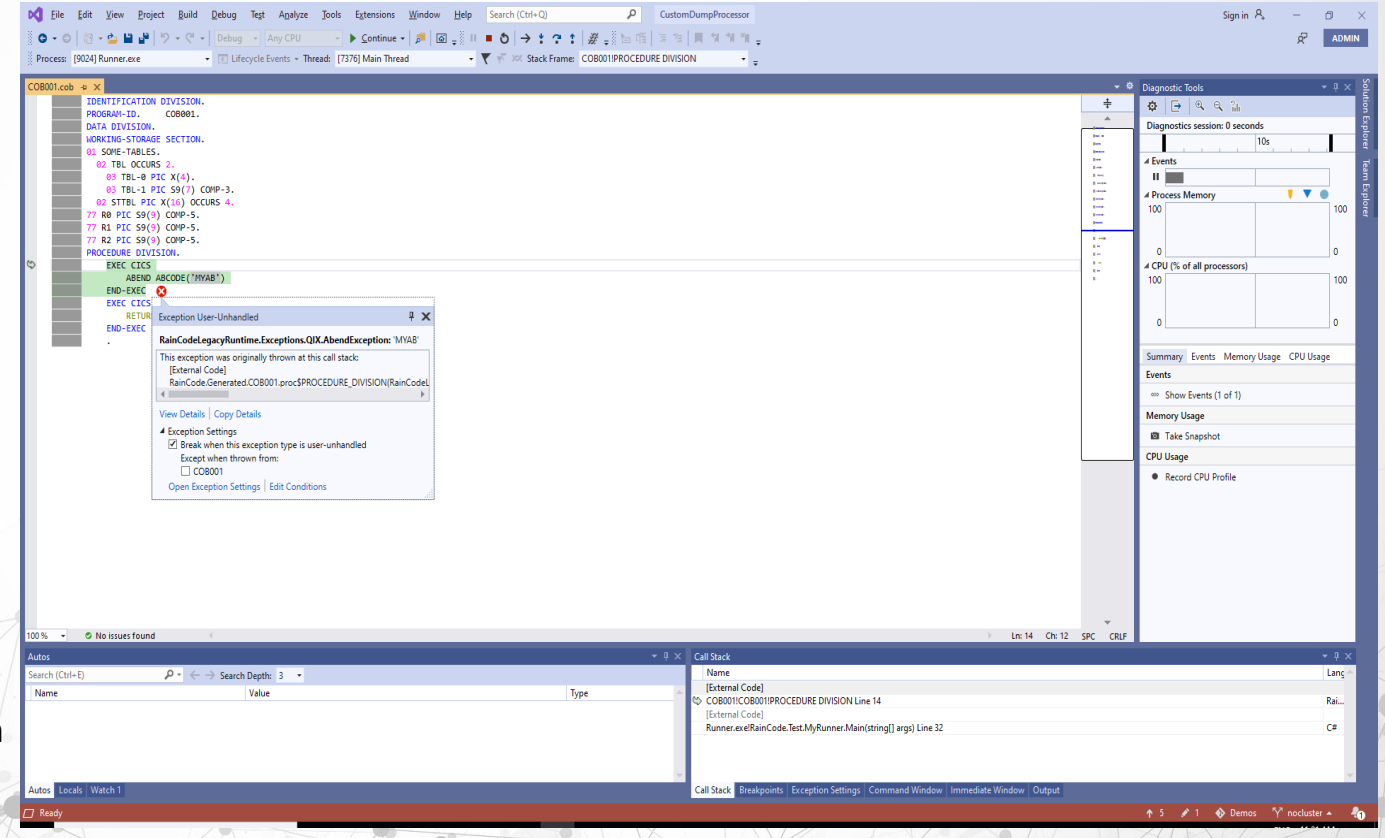


Dump Processor – C# Main: How to demonstrate

- Step 1: Open the solution from the folder Compiler - Dump Processor – Csharp Main

- Step 2: Start the Raincode debugger

Execution will hit a breakpoint in COB001.cob on the CICS ABEND statement



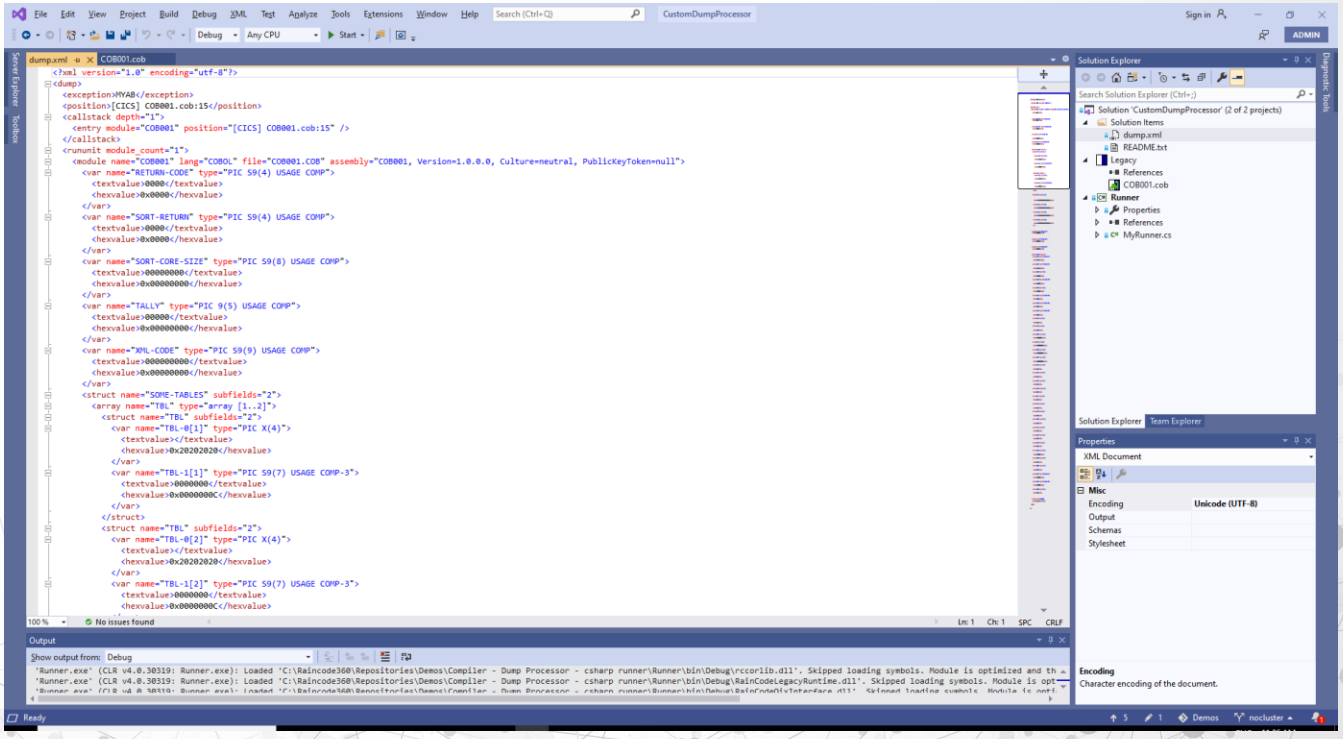
- Step 3: Click continue or close the application window to exit the debugger

RAINCODE 360 Advanced Showcase

Dump Processor – C# Main: How to see output

- Open the dump.xml from solution items

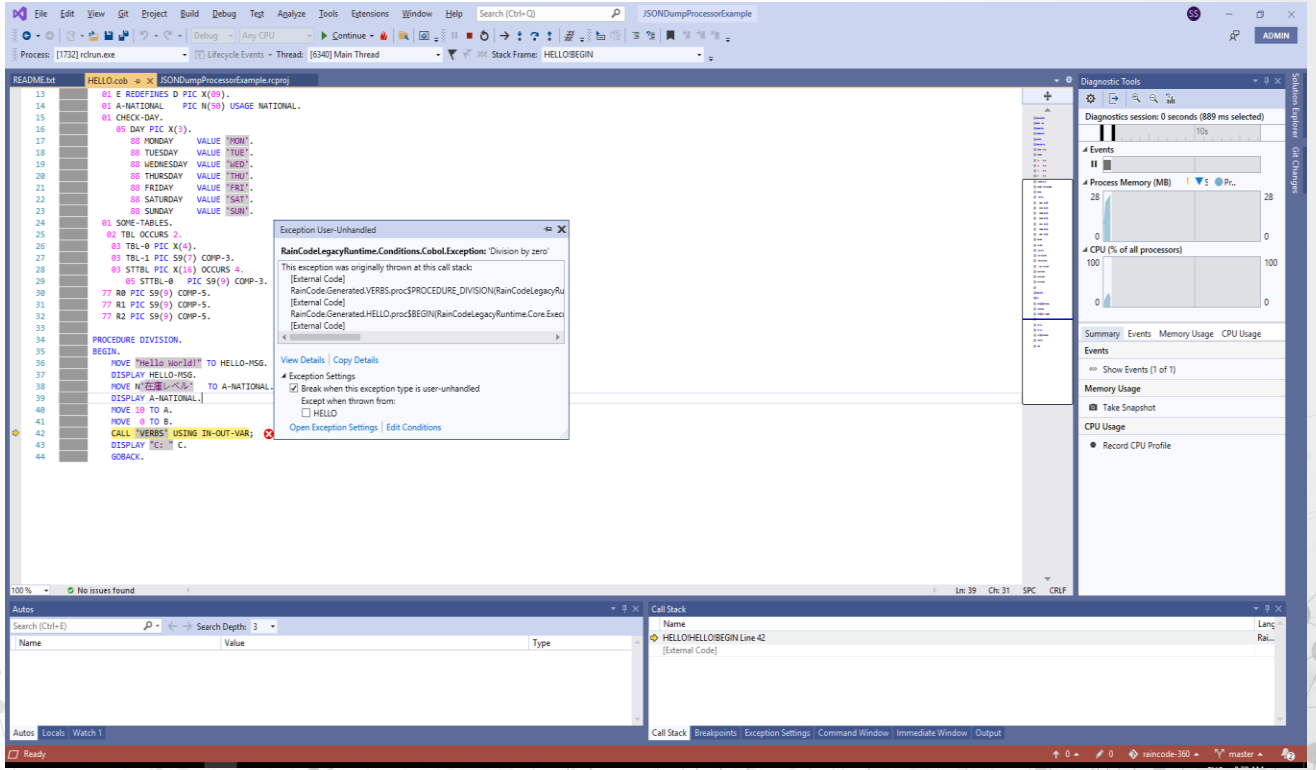
- The file contains the working storage of COB001.cob time of the crash plus the variables of the CICS Exec Interface Block (EIB)



Dump Processor – Plugin - JSON: How to demonstrate

- Step 1: Open the solution from the folder Compiler - Dump Processor - Plugin - JSON

- Step 2: Start the Raincode debugger



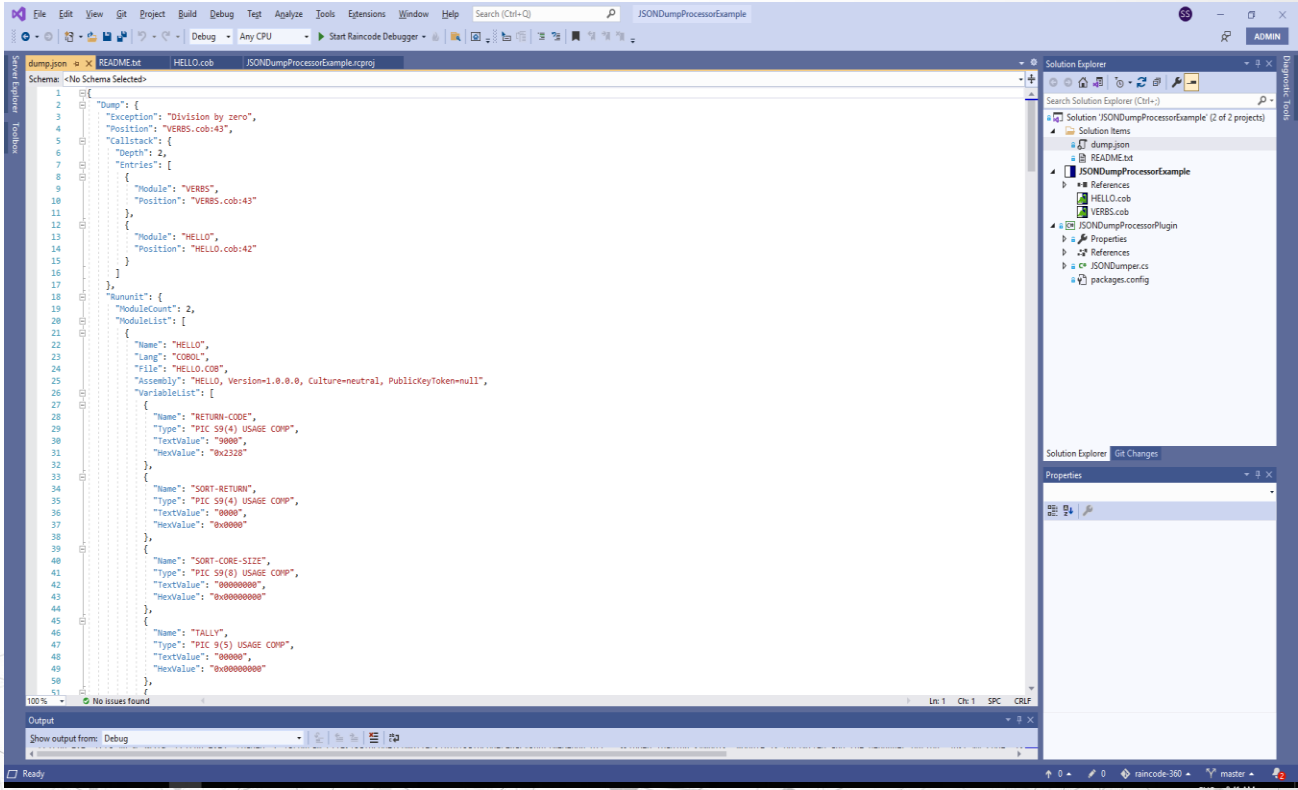
The execution will hit a breakpoint in HELLO.cob caused by a DivideByZeroException in the called module VERBS.cob.

- Step 3: Click continue or close the application window to exit the debugger

RAINCODE BEO
Advanced Showcase

Dump Processor – Plugin - JSON: How to see output

- Open the dump.json from the solution items
- The JSON contains the working storage of the two COBOL programs involved at the time of the crash



Custom Logger

Custom Logger

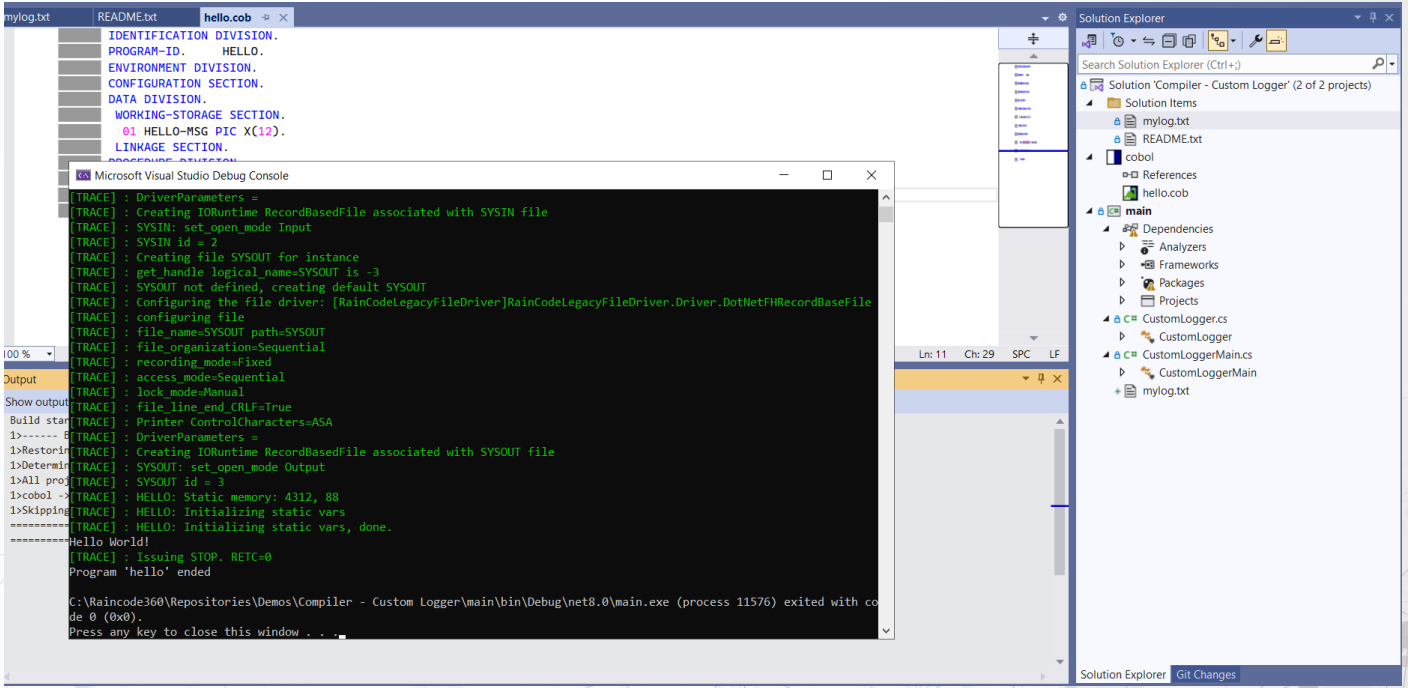
- Demonstrates the use of loggers in a C# runner.
  - Loggers are used to write formatted log messages to a particular target, such as the console or a dedicated log file.
  - The Log Level attribute of a logger can be set to show messages to that level or higher.
- Demonstrates how to create and register multiple loggers.
- Demonstrates how to set the log level and how to issue the additional messages.

For more details, see Logging.

Custom Logger – How to demonstrate

- Step 1: Open the solution from the folder Compiler – Custom Logger

- Step 2: Select Start without debugging(Ctrl + F5).

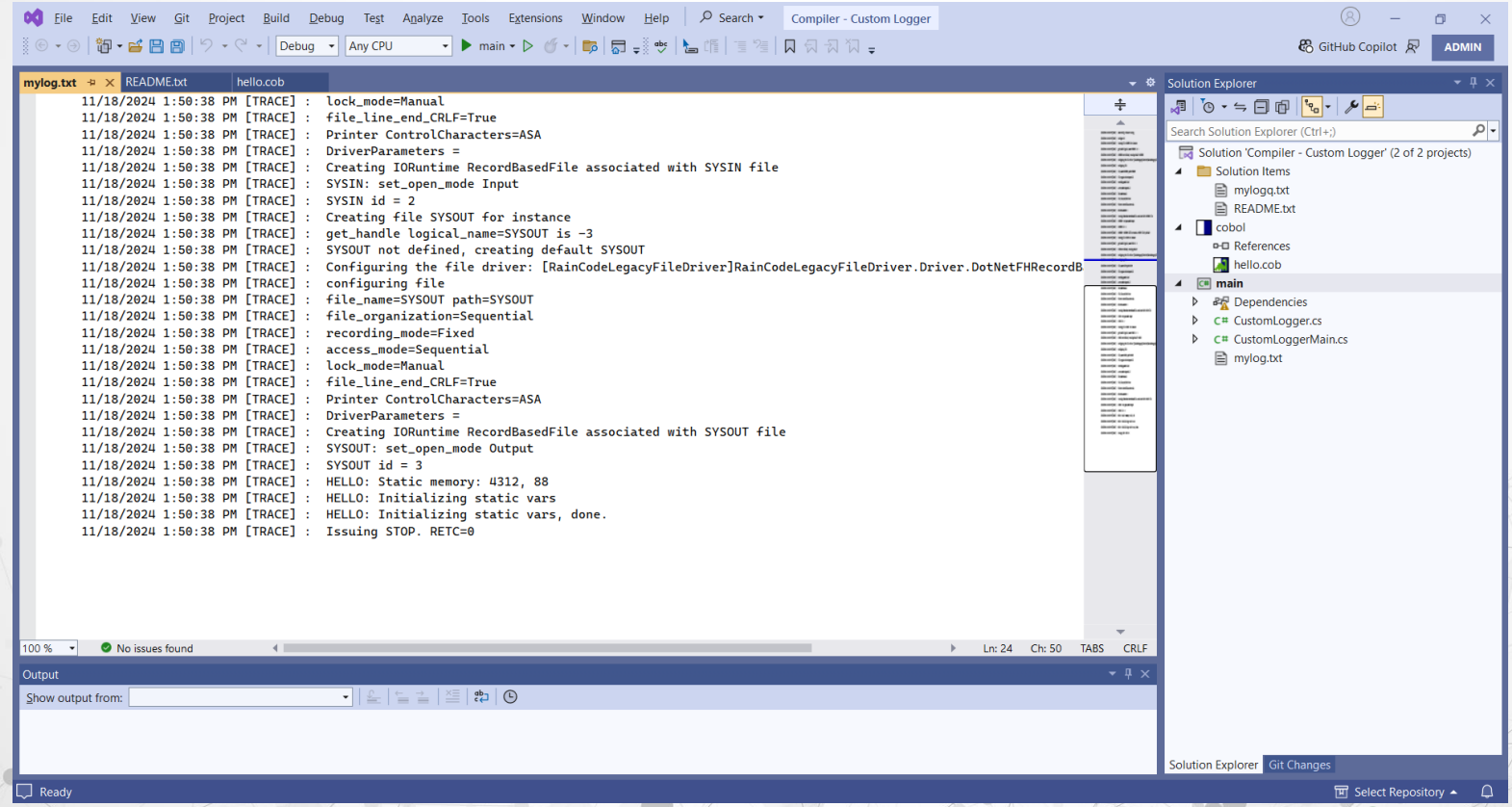


It will avoid automatic breakpoints and will keep the console window open after the program ends.

RAINCODE 360 Advanced Showcase

Custom Logger – How to see output

- You can also check the contents of mylog.txt. It should match what is shown in the console window.



Legacy with database access - Csharp main

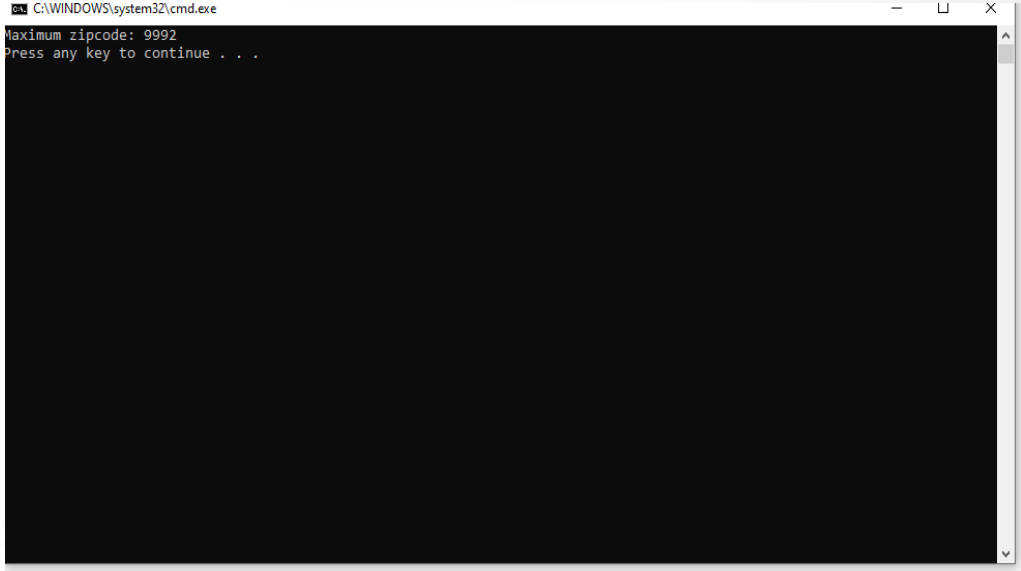
Legacy with database access - Csharp main – Introduction

- C# calling Legacy with database access using a COBOL call
- Demonstrates how a COBOL program can be called from a C# main program (a proxy)
- The C# program uses the regular COBOL call mechanism
- Demonstrates how to prepare an execution context with a database connection at the C# level
- In this demo, the legacy program fetches an item (maximum zip code) from the database, displays it on the console and exits

Legacy with database access - Csharp main – How to demonstrate

- Step 1: Open the solution from the folder Compiler - Legacy with Database Access - Csharp Main

- Step 2: Select Start without debugging (Ctrl + F5)

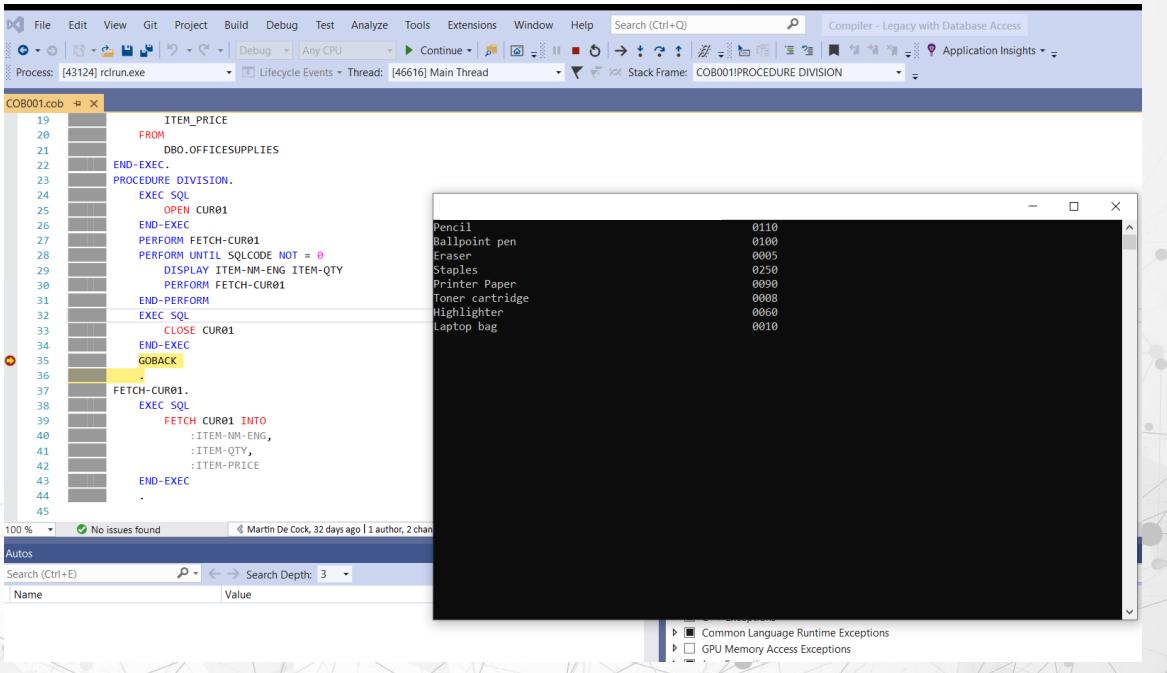


It will avoid automatic breakpoints and will keep the console window open after the program ends.

Legacy with database access

Legacy with database access

- Demonstrate how to connect a COBOL program directly to a database
- Step 1: Open the solution from the folder Compiler - Legacy with Database Access
- Step 2: Put a breakpoint on the GOBACK statement
- Step 3: Start the Raincode Debugger



RAINCODE 360 Advanced Showcase

Legacy calling Csharp with Helper class

Legacy calling CSharp with helper class

- Demonstrates how a COBOL program can transparently call C# routine (as if it were COBOL).
- Demonstrates how the C# routine can use a helper class to make sense of the data being passed to it by the COBOL program.
- This example needs a CSharp main because the CSharp routines (cshello.cs and cssquare.cs) must be added to the module dictionary before the COBOL programs can find them.
- HELLO.cob calls CSHELLO.cs without a helper class. The parameter being passed is a simple string.
- COBSQUARE.cob calls CSSQUARE.cs. The parameters are passed via the copybook W-STRUCT of COBSQUARE.cob. W-STRUCT has a complex structure.
- CSSQUARE.cs uses the generated helper class WStruct.cs to access the parameters.

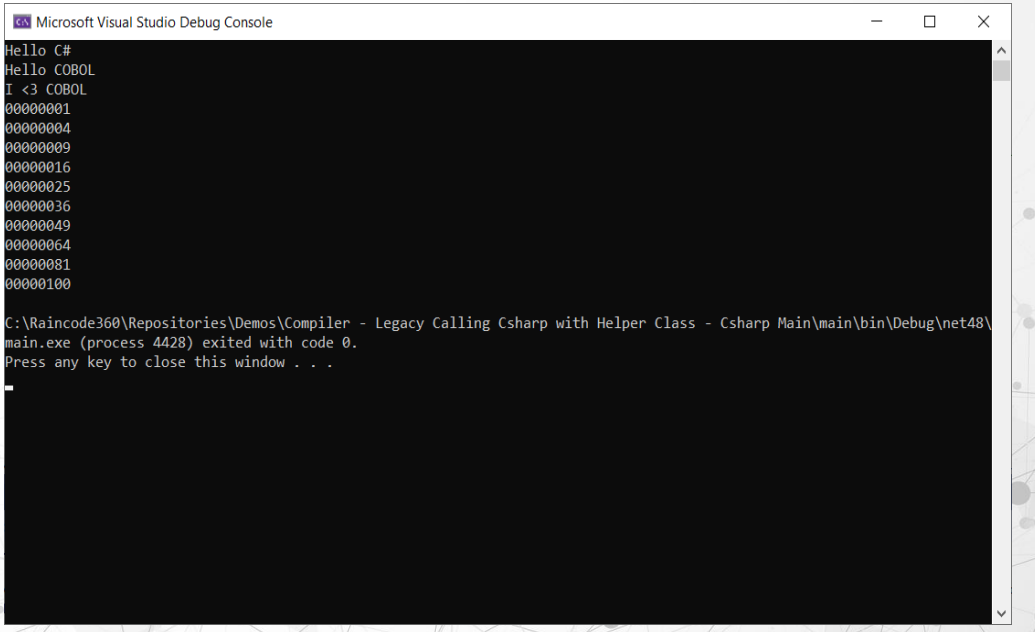
Legacy calling C# with helper class – How to demonstrate

- Step 1: Open the solution from the folder Compiler - Legacy calling CSharp with helper class

- Step 2: Start without debugging (Ctrl-F5)

Optionally, run Helper.bat in the solution directory to re-generate the helper class (WStruct.cs in the CSharp folder)

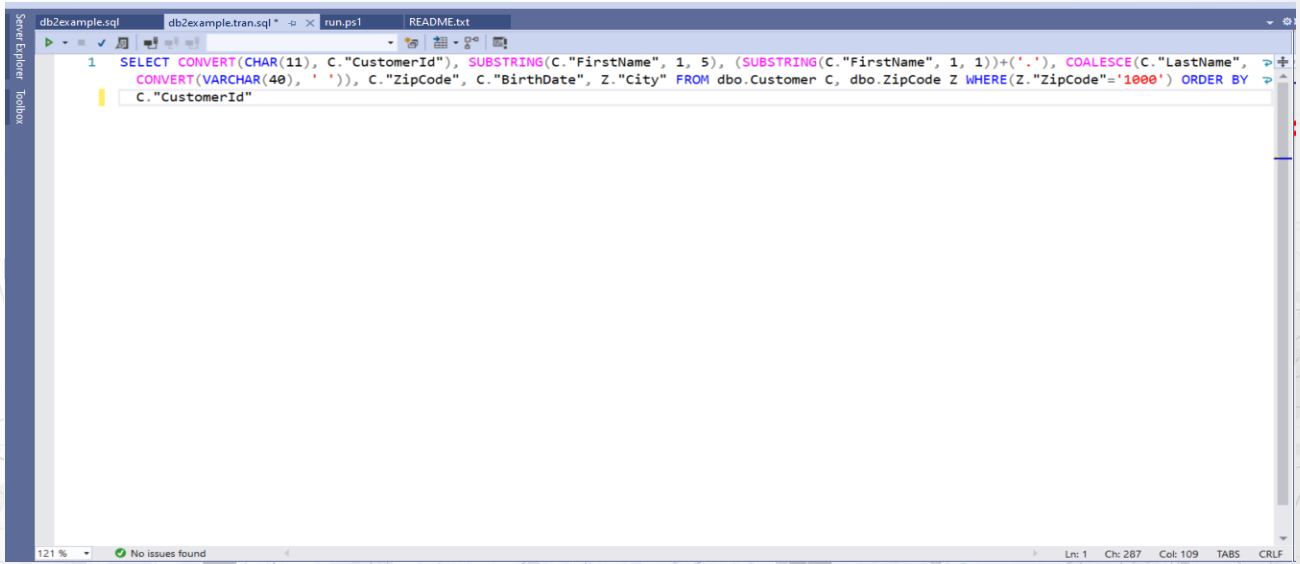
Output



Standalone SQL Conversion

Standalone SQL Conversion

- Illustrates how the compiler can be used to convert SQL statements that are not part of a program
- Useful for SQLs that appear inside scripts or JCLs
- Includes an example of script-based transformation
- Input is taken from the file db2example.sql
- The output is written to db2example.tran.sql
- The rewrite procedure's impact is not visible in the output file as they convert the input before conversion.



RAINCODE 360 Advanced Showcase

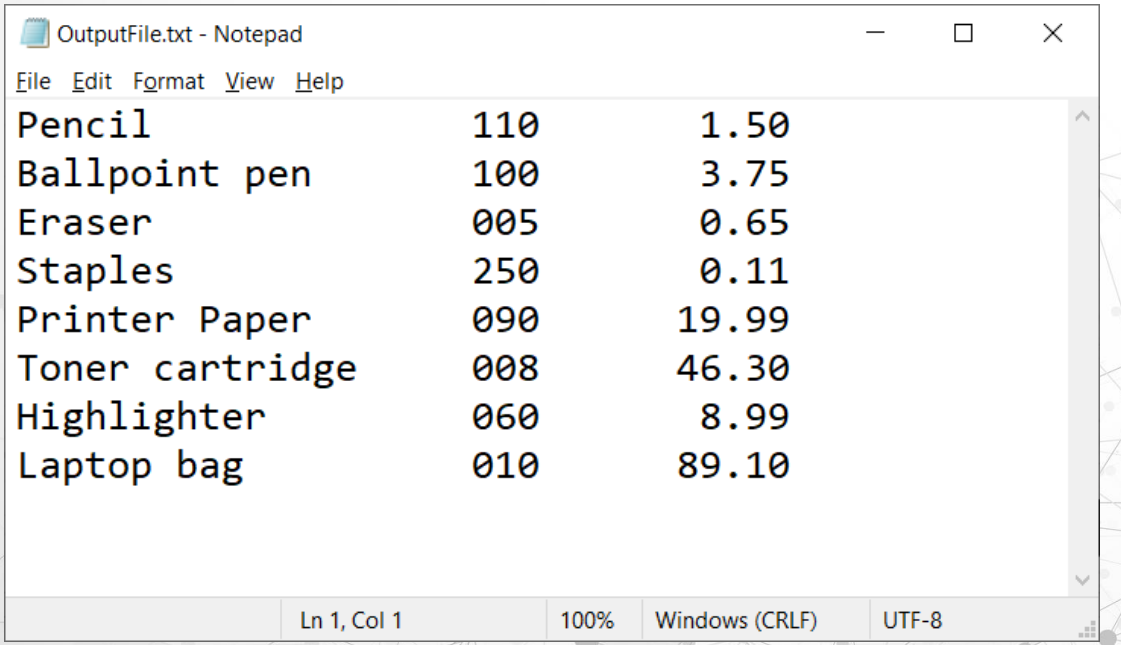
File Access via FileConfig

File Access via FileConfig - Introduction

- When not running in the context of a JCL, legacy programs need the help of a FileConfig to locate external files.
- A FileConfig is an XML file that links logical file names to the actual file names on the disk.
- A FileConfig can contain many attributes that depend on the file type.
  - The logical and physical name is sufficient for this demo
- This demo uses FileConfig to write to a file
- The main program reads a table in the database and writes this information to an output file.

File Access via FileConfig - Demo

- This demo uses FileConfig to write to a file.
- The main program reads a table in the database and writes this information to an output file.
- Press Start RainCode Debugger/F5/ Ctrl-F5 to start the demo.



The output file is defined as a solution item (OutputFile.txt)

National Character Output on Console

National Character Output on Console

- National character strings are declared as PIC N(n) USAGE NATIONAL.
- Each character is represented by two or four bytes.
- Numeric edited variables can also be declared with USAGE NATIONAL.
- Supported operations are:
  - computations,
  - output to the console,
  - viewing values in the debugger,
  - reading and writing to a database.

National Characters Output on Console

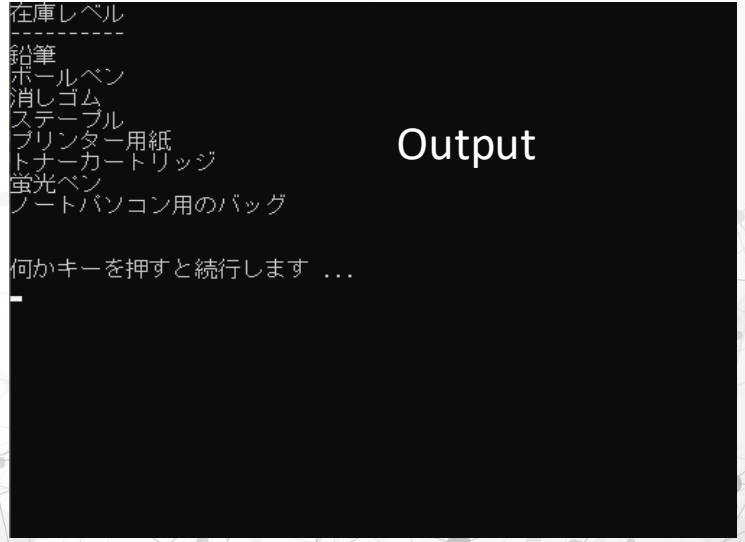
- To compile programs with the embedded national characters, **MOVE N'世界！' TO REC**, the compilation command line must contain the following option: **StringSourceFileEncoding = utf-8**

- National characters are encoded in UTF-16 big endian format.

Note: If the console cannot display Unicode characters, change the language to Japanese in [configuration/time and language/language/administrative language settings/change system locale] to make the Japanese characters visible. This requires rebooting your machine.

National Character Output on Console – Demo

- This demo shows how national characters can be written to files
- This demo reads English and Japanese descriptions from a table file
- Step 1: Open the solution from the folder Compiler – National Character Output on Console.
- Step 2: Place a breakpoint at GOBACK in COB001.cob and click on Start Raincode Debugger to execute the project.



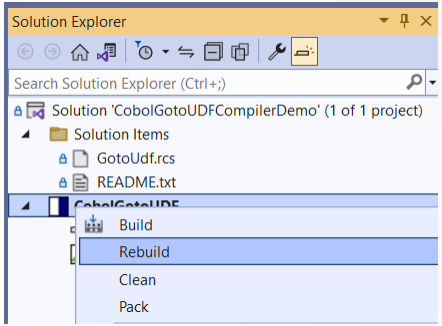
User defined function to implement a coding guideline

User defined function to implement a coding guideline

- Using the Raincode scripting language, create a user-defined function to implement a coding guideline.
- This UDF adds extra validation to the compiler to throw an error when the "GO TO" statement is encountered in the COBOL module.

User defined function – How to demonstrate in Normal flow

- Step1: Right-click on "CobolGotoUDF" from the solution explorer and select the "Rebuild" option.



CobolGotoUDF.rcproj | GOTOCBL.cob | README.txt

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.    GOTOCBL.
       ENVIRONMENT DIVISION.
        CONFIGURATION SECTION.
       DATA DIVISION.
        WORKING-STORAGE SECTION.
         01 WS-X PIC 9(01).
        LINKAGE SECTION.
       PROCEDURE DIVISION.
       PARA-0.
             DISPLAY 'IN PARA-0'
             MOVE 1 TO WS-X
             GO TO PARA-1 PARA-2 DEPENDING ON WS-X
             DISPLAY 'IN PARA-0 AGAIN'
             STOP RUN.
       PARA-1.
             DISPLAY 'IN PARA-1'.
             GOBACK.
       PARA-2.
             DISPLAY 'IN PARA-2'.
             GOBACK.
```

- Step 2: Once the build is successful. Open GOTOCBL.cob from the solution explorer and set a breakpoint at the GOBACK statement in PARA-1.

- Step 3: Start the Raincode Debugger.

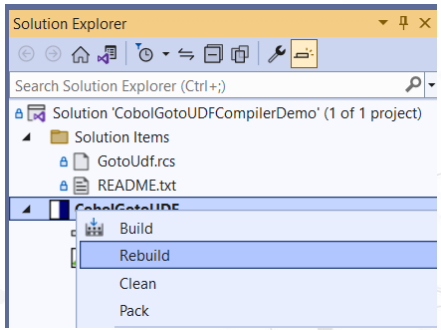
- Step 4: The execution will print "IN PARA-0" and "IN PARA-1" in consecutive lines. Close the application window to exit the debugger.

```
[2022-07-01 09:05:10.473] [1] [WARNING] [RainCode.Core.Comm
ubject to change at any time.
IN PARA-0
IN PARA-1
```

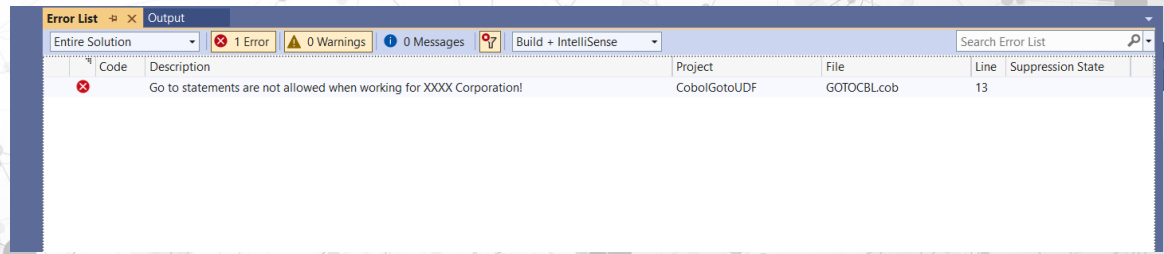
RAINCODE 360 Advanced Showcase

User defined function – How to demonstrate in Error flow

- Step 1: Click on "CobolGotoUDF" from the solution explorer.
- Step 2: From the displayed xml lines, uncomment i.e remove tags **<!-- and -->** from the line **" <UserDefinedParameters>:ScriptPaths="$(SolutionDir)" :UserDefinedValidationProc="GotoUdf.Validate"</UserDefinedParameters>** " and save the changes by pressing Ctrl +S.
- Step 3: Right click on "CobolGotoUDF" from the solution explorer and chose "Rebuild" option.



- Step 4: The compilation fails, and you see the error message "Goto statements are not allowed when working for XXXX Corporation!" in the Output tab.



Timeshift

Timeshift

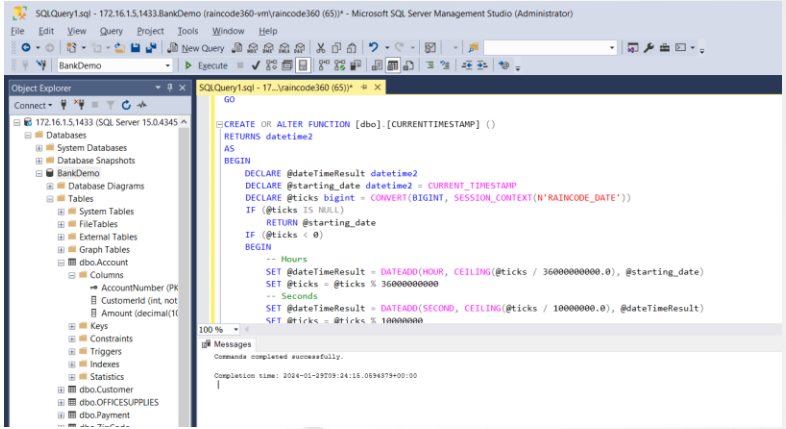
- Timeshift is a feature that enables you to run programs at any particular time. It accomplishes this by manipulating the current date and time built into the Raincode runtime.

- With some additional programming, this mechanism can be further expanded to include the current date and time in embedded SQL.

Timeshift – How to demonstrate

- Step 1: Execute the script `Stored Procedures.sql` and `Stored procedures_TimeConversion_Raincode.sql` in the Bankdemo database.

- Step 2: Start the Raincode debugger



Expected Output:

```
COBOL TS: 2032041818082831+000
SQL   TS: 2032-04-18 18:08:28.383333
```

RAINCODE 360 Advanced Showcase

RadaR

RadaR – Raincode Database Runner

- **RadaR** (Raincode Database Runner) allows the execution of the code compiled by the Raincode legacy compilers locally on the server that hosts the database. This minimizes latency and maximizes communication throughput between the program and the database.

- This demo compares the traditional client/server approach with **RadaR**

RadaR- How to demonstrate

Prerequisite:

Raincode Stack installation is to be done on the database server. For more details on installing Raincode Stack, refer to the readme available in the folder Benchmark – VsamSql with RadaR under solution items.

How to run JCL without RadaR:

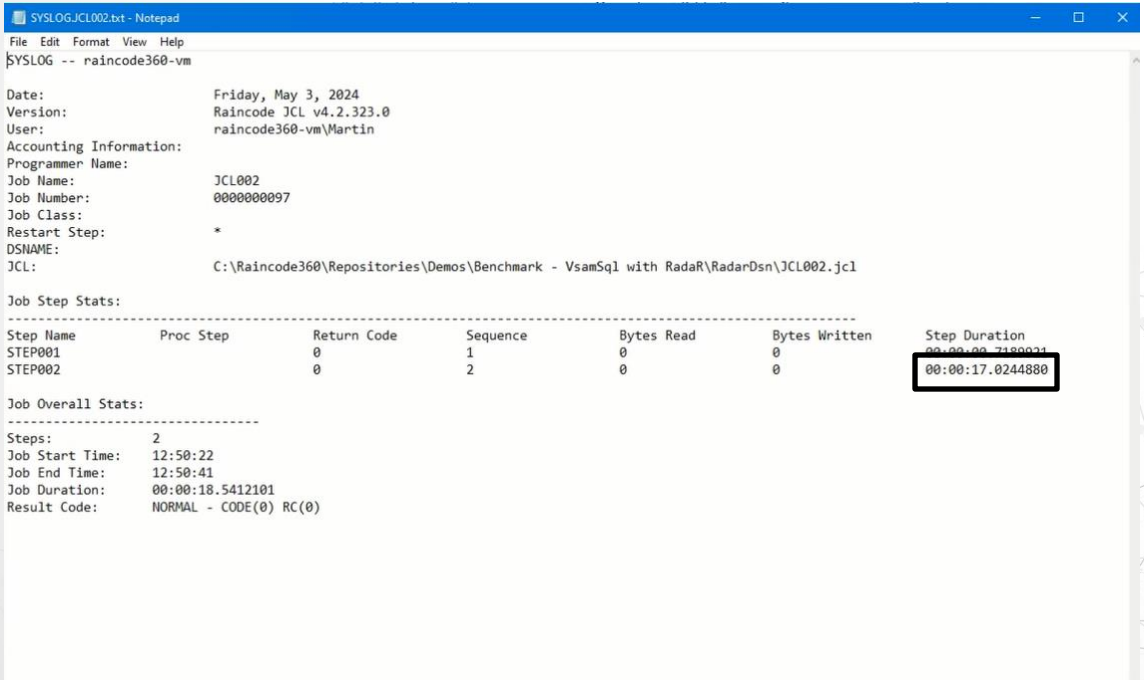
Open the solution from the Benchmark–VsamSql folder with RadaR and press the start Raincode Debugger on the main menu; it will execute the JCL without RadaR.

(*JCL runs a PGM=FSET as a second step*)

Output – JCL without RadaR

- Once the demo is executed successfully, the output file (SYSLOG.JCL002.txt) will be available in the folder SYSOUT at C:\ProgramData\Raincode\Batch.

Output:



The output without RadaR shows that both steps were executed successfully, and the second step took 17 seconds.

Output - JCL with RadaR

How to run JCL with RadaR:

To make the JCL run on the database server instead of the client, open the PowerShell from Tools->Command Line -> Developer PowerShell and run **FSET.ps1.**

Expected Output:

SYSLOG.JCL002.txt - Notepad

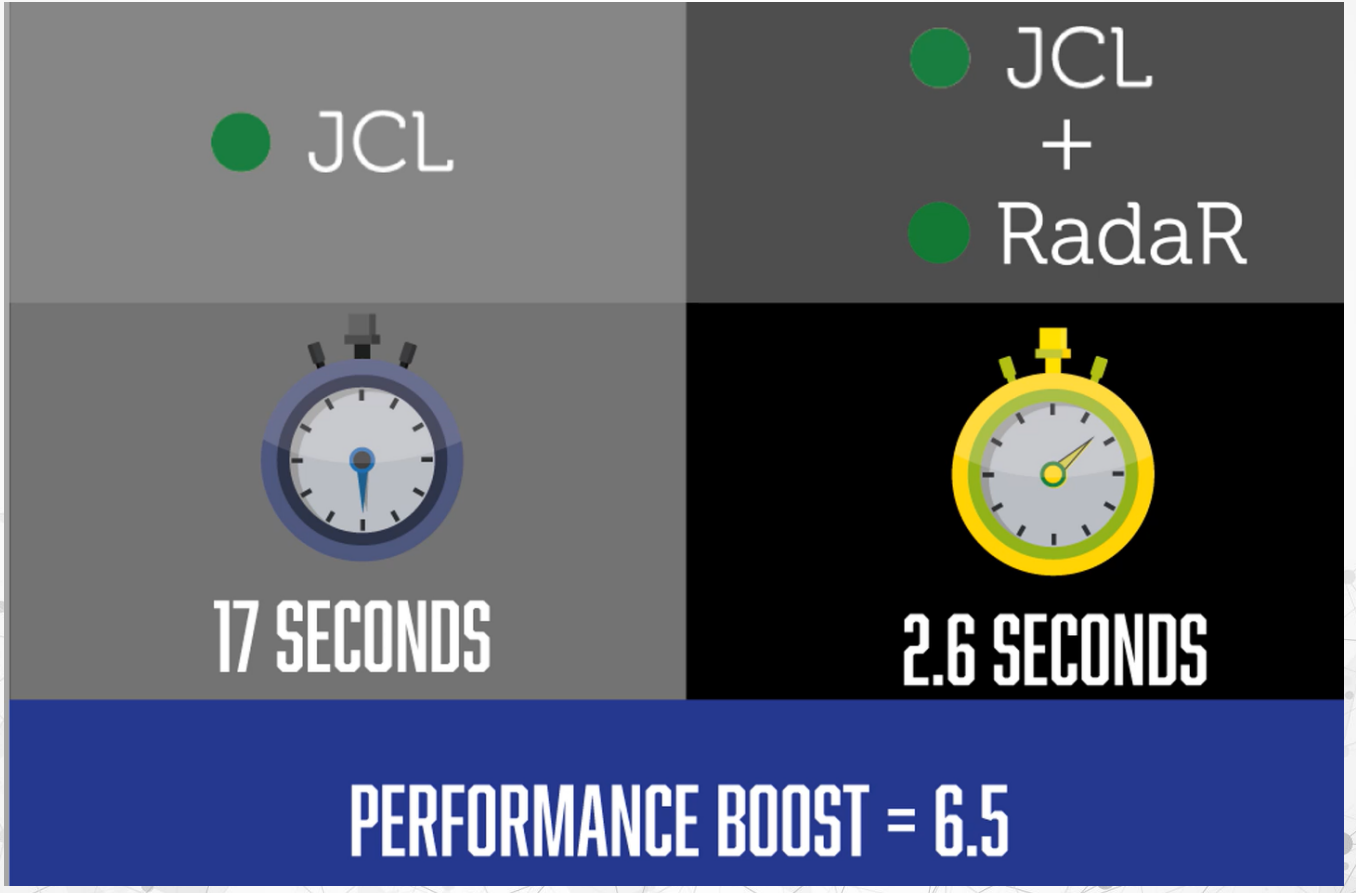
File Edit Format View Help

```
SYSLOG -- raincode360-vm

Date:                   Friday, May 3, 2024
Version:                Raincode JCL v4.2.323.0
User:                   raincode360-vm\Martin
Accounting Information:
Programmer Name:
Job Name:               JCL002
Job Number:             0000000098
Job Class:
Restart Step:           *
DSNAME:
JCL:                    C:\Raincode360\Repositories\Demos\Benchmark - VsamSql with RadaR\RadarDsn\JCL002.jcl

Job Step Stats:
-------------------------------------------------------------------------------------------------------
Step Name       Proc Step       Return Code     Sequence        Bytes Read      Bytes Written   Step Duration
STEP001                         0               1               0               0               00:00:00.7698388
STEP002                         0               2               0               0               00:00:02.6058503

Job Overall Stats:
-----------------------------
Steps:          2
Job Start Time: 13:02:41
Job End Time:   13:02:45
Job Duration:   00:00:04.1941841
Result Code:    NORMAL - CODE(0) RC(0)
```

The output with RadaR shows that both steps were executed successfully, and the second step took 2.6 seconds.

RAINCODE 360 Advanced Showcase

RadaR - Comparison of outputs



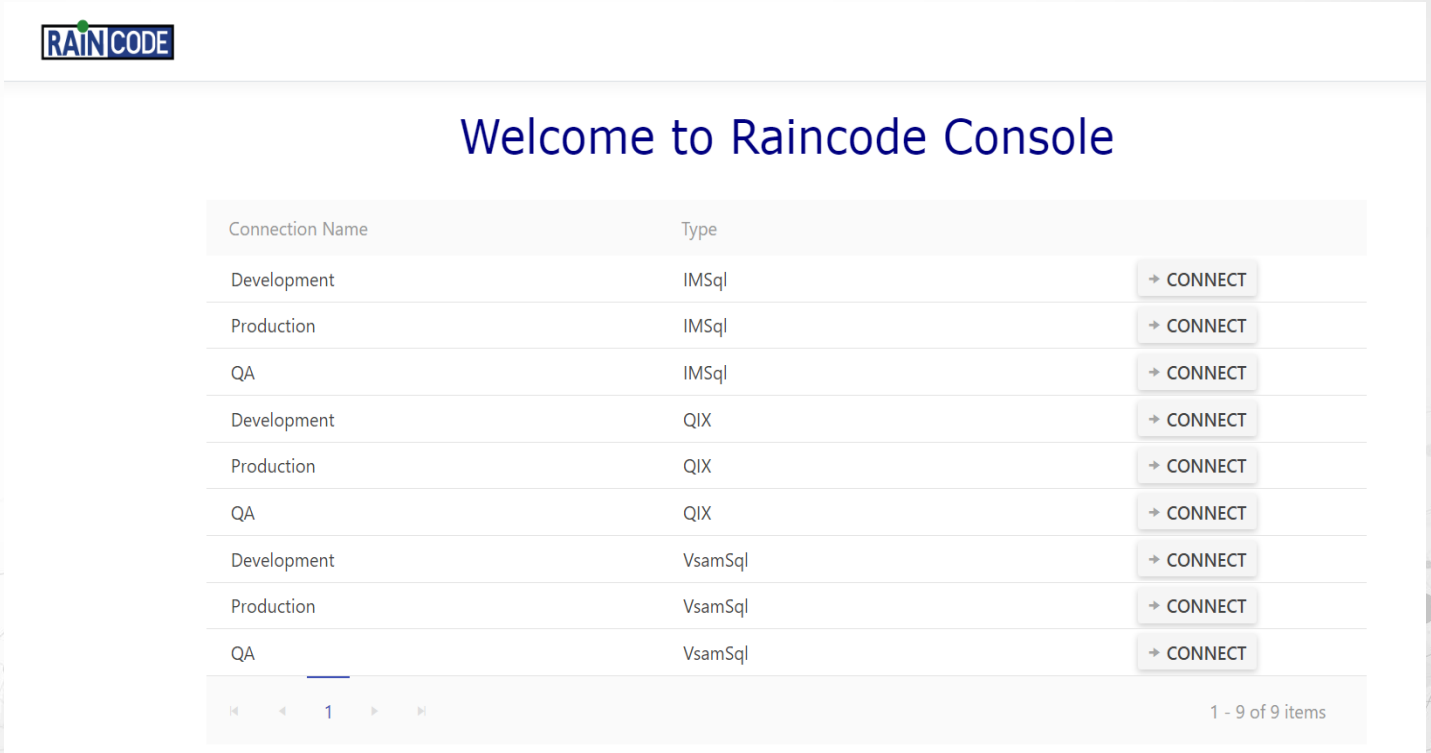
Raincode Console

RAINCODE 360
Advanced Showcase

Raincode Console

- Raincode Console is a tool that allows you to manage IMS, QIX and Vsam Sql systems in one place.

- Double-click on the shortcut (Raincode console) placed on the desktop, and it will launch the tool in the web browser

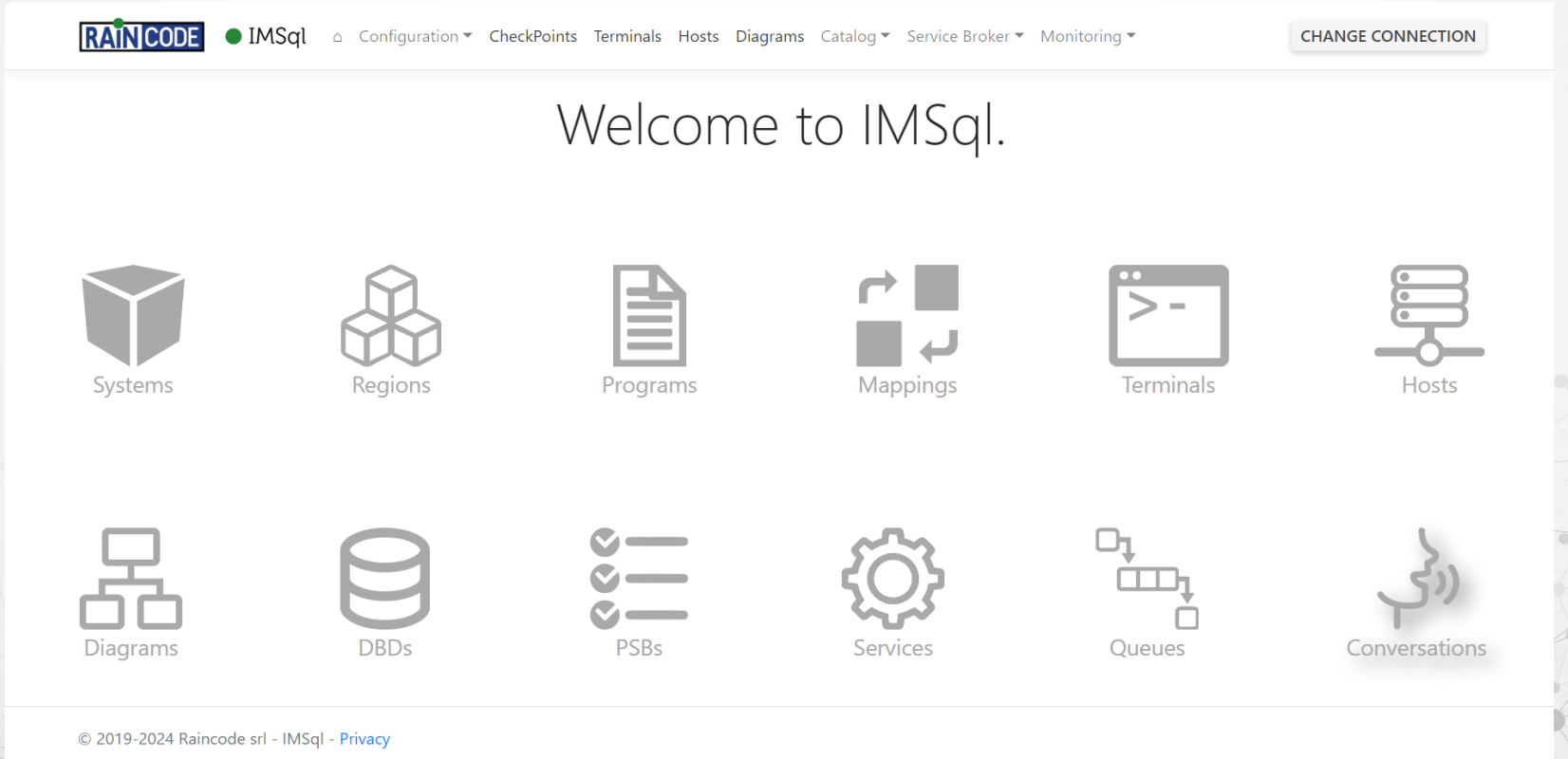


- For more details, refer to the Raincode Console documentation.

RAINCODE 360 Advanced Showcase

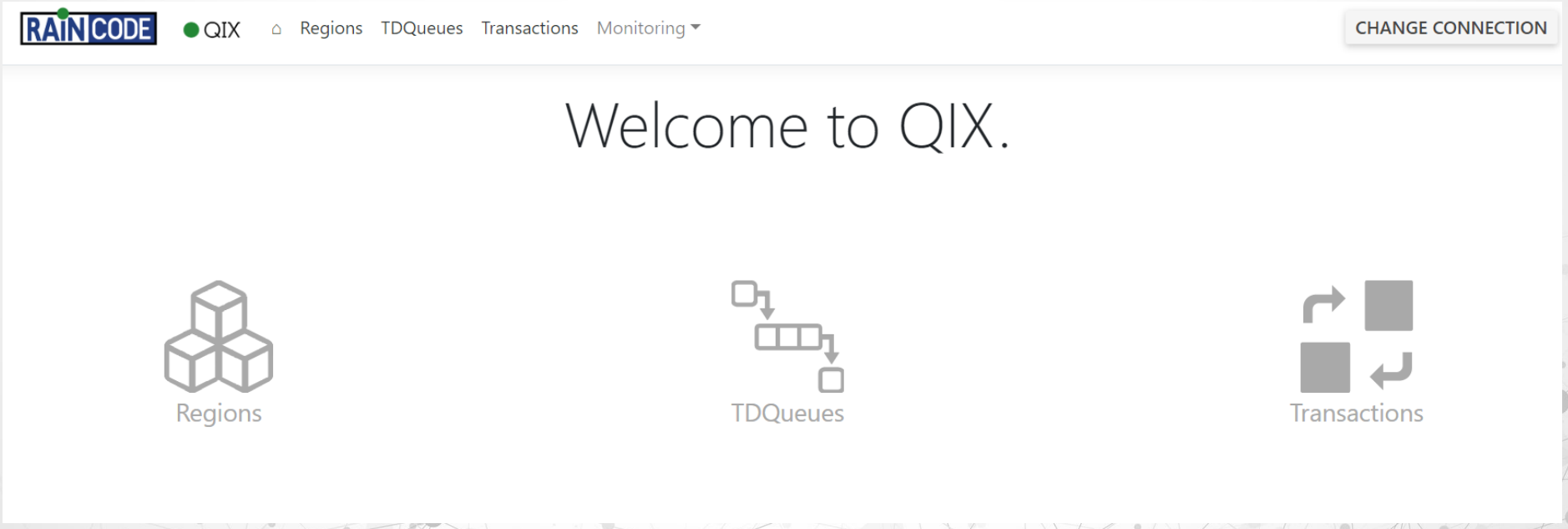
Raincode Console - IMSql

- IMSql panel allows you to manage an IMS system. It allows you to view and manage IMS/DB and IMS/TM



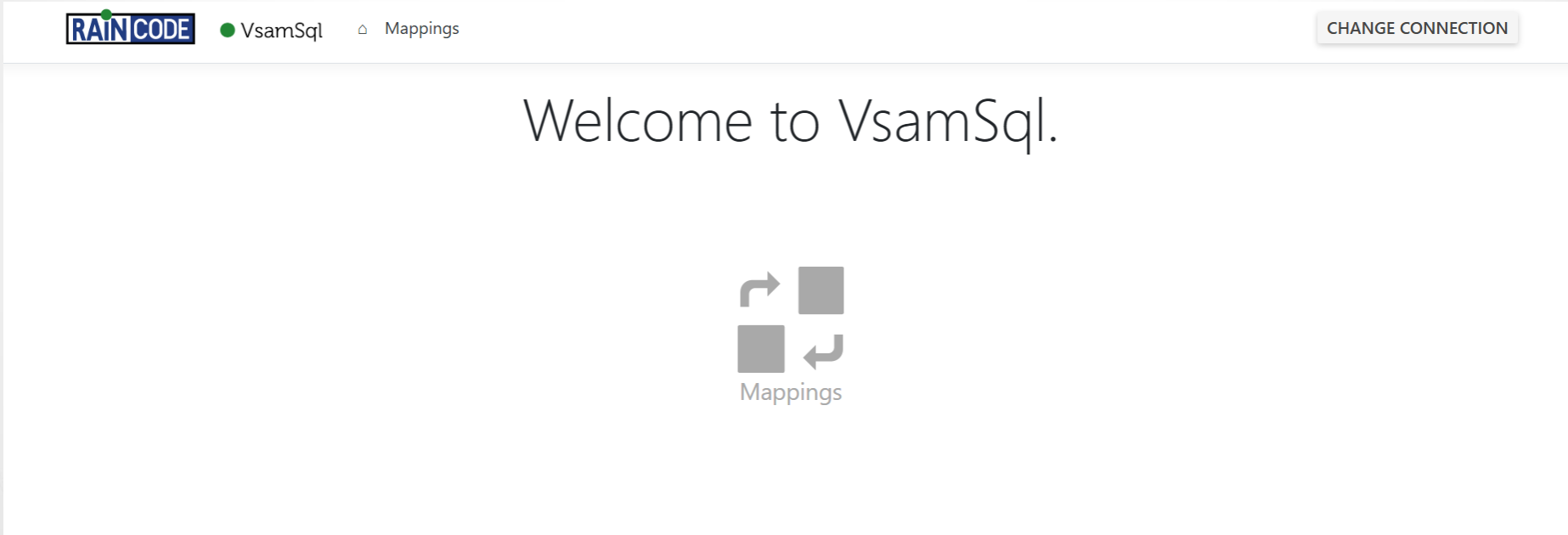
Raincode Console - QIX

- QIX panel allows you to manage a QIX system such as configuring a region, transactions, etc.



Raincode Console - Vsam Sql

VSAM Sql panel allows you to manage a VSAM system, such as viewing the dataset and table mappings and deleting the datasets and their data.



Raincode Batch: JCL Interpreter

Architecture

Scaffolding of input parsers
(support for Autoedit, for instance).

Windows file system locking primitives for synchronization

Catalog management as a reusable component

Comprehensive DF-SORT utility

Parser generator for additional utilities

Scheduler

Externalized

Focus on scale and flexibility

Support for multi-year migration
when big bang is not an option

```
000100 //IEFBR141  JOB 'AK99','AVINASH',NOTIFY=&SYSUID
000200 //*******************************************************************
000300 //*THIS UTILITY USED TO ALLOCATE OR DEALLOCATE A DATASET
000400 //*CREATES BOTH PS AND PDS FILES ALSO DELETES SAME
000410 //*******************************************************************
000420 //MYPROC PROC ----->MYPROC IS PROCEDURE NAME THEN PROC THIS IS SYNTAX
000500 //DELETE    EXEC PGM=IEFBR14
000600 //SYSOUT    DD SYSOUT=*
000700 //SYSPRINT  DD SYSOUT=*
000800 //SYSUT1    DD DSN=AK999K.AVINASH.PS4,
000900 //          DISP=(NEW,CATLG,DELETE),
000910 //          DCB=(LRECL=80,BLKSIZE=8000,RECFM=FB),
000920 //          SPACE=(TRK,(2,3),RLSE)
000930 //          PEND ---->END OF PROC.
000940 *******************************************************************
000950 //STEP2 EXEC MYPROC  ---->HERE I AM CALLING MY INSTREAM PROC IN SAME JCL
001000 //SYSIN     DD DUMMY
001100 //
```

Raincode Batch – Introduction

JCL Contains:

- IKJEFT01
- IDCAMS
- SORT
- COBOL Program:
  - Launched by IKJEFT01
  - Database access
  - Uses CICS LINK to access a PL/I program
- Legacy calling assembler
- C# launcher
- JCL examples:
  - DSNTIAUL
  - IEBGENER
  - DSNUTLIB load and unload

Raincode JCL Interpreter

Demos:

- Legacy Calling Assembler
- Legacy with Database Access –EBCDIC Version
- SORT and IKJEFT01
- DSNTIAUL
- DSNUTILB
- Batch – File Types
- Generation Data Group
- Partitioned Dataset
- Catalog Explorer
- Automatic Restart
- Syncsort Translator
- Files with National Character
- FBA with EBCDIC
- zTrieve
- Legacy with MultiDatabase Access
- Unload MultiDatabase

Legacy Calling Assembler

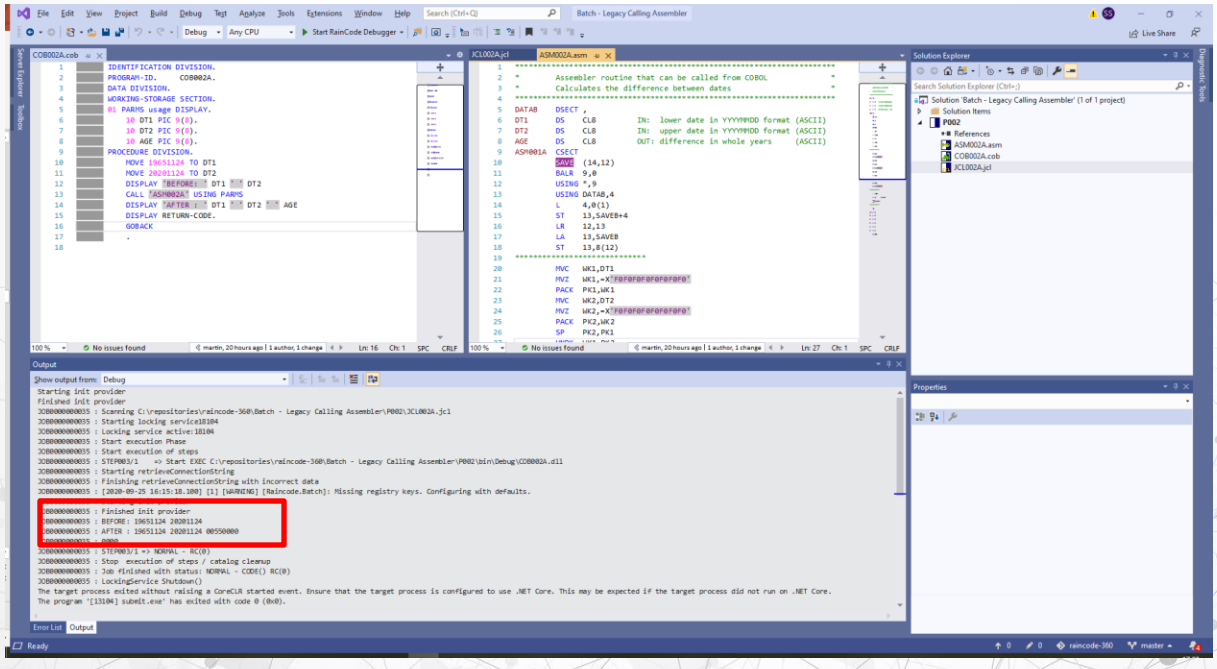
Legacy Calling Assembler

- This demo is an example of a JCL executing a COBOL program with no database access.
- The COBOL program illustrates how to call a subroutine written in Assembler.
- The assembler routine calculates the difference in years between the two dates that are passed as parameters by the COBOL main program.

Legacy Calling Assembler – How to demonstrate

- Step 1: Open the solution from the folder Batch - Legacy Calling Assembler

- Step 2: Start Raincode Debugger, and after successful debugging, it will calculate the difference between two dates.



RAINCODE 360 Advanced Showcase

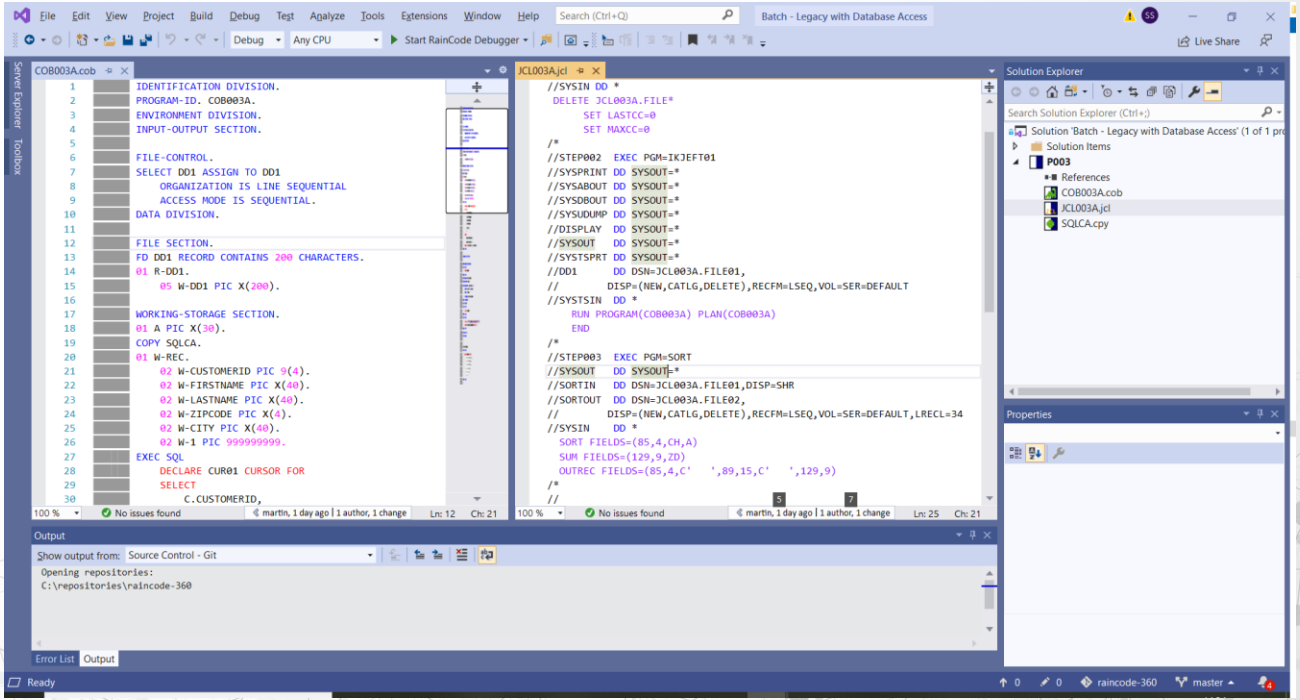
SORT and IKJEFT01

SORT and IKJEFT01

- Connect a COBOL program to the BANKDEMO database via IKJEFT01
- Example: sort then aggregate the number of existing customers per zip code.

- Step 1: Open the solution from the folder Batch – Legacy with Database Access

- Step 2: Press Start Raincode Debugger to compile and run the project

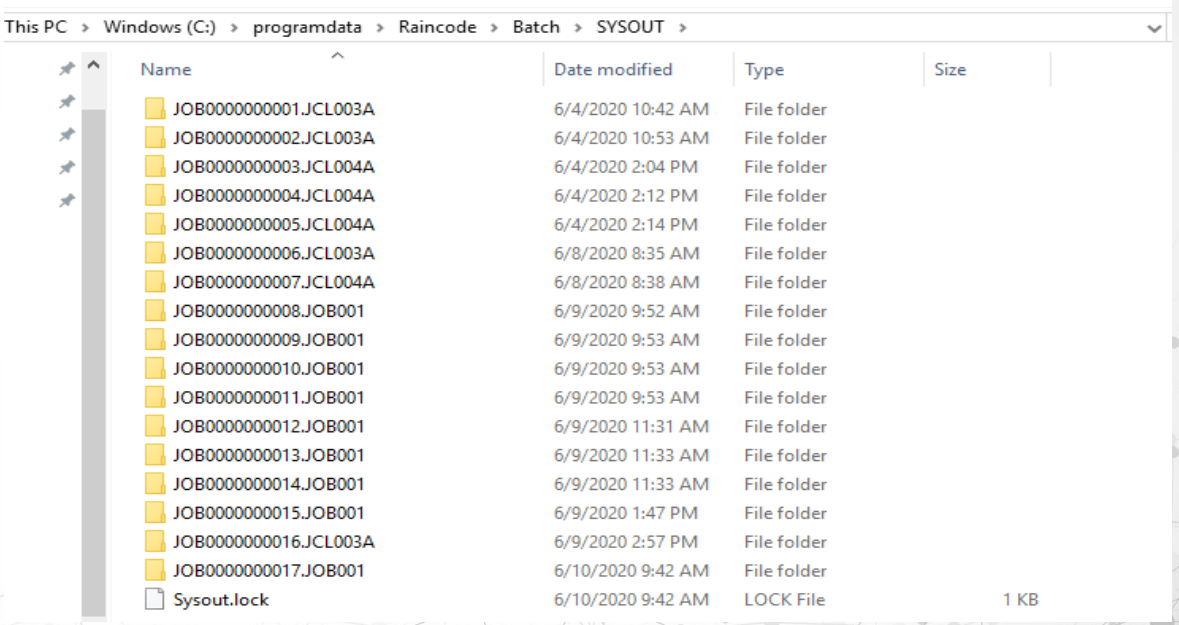


RAINCODE 360
Advanced Showcase

SORT and IKJEFT01 – Where to See Output Files and Job Queue

- Output and job queue created will be available at C:\ProgramData\Raincode\Batch

- Navigate to the DefaultVolume folder to the see output files
- Navigate to the SYSOUT folder to see the Job Queue
  - Output of the job that just ran, together with the previous run's output is available here.
  - The highest number represents the most recent run.



Legacy with Database Access – EBCDIC Version

Legacy with Database Access – EBCDIC version

- This demo produces file output in EBCDIC
- This demo is a modified version of the demo 'Batch - Legacy with Database Access'
- Differences:
  - The Encoding is EBCDIC rather than ASCII
  - The files are FB rather than LSEQ

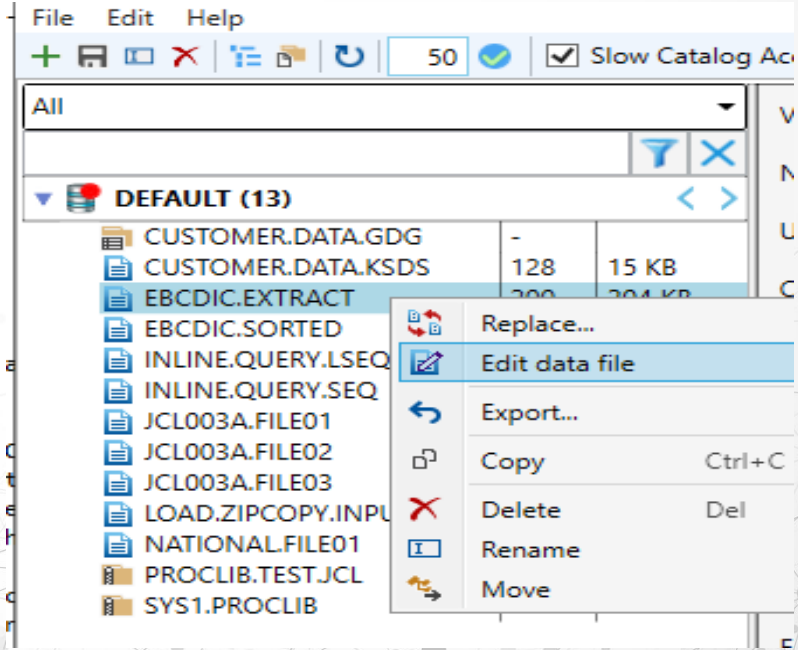
- The sort utility is installed in ASCII mode, so the character literal C'   ' (three spaces) in the ASCII version was replaced by X'404040' to produce readable EBCDIC.

Legacy with Database Access – EBCDIC version –How to demonstrate

- Step 1: Open the solution from the folder Batch - Legacy with Database Access - EBCDIC Version
- Step 2: Press Start Raincode Debugger to compile and run the project

You can examine the output file (EBCDIC.EXTRACT and EBCDIC.SORTED) using the Catalog Explorer

- Step 3: Open the Catalog Explorer by double click on the Catalog Explorer shortcut placed on the desktop
- Step 4: Right-Click the output file and select Edit data file from the context menu.



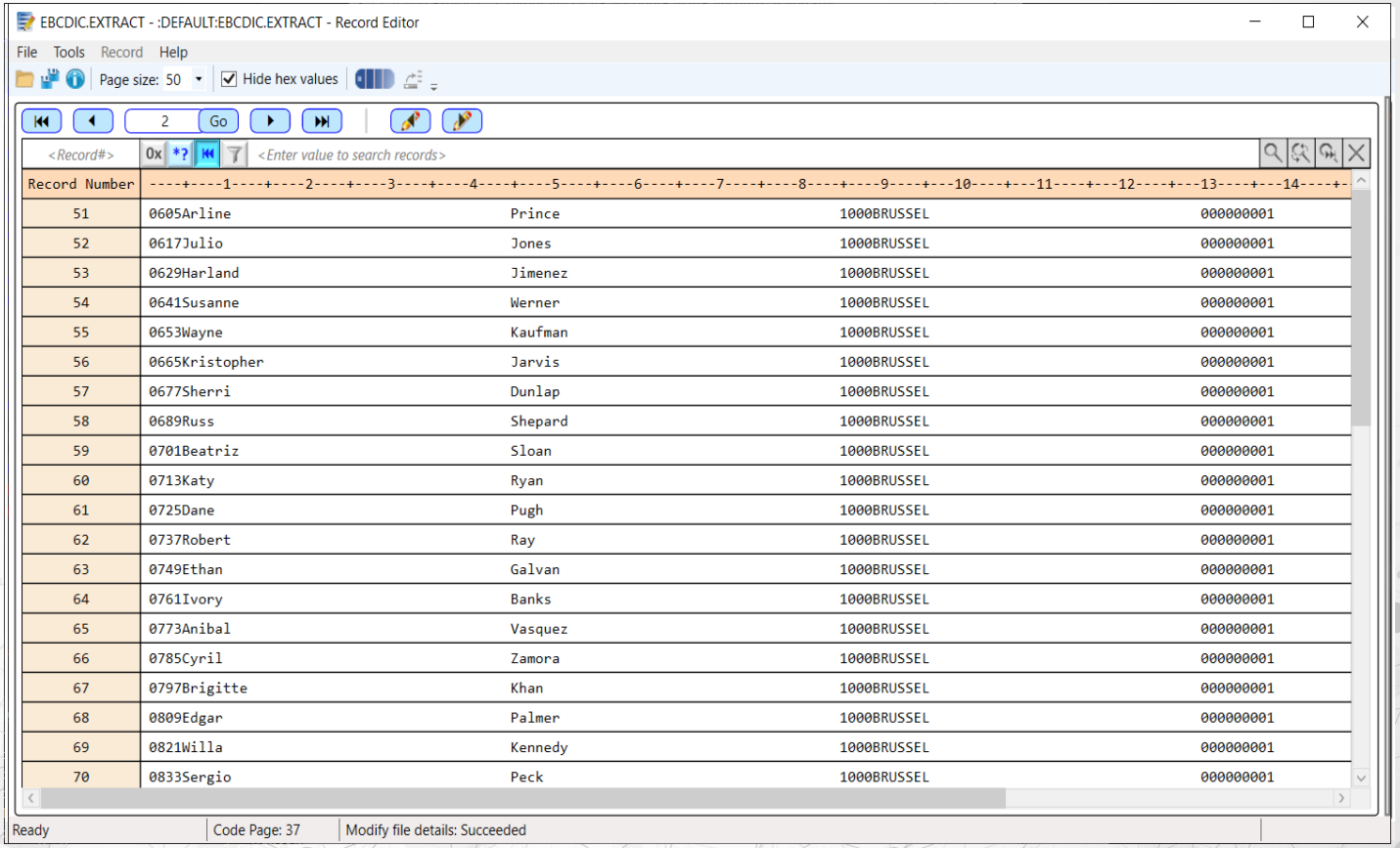
The output file will open in the Record Editor

RAINCODE 360
Advanced Showcase

Legacy with Database Access – EBCDIC version

- Step 5: Click on the File details icon and select Code Page as 37 from the drop-down menu of Encoding

- Step 6: Press Reload File to load the file in EBCDIC.



Record Number	----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----+----9----+---10---+---11---+---12---+---13---+---14---+-			
51	0605Arline	Prince	1000BRUSSEL	000000001
52	0617Julio	Jones	1000BRUSSEL	000000001
53	0629Harland	Jimenez	1000BRUSSEL	000000001
54	0641Susanne	Werner	1000BRUSSEL	000000001
55	0653Wayne	Kaufman	1000BRUSSEL	000000001
56	0665Kristopher	Jarvis	1000BRUSSEL	000000001
57	0677Sherri	Dunlap	1000BRUSSEL	000000001
58	0689Russ	Shepard	1000BRUSSEL	000000001
59	0701Beatriz	Sloan	1000BRUSSEL	000000001
60	0713Katy	Ryan	1000BRUSSEL	000000001
61	0725Dane	Pugh	1000BRUSSEL	000000001
62	0737Robert	Ray	1000BRUSSEL	000000001
63	0749Ethan	Galvan	1000BRUSSEL	000000001
64	0761Ivory	Banks	1000BRUSSEL	000000001
65	0773Anibal	Vasquez	1000BRUSSEL	000000001
66	0785Cyril	Zamora	1000BRUSSEL	000000001
67	0797Brigitte	Khan	1000BRUSSEL	000000001
68	0809Edgar	Palmer	1000BRUSSEL	000000001
69	0821Willa	Kennedy	1000BRUSSEL	000000001
70	0833Sergio	Peck	1000BRUSSEL	000000001

Syncsort Translator

Syncsort Translator

- The Raincode Syncsort Translator allows for SORT and ICETOOL steps in your JCL to be executed by Syncsort.
- Once installed, the Syncsort Translator can be invoked by invoking XSORT/SORTDTL to translate the SORT syntax, or by invoking ICETRANS/ICEDTL to translate the ICETOOL syntax. No further changes to the JCL are needed.
- Syncsort is a third-party sort utility that is not installed as part of the demo.
- The demo is a single JCL with each step illustrating a different sort method.

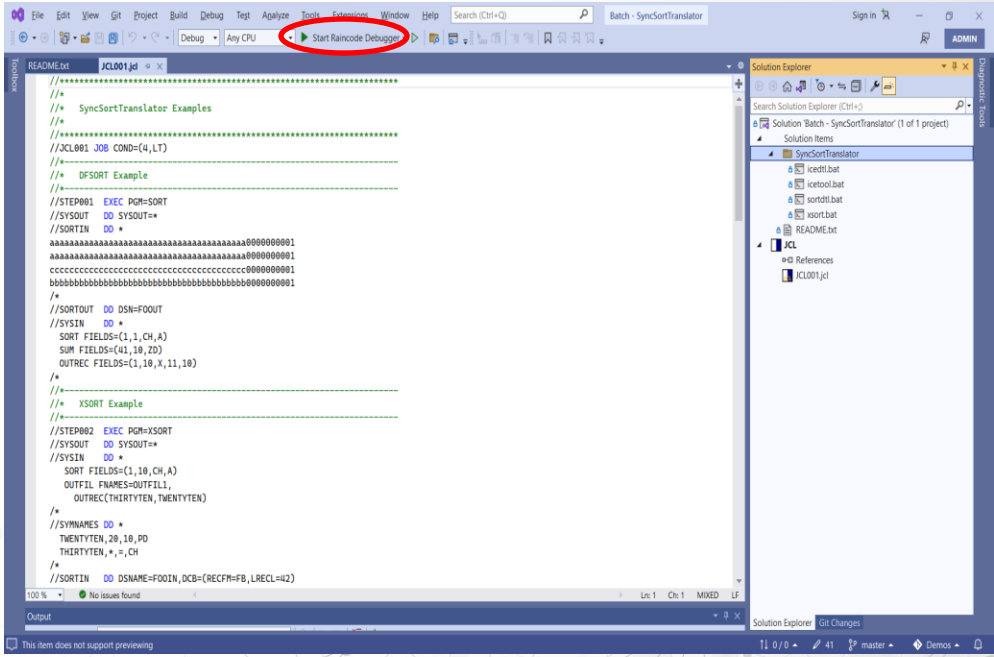
Step	Program	Input Syntax	Sort Type	Target Syntax
1	SORT	DFSORT	basic	DFSORT (no translation)
2	XSORT	DFSORT	basic	Syncsort
3	SORTDTL	same as 2	basic	Syncsort DTL
4	SORTDTL	DFSORT	join	Syncsort DTL
5	ICETOOL	ICETOOL	splice	Syncsort
6	ICEDTL	same as 5	splice	Syncsort DTL

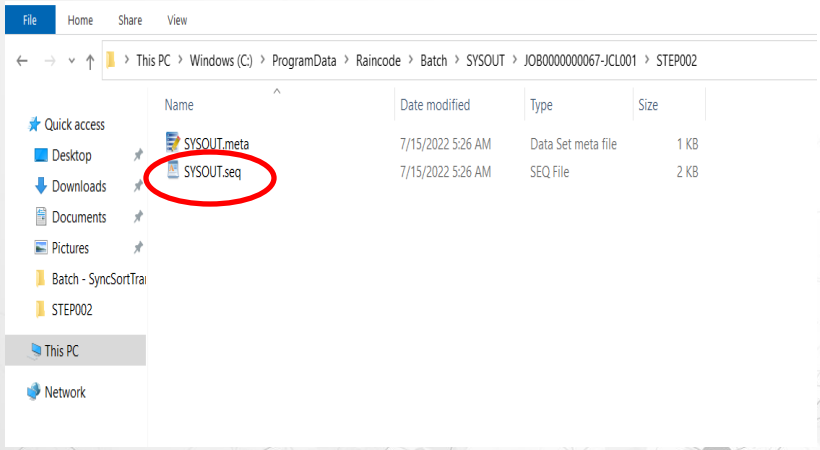
Syncsort Translator – How to demonstrate

- Step 1: Open the solution from the folder Batch - SyncSortTranslator

- Step 2: Press Start Raincode Debugger to compile and run the project

- Step 3: Find the job output in the folder *C:\ProgramData\Raincode\Batch\SYSOUT*





RAINCODE 360
Advanced Showcase

Syncsort Translator – How to demonstrate

- Step 4: Open the job output SYSOUT.seq, and you can find the location for the translated output file

SYSOUT.seq

C: > ProgramData > Raincode > Batch > SYSOUT > JOB0000000067-JCL001 > STEP002 > SYSOUT.seq

```
1  "Starting translator"
2  Starting SyncsortTranslator : -FromJCL=true -WorkDir=workspace_directory -OutputDir=C:\Users\RAINCO~1\AppData\Local\Temp\2\RC_SSXLate
3  defaultRecordLength is Different of 0 This will not permit detection of non specified LREC VALUE. Value is Ignored
4  Starting init provider
5  Finished init provider
6  "Translation finished"
7  "Starting SyncSort"
8  "SyncSort Finished"
9  "Step Done"
10
```

- Translated Output

JCL001-STEP002_2.dme.001.txt - Notepad

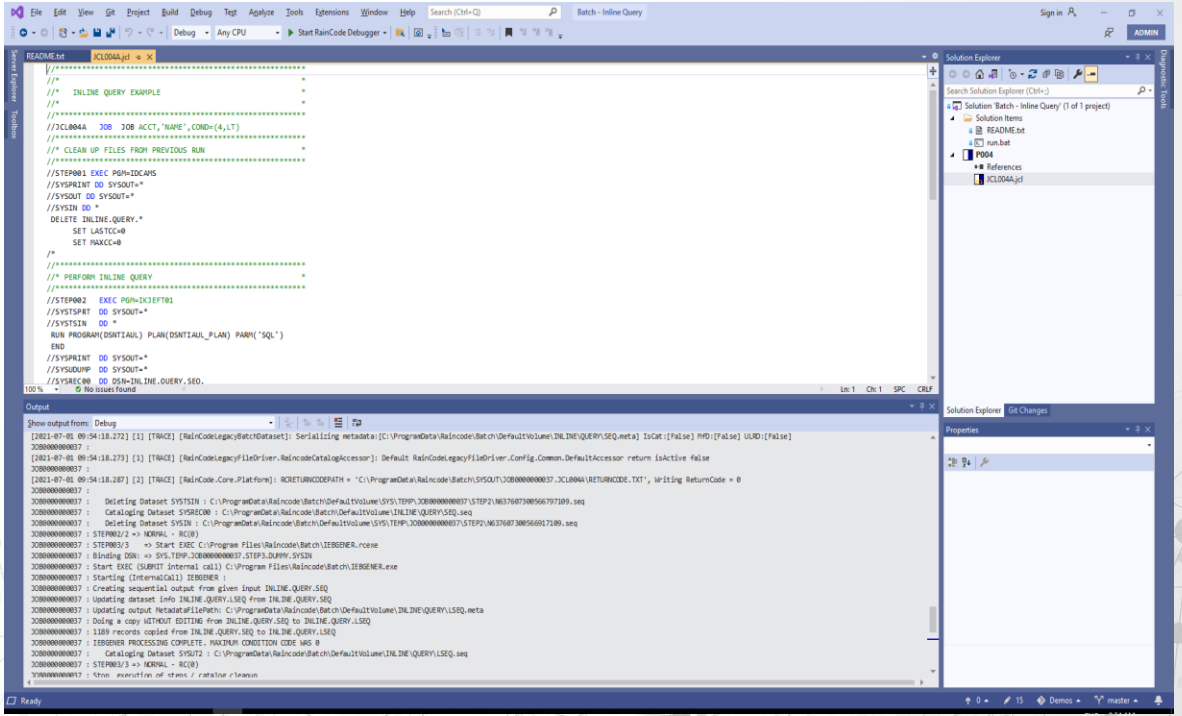
File Edit Format View Help

```
/FIELDS Src_0 1 10 CHARACTER
/INFILE C:\ProgramData\Raincode\Batch\DefaultVolume\FOOIN.seq MAINFRAMEFIXED 42
/KEYS Src_0 ASCENDING
/COLLATINGSEQUENCE DEFAULT EBCDIC
/OUTFILE C:\ProgramData\Raincode\Batch\DefaultVolume\FOOUT.seq MAINFRAMEFIXED 42 OVERWRITE
/END
```

DSNTIAUL

INLINE Query with DSNTIAUL

- Illustrate the use of DSNTIAUL in a JCL
- Step 1: Open the solution from the folder Batch - Inline Query
- Step 2: Press Start Raincode Debugger to compile and run the project

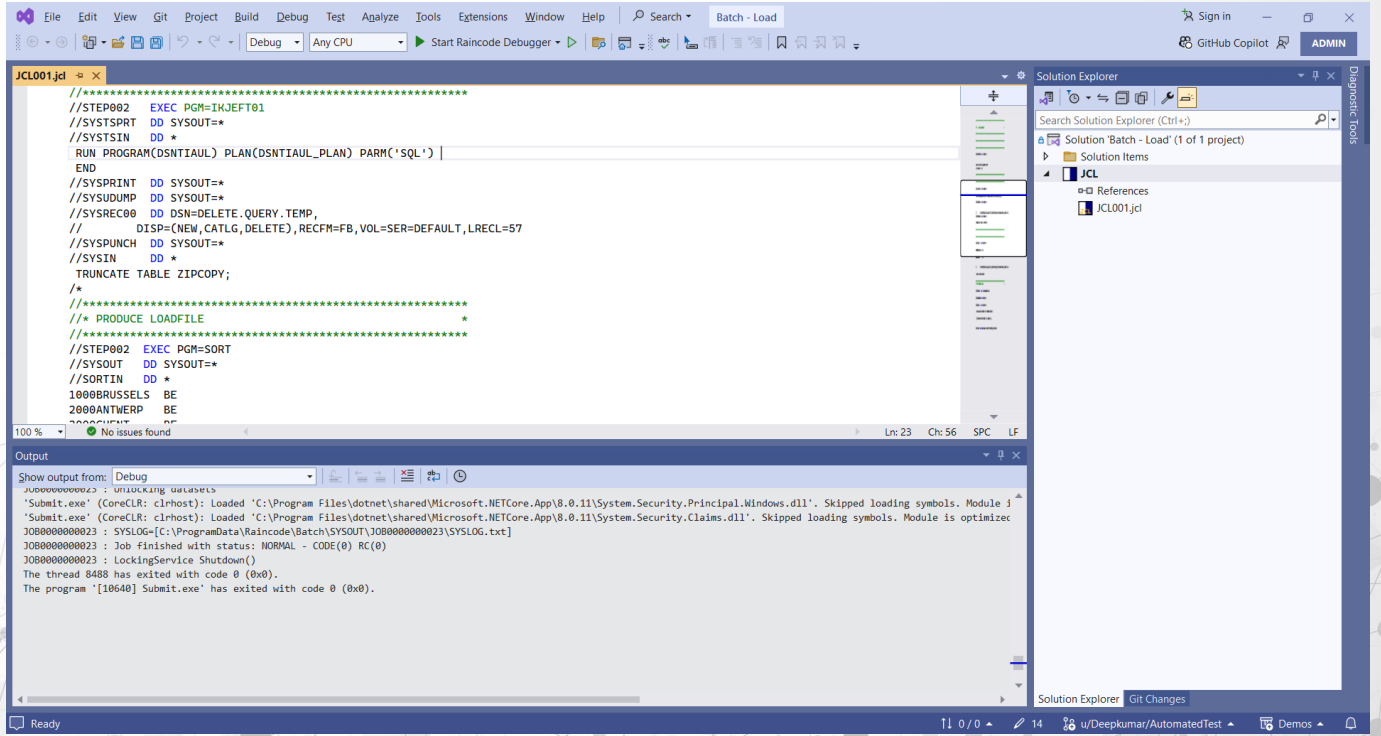


Output file created will be available at:
C:\ProgramData\Raincode\Batch\DefaultVolume\Inline

DSNUTILB

Loading Data with DSNUTILB

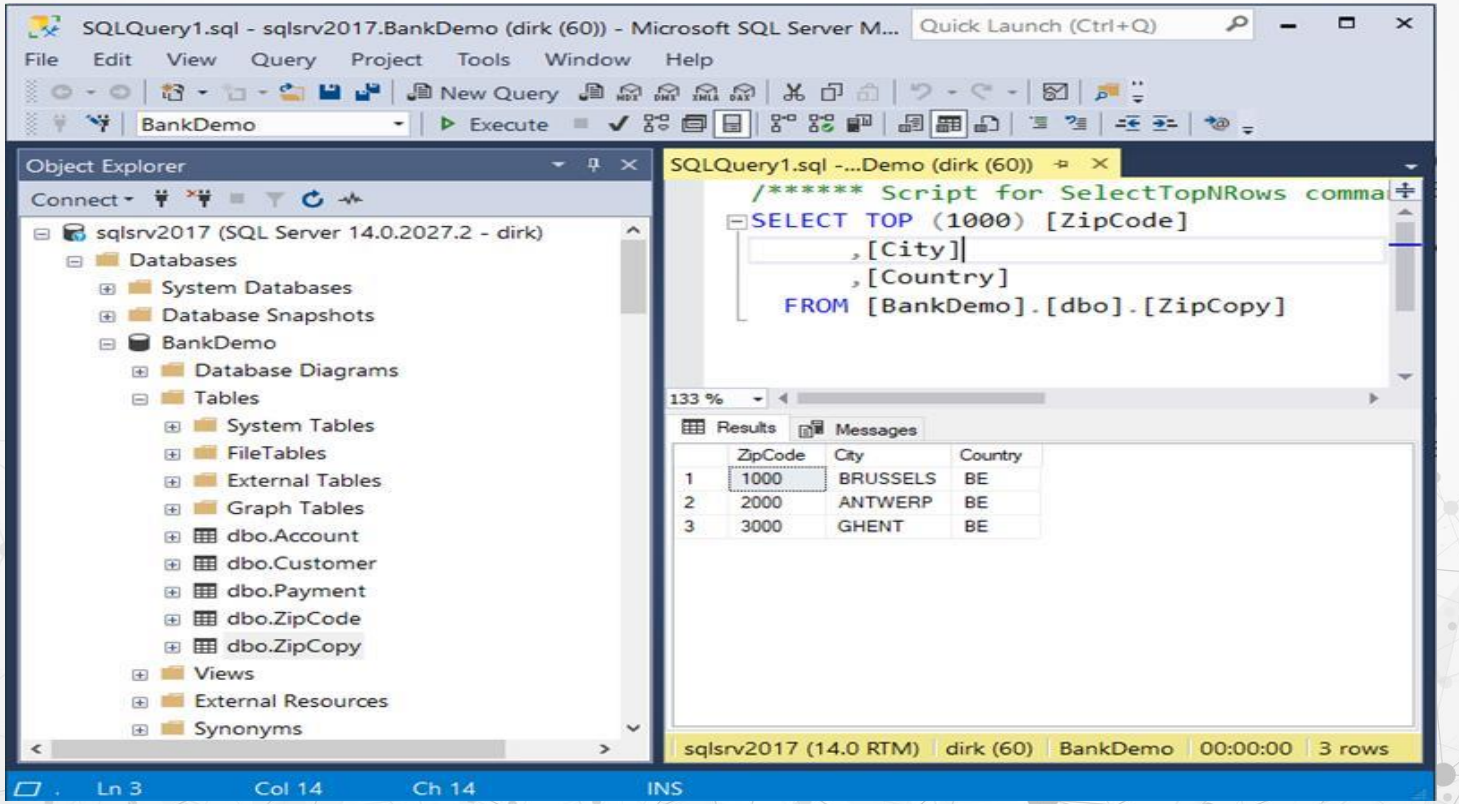
- Illustrate the use of DSNUTILB for loading data in a JCL context. This demo copies the contents from a sequential file into a database table.
- Step 1: Open the solution from the folder Batch - Load
- Step 2: Press Start Raincode Debugger to compile and run the project



RAINCODE 360 Advanced Showcase

Loading Data with DSNUTILB

- After the successful execution, the content will be copied in the database table.

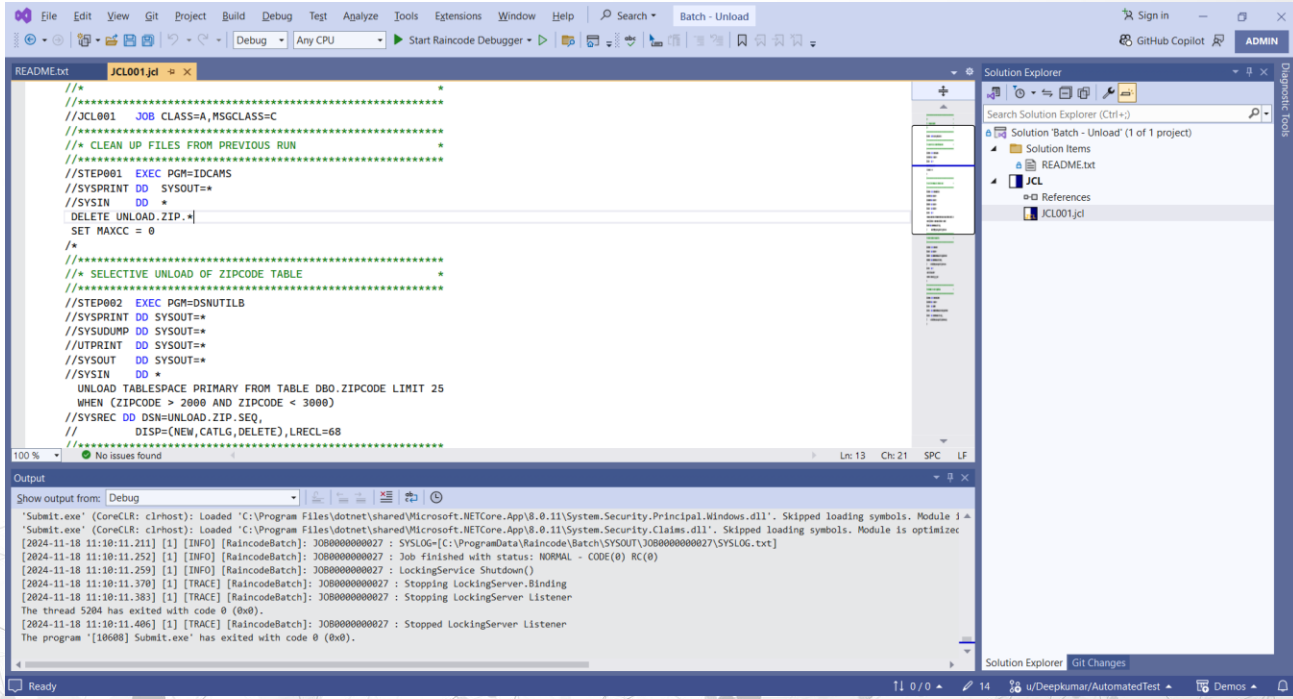


Unloading Data with DSNUTILB

- Illustrate the use of DSNUTILB for unloading data in a JCL context. This demo copies the contents of a database table into a sequential file with the DSNUTILB utility.

- Step 1: Open the solution from the folder Batch - Unload

- Step 2: Press Start Raincode Debugger to compile and run the project



RAINCODE 360 Advanced Showcase

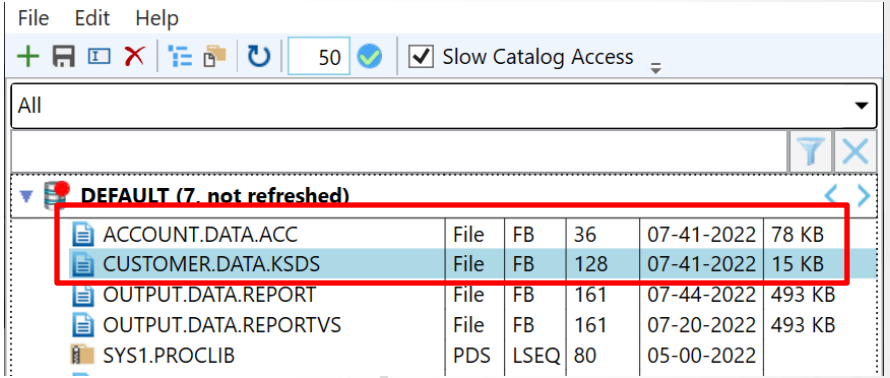
Raincode Batch – File types

Indexed Sequential File – Introduction

- Demo of Indexed Sequential File is available in the folder Batch - Indexed File
- This demo demonstrates two aspects of the indexed files:
  - JCL001.jcl: This demo creates two indexed sequential files created at **C:\Program Data\Raincode\Batch\DefaultVolume** and can also be viewed in the **Catalog Explorer**.
  - JCL002.jcl: reads two indexed files (one sequentially and one using the record key) to create a report (a sequential file).

Indexed Sequential File – How to Demonstrate – JCL001.jcl

- Step 1: Open the solution from the folder Batch – Indexed File
- Step 2: Click on Start Raincode Debugger to execute the project
- Step 3: To view the result, click on the Catalog Explorer shortcut placed on the desktop
- Step 4: To view the file content, right-click on the file name and select “Edit data file”

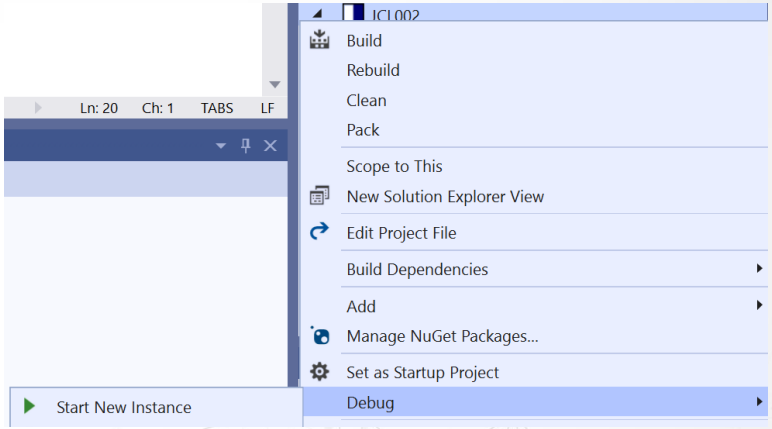


CUSTOMER.DATA.KSDS and ACCOUNT.DATA.ACC are the created indexed files

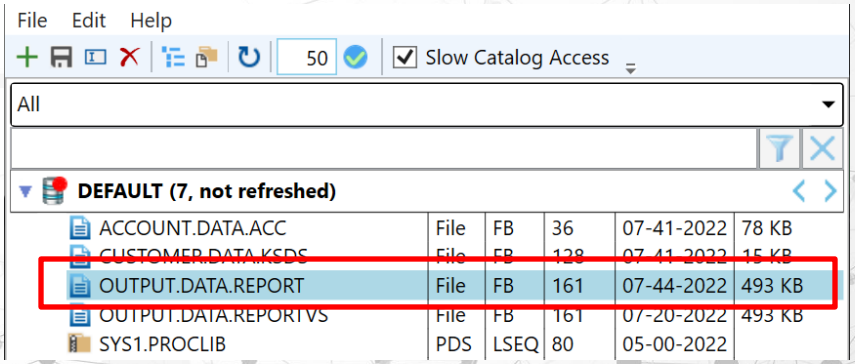
Indexed Sequential File – How to Demonstrate – JCL002.jcl

- Step 1: Open the solution from the folder Batch – Indexed File
  Note: JCL001.jcl should have been executed before

- Step 2: Right click on the "JCL002" project and select "Debug - Start a new instance".



- Step 3: To view the result, click on the Catalog Explorer shortcut placed on the desktop



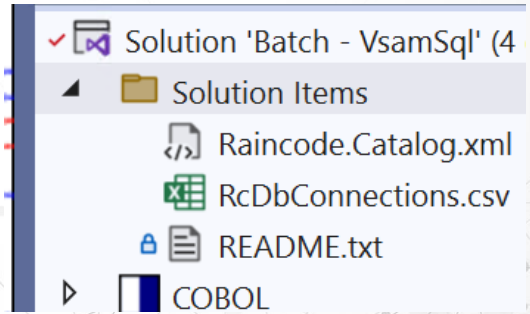
OUTPUT.DATA.REPORT is the created indexed file

VSAMSql Indexed Sequential File – Introduction

- Demo of VSAMSql Indexed Sequential File is available in the folder Batch – VsamSql

- VSAMSql is a file driver that stores data in an SQL database instead of saving them on a disk (refer to Catalog Configuration).

  - `Raincode.Config.xml` contains the configuration of the Catalog
    - `catalogFormat="db"` means that catalog is stored in the DB
    - `catalogDbConnection="VSAMSQL"` gives the connection to the DB. `VSAMSQL` is the name of the plan that can be find in the `RcDbConnections.csv` (refer to the Map Plan Name to Actual Connection String)
    - The mappings between DSN and table names are given in the <datasetSqlMapping> element (refer to Dataset mapping with tables)



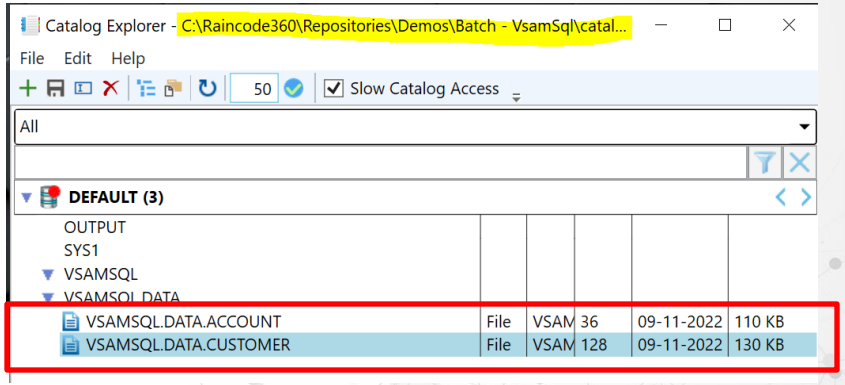
VSAMSql Indexed Sequential File – Introduction

- This demo demonstrates five aspects of indexed files:
  - JCL001.jcl: This demo creates two VSAMSql files by reading data from a database
  - JCL002.jcl: Reads two VSAMSql indexed files (one sequentially and one using the record key) to create a report (a sequential file)
  - JCL003.jcl: This demo migrates a VsamSql indexed file to a flat file on disk and back
  - ‘generate tables’: This demo generates the creation script for tables used to store the catalog and the data
  - ‘generate view’: This demo generates the views and triggers the creation script for the two files

VSAMSql Indexed Sequential File – JCL001.jcl

This project is the same as the "Batch-Indexed File" demo, except that the output files are VSAMSql files instead of VSAM files (on disk).

- Step 1: Open the solution from the folder Batch – VsamSql
- Step 2: Click on Start Raincode Debugger to execute the project to execute JCL001
- Step 3: To view the result, click on the Catalog Explorer shortcut placed on the desktop
  - Because this project doesn't use the default catalog, you should change the configuration (**File – Change configuration**) to `C:\Raincode360\Repositories\Demos\Batch - VsamSql\catalog\Raincode.Catalog.xml`



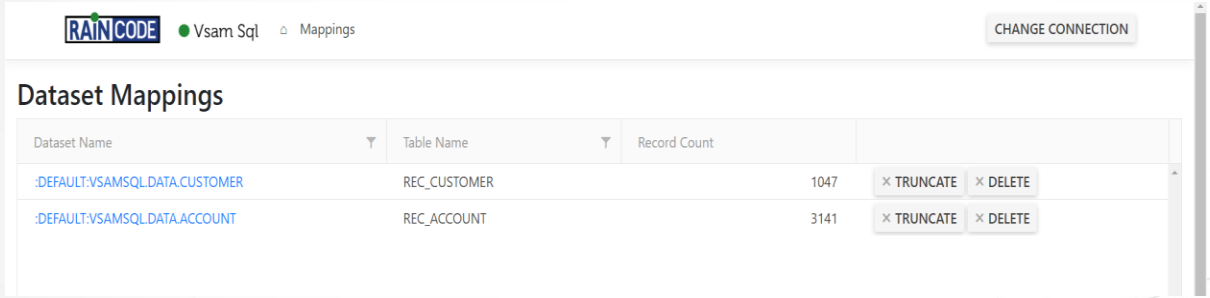
VSAMSQL.DATA.ACCOUNT and VSAMSQL.DATA.CUSTOMER are the created VSAM indexed files

The data is stored in a database.

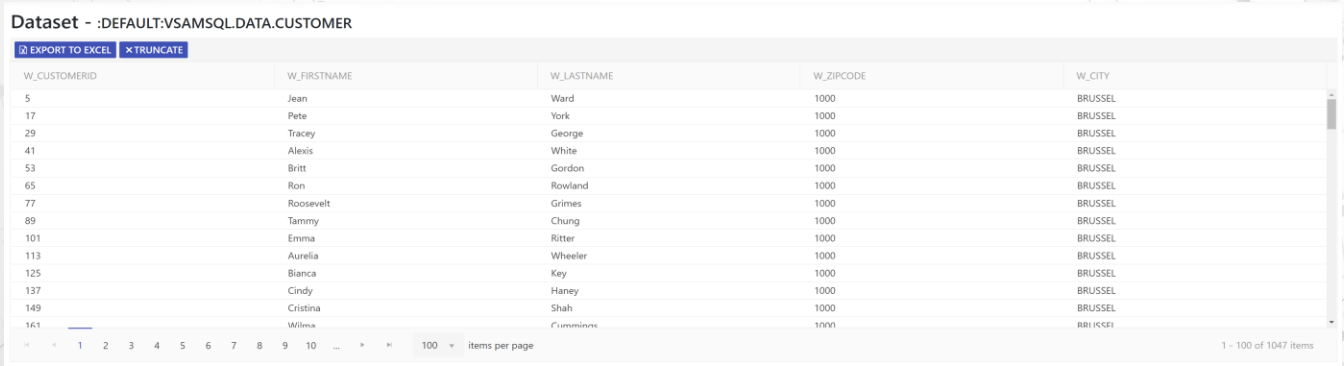
RAINCODE 360 Advanced Showcase

VSAMSql Indexed Sequential File – Raincode Vsam Sql Console

VSAMSQL.DATA.ACCOUNT and VSAMSQL.DATA.CUSTOMER are available under the mappings tab of the Raincode VSAMSql console. The Raincode VSAMSql console allows you to Truncate data or delete the files.



Click on one of the file name to display the data

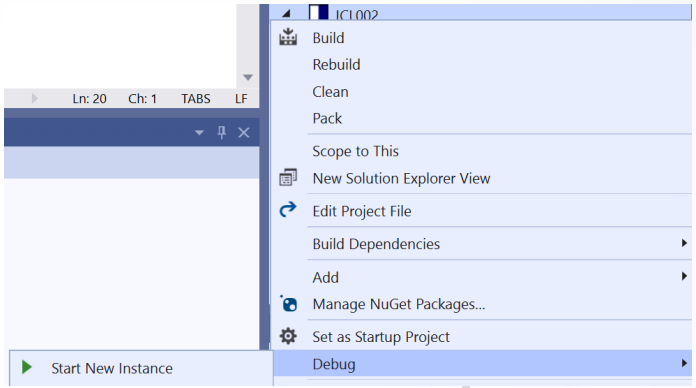


RAINCODE 360 Advanced Showcase

VSAMSql Indexed Sequential File – JCL002.jcl

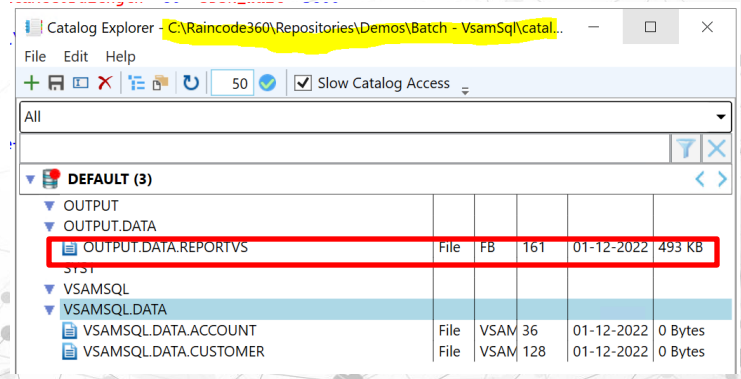
JCL001.jcl should have been executed before

- Step 1: Right-click on the "JCL002" project and select "Debug - Start a new instance".



- Step 2: To view the result, click on the Catalog Explorer shortcut placed on the desktop

OUTPUT.DATA.REPORTVS is the created file



RAINCODE 360 Advanced Showcase

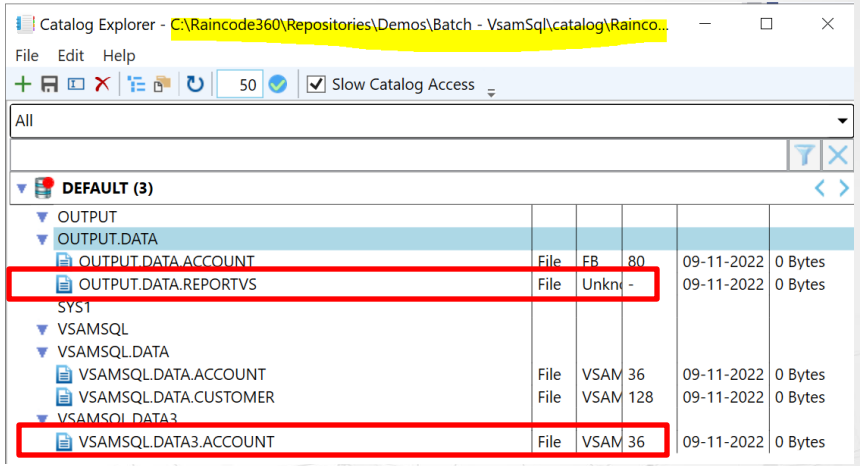
VSAMSql Indexed Sequential File – JCL003.jcl

JCL001.jcl should have been executed before

- This project migrates the VSAMSql indexed file
  `VSAMSQL.DATA.ACCOUNT`
  to the flat file on disk `OUTPUT.DATA.ACCOUNT`
  and back to `VSAMSQL.DATA3.ACCOUNT`

- Step 1: Right-click on the "JCL003" project and select "Debug - Start a new instance“

- Step 2: To view the result, click on the Catalog Explorer shortcut placed on the desktop (change configuration as in JCL001)



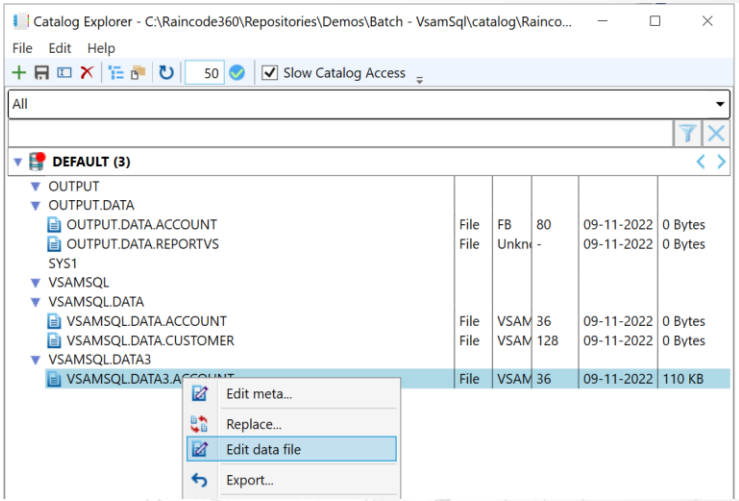
OUTPUT.DATA.ACCOUNT is a flat file

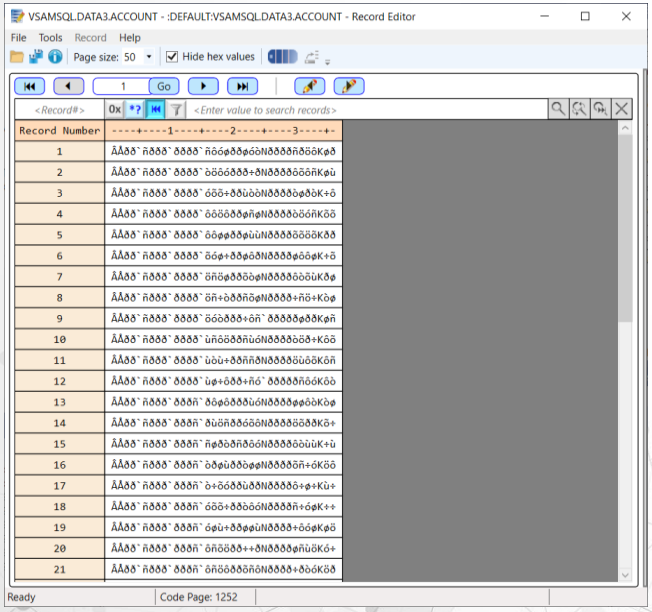
VSAMSQL.DATA3.ACCOUNT is a VSAM indexed files

RAINCODE 360 Advanced Showcase

VSAMSql Indexed Sequential File – JCL003.jcl – Contd..

- Step 4: To view the file content, right-click on the file name in the Catalog Explorer and select “Edit data file”.
  - The file will get open in the Record Editor.





RAINCODE 360 Advanced Showcase

VSAMSql Indexed Sequential File – generate tables

- Execute 'generate_tables.bat'

- This step reads the catalog definition (Catalog\Raincode.Catalog.xml) and generate the tables creation script (table_creation.sql)
  - CATALOG_META: is the table to store the catalog information
  - VSAM_META: is the table to store the DSN for which the data are in the database
  - REC_ACCOUNT and REC_CUSTOMER: the tables to store the data

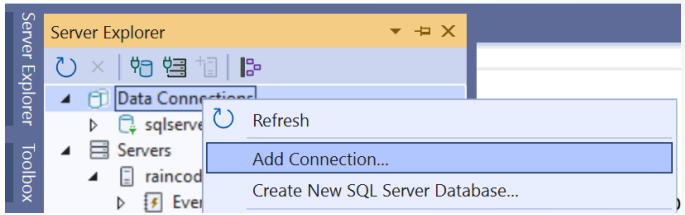
- The tables are already created. You didn't need to execute the Sql script

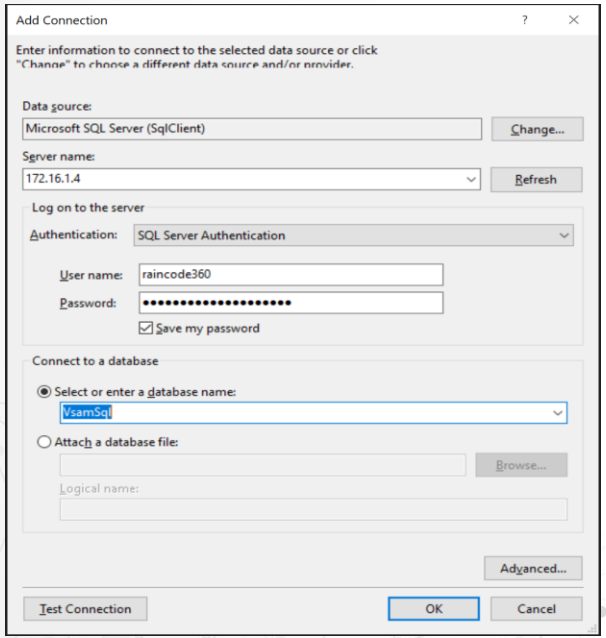
VSAMSql Indexed Sequential File – generate views

- Execute 'generate_views.bat'
- For each copybook (COBOL\ACCOUNT.cpy, COBOL\CUSTOMER.cpy) this script:
  - Use the COBOL compiler (cobrc) to convert the .cpy into an .xml file containing the declaration of the variables
  - The CopybookViewGenerator use the .xml file and connects to the DB to generate the views and the triggers creation scripts
- The views and triggers are already created. You didn't need to execute the SQL script.
- For each copybook
  - A view is created that contains one column for each copybook's variable with a picture clause and all the columns of the table
  - A "trigger instead of update" to update the data using the view
  - A "trigger instead of insert" to insert data using the view
  - A "trigger instead of delete" to delete data using the view

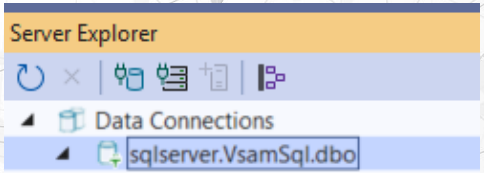
VSAMSql Indexed Sequential File – How to connect the database

- Step 1: Right-click on the *Data Connections* in the Server Explorer of the Visual Studio and select *Add Connection...*





- Step 2: Select Microsoft SQL Server and Add Connection dialog box will appear. Fill the dialog box as follows:
  - Server Name – the value of the environment variable *DB_SERVER.*
  - Authentication- *SQL Server Authentication*
    - *Username: raincode360*
    - *Password: the value of the environment variable DB_PASSWORD*
  - Connect to a database – Choose VsamSql from the list.



- Step 3: Click OK, and the database will appear under *Data connections* in the Server Explorer

RAINCODE 360 Advanced Showcase

VSAMSql Indexed Sequential File – How to query the database

- Right-click on the newly created data connection and select *New Query*

- The database contains two kinds of information
  - The catalog
  - The data themselves

- The catalog meta-information are stored in the CATALOG_META table.
  Execute the following query to see the contains of the catalog

```
select * from CATALOG_META
```

	ID	TYPE	PARENT_NAME	LEVEL	CATALOG_KEY	FULL_NAME	CONFIG
1	1015	P	:DEFAULT:SYS1	1	:DEFAULT:SYS1.PROCLIB	:DEFAULT:SYS1.PROCLIB	<dataSet version="4" name="SYS1.PROCLIB" vo...
2	1025	D	:#SYSOUT:JOB0000000011-JCL001.STEP002_3	2	:#SYSOUT:JOB0000000011-JCL001.STEP002_3.SYSOUT	:#SYSOUT:JOB0000000011-JCL001.STEP002_3.SYSOUT	<dataSet version="4" name="JOB0000000011-JCL...
3	1029	D	:DEFAULT:OUTPUT.DATA	2	:DEFAULT:OUTPUT.DATA.ACCOUNT	:DEFAULT:OUTPUT.DATA.ACCOUNT	<dataSet version="4" name="OUTPUT.DATA.ACCO...
4	1031	D	:DEFAULT:VSAMSQL.DATA	2	VSAMSQL.DATA.CUSTOMER	:DEFAULT:VSAMSQL.DATA.CUSTOMER	<dataSet version="4" name="VSAMSQL.DATA.CUS...
5	1032	D	:DEFAULT:VSAMSQL.DATA	2	VSAMSQL.DATA.ACCOUNT	:DEFAULT:VSAMSQL.DATA.ACCOUNT	<dataSet version="4" name="VSAMSQL.DATA.ACC...

the `CONFIG` column contains the meta information about the DSN

VSAMSql Indexed Sequential File – How to connect the database

- The table VSAM_META contains the list of files whom data are stored in the DB.
  Execute the following query to see the data
  `select * from VSAM_META`
  It contains two lines (one for each file) and each file is associated with one table

	ID	FILE_NAME	TABLE_NAME
1	2	SYSDA:DEFAULT:VSAMSQL.DATA.CUSTOMER	REC_CUSTOMER
2	3	SYSDA:DEFAULT:VSAMSQL.DATA.ACCOUNT	REC_ACCOUNT

- The data in the tables are difficult to read because they are in the binary format (VARBINARY)
  `select top(10) * from REC_ACCOUNT`

	RID	FILE	data	KEY
1	1	3	0xC2C5F0F060F1F0F0F060F0F0F0F060F1F4F3F8F0F0F8F3...	0xC2C5F0F060F1F0F0F060F0F0F0F060F1F4F3F8
2	2	3	0xC2C5F0F060F1F0F0F060F0F0F0F060F2F6F4F3F0F0F0F7...	0xC2C5F0F060F1F0F0F060F0F0F0F060F2F6F4F3

- For each table, there is a view that shows the data in a more readable format
  `select top(10) * from REC_ACCOUNT_V`

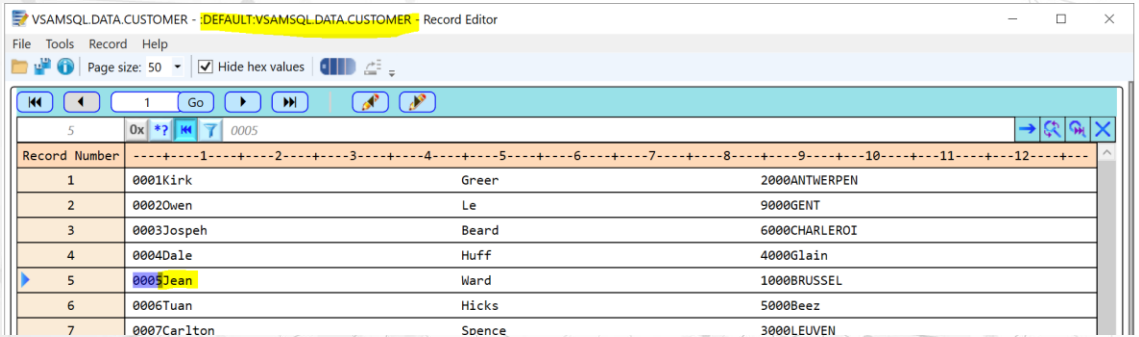
	RID	FILE	data	KEY	W_ACCOUNT_ID	W_CUSTOMERID_ID	W_AMOUNT
1	1	3	0xC2C5F0F060F1F0F0F060F0F0F0F060F1F4F3F8F0F0F8F3...	0xC2C5F0F060F1F0F0F060F0F0F0F060F1F4F3F8	BE00-1000-0000-1438	832	+00001064.80
2	2	3	0xC2C5F0F060F1F0F0F060F0F0F0F060F2F6F4F3F0F0F0F7...	0xC2C5F0F060F1F0F0F060F0F0F0F060F2F6F4F3	BE00-1000-0000-2643	70	+00004541.89

VSAMSql Indexed Sequential File – How to connect the database

- Triggers have been defined to modify (update, insert, delete) the data using the view
- To update the 'W_AMOUT' of account id 'BE00-1000-0000-1438'
  ```
  update REC_CUSTOMER_V set W_FIRSTNAME = 'Jean' where W_CUSTOMERID = 5;
  ```
- Check the data have been update in the DB
  ```
  select * from REC_CUSTOMER_V where W_CUSTOMERID = 5;
  ```

	RID	FILE	data	KEY	W_CUSTOMERID	W_FIRSTNAME	W_LASTNAME	W_ZIPCODE	W_CITY
1	15237	3008	0xF0F0F0F5D1858195404040404040404040404040404040...	0xF0F0F0F5	5	Jean	Ward	1000	BRUSSEL

- The record can also be used to verify the contains of the dataset (VSAMSQL.DATA.ACCOUNT)



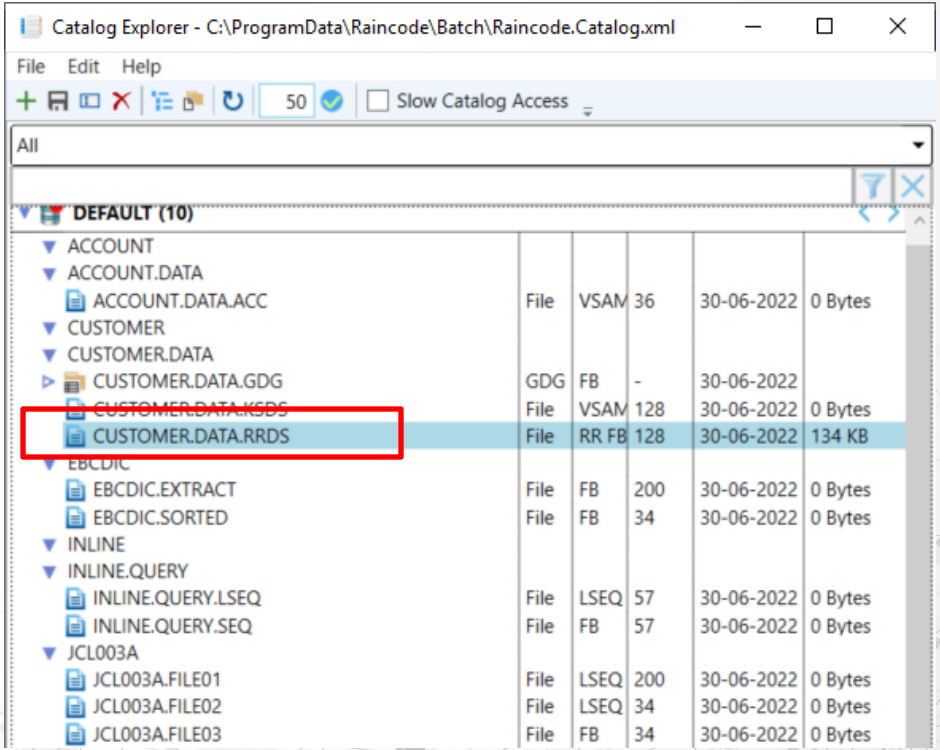
RAINCODE BEO Advanced Showcase

Relative File – Introduction

- Demo of Relative File is available in the folder Batch - Relative File
- This demo creates a Relative File created at **C:\Program Data\Raincode\Batch\DefaultVolume** and can also be viewed in the **Catalog Explorer**.

Relative File – How to Demonstrate

- Step 1: Open the solution from the folder Batch – Relative File
- Step 2: Click on Start Raincode Debugger to execute the project
- Step 3: To view the result, click on the Catalog Explorer shortcut placed on the desktop



CUSTOMER.DATA.RRDS is the created relative file

RAINCODE 360 Advanced Showcase

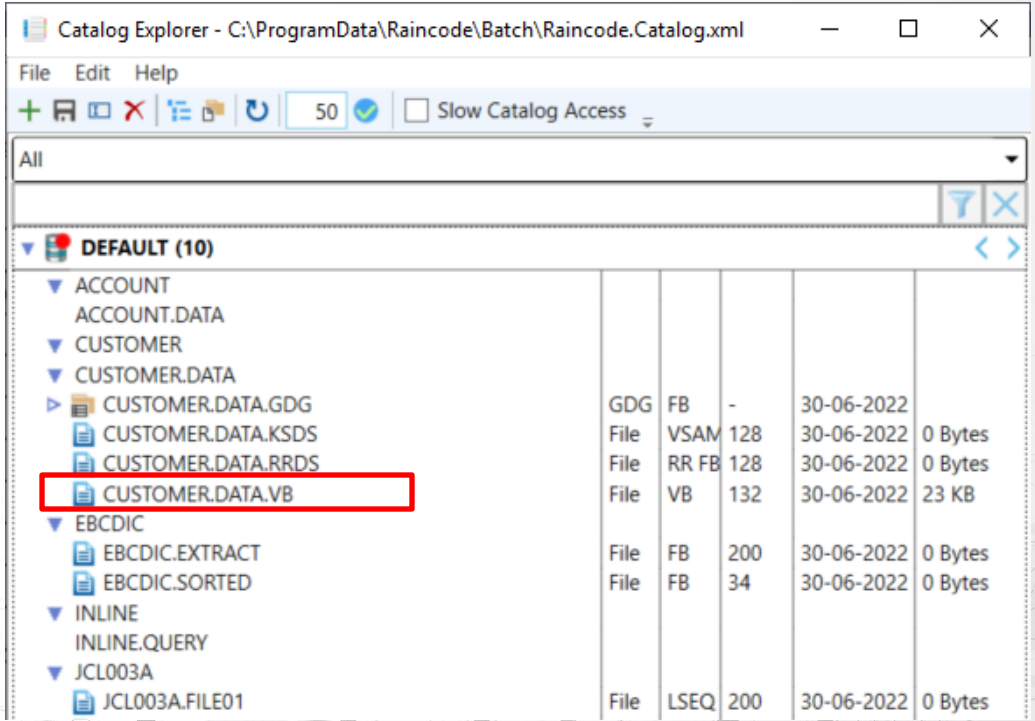
Variable Length File – Introduction

- Demo of Variable Length File is available in the folder Batch – Variable Length File
- This demo creates a Variable Length File created at **C:\Program Data\Raincode\Batch\DefaultVolume** and can also be viewed in the **Catalog Explorer**.

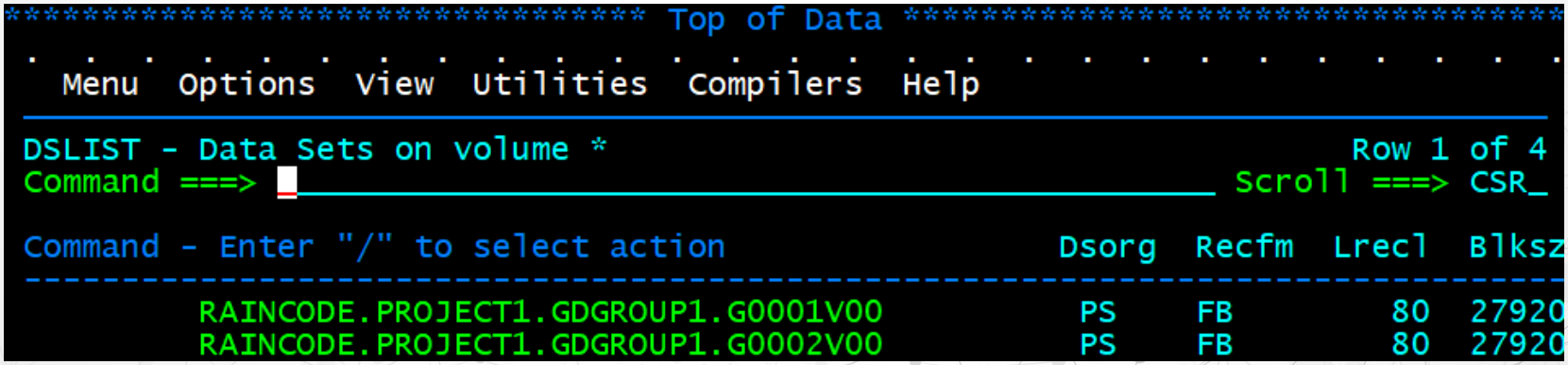
Variable Length File – How to Demonstrate

- Step 1: Open the solution from the folder Batch – Variable Length File
- Step 2: Click on Start Raincode Debugger to execute the project
- Step 3: To view the result, click, on the Catalog Explorer shortcut placed on the desktop

CUSTOMER.DATA.VB is the created variable length file



Generation Data Group



Generation Data Group – Introduction

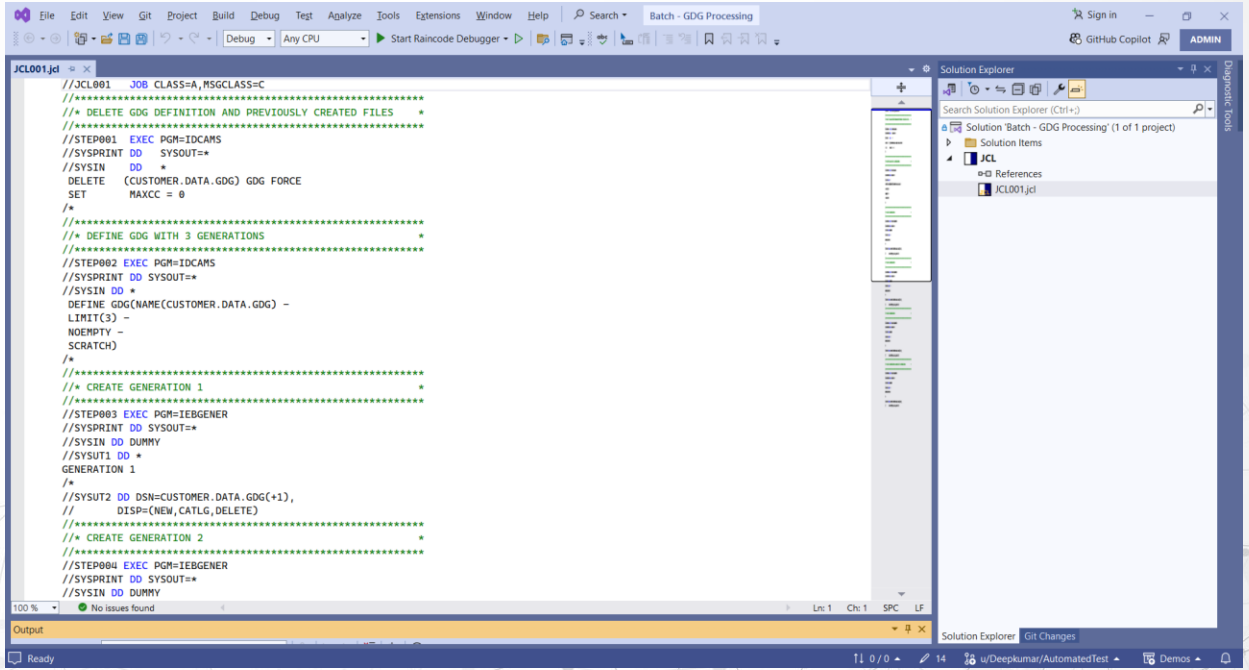
- A Generation Data Group, also known as GDG, is versioning of data sets that are successive generations of historically related data
- This Batch demo creates multiple generations of a GDG
- Created GDG and its version will be available at C:\Program Data\Raincode\DefaultVolume.

Generation Data Group – How to demonstrate

- Step 1: Open the solution from the folder Batch - GDG Processing

- Step 2: Click on Start Raincode Debugger to execute the solution

- The JCL starts by deleting the output from the previous run (if any)



- Created GDG and its version can also be seen using [Catalog Explorer]

RAINCODE BEO Advanced Showcase

Partitioned Dataset

Partitioned data set - Introduction

- A Partitioned Data Set, also known as **PDS**, is a Data Set that contains directory and members.
- Illustrated features:
  - Creates a PDS (pdslib) via JCL
  - Usage of IEBFR14 to create a file
  - Usage of IEBFR14 with the PATH parameter

Partitioned Dataset– How to demonstrate

- Step 1: Open the solution from the folder Batch – Partitioned Dataset.
- Step 2: Click on Start Raincode Debugger to execute P001 project.
  - It will run JCL001, which creates the pdslib PROCLIB.TEST.JCL with a single member named MYPROC
  - MYPROC is a JCL procedure that can be invoked from other JCLs with parameters.
- Step 3: Right-click on P002 and select Debug->Start New instance to execute P002 project.
  - It will run JCL002, which has the statement “LIB001 JCLLIB ORDER=PROCLIB.TEST.JCL”, that indicates where the procedure can be found.

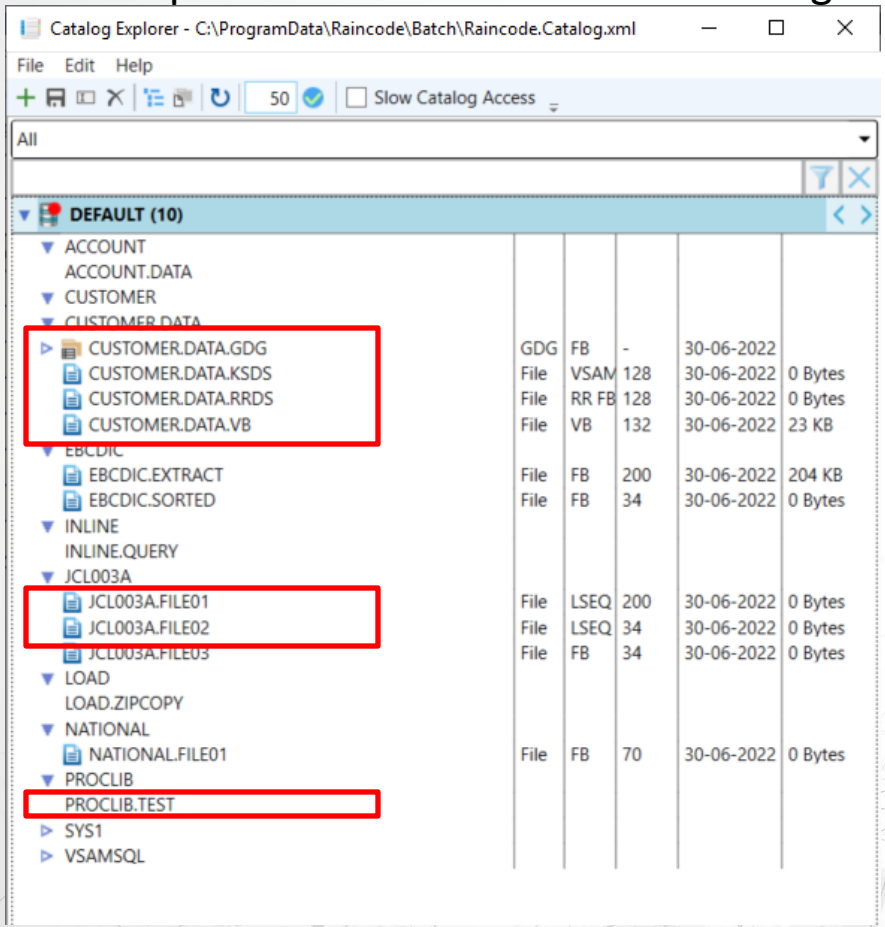
Default location for JCL procedures is SYS1.PROCLIB

Created PDS can also be seen using Catalog Explorer

Catalog Explorer

Catalog Explorer

- A Graphical User Interface for accessing the (virtual) mainframe file system.



The Raincode Catalog Explorer allows you to manage mainframe files, PDS (Partitioned Data Set) and GDG (Generation Data Group), and more.

When Catalog Explorer starts, it populates all the data in the DSN view available at C:\ProgramData\Raincode\Batch\DefaultVolume

CUSTOMER.DATA.GDG and its generation were created when GDGProcessing.sln was compiled and executed.

JCL003A.FILE01 and JCL003A.FILE02 are output files that were created when Batch - Legacy with Database Access.sln was compiled and executed.

PROCLIB.TEST is created when Batch – Partitioned Dataset.sln was compiled and executed.

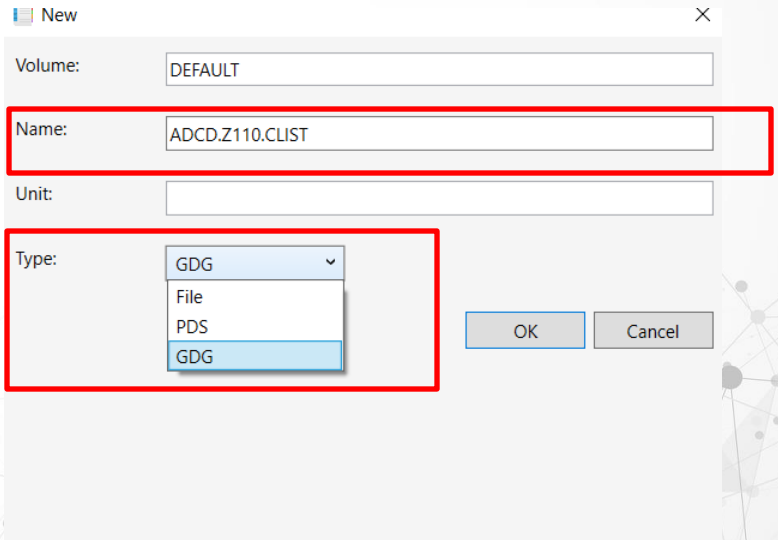
RAINCODE BEO Advanced Showcase

Catalog Explorer – Features

- Files
  - Adding, replacing, exporting, deleting, renaming
  - Editing (Record Editor)
- Partitioned data sets (PDS)
  - Adding a new PDS, deleting a PDS
  - Adding members to a PDS
  - Deleting members from a PDS
- Generation data group (GDG)
  - Adding a new GDG, Exporting, Deleting
  - Adding generation to an existing GDG
  - GDG (+1), GDG (-1)
- Import/Export

Catalog Explorer – How to create datasets

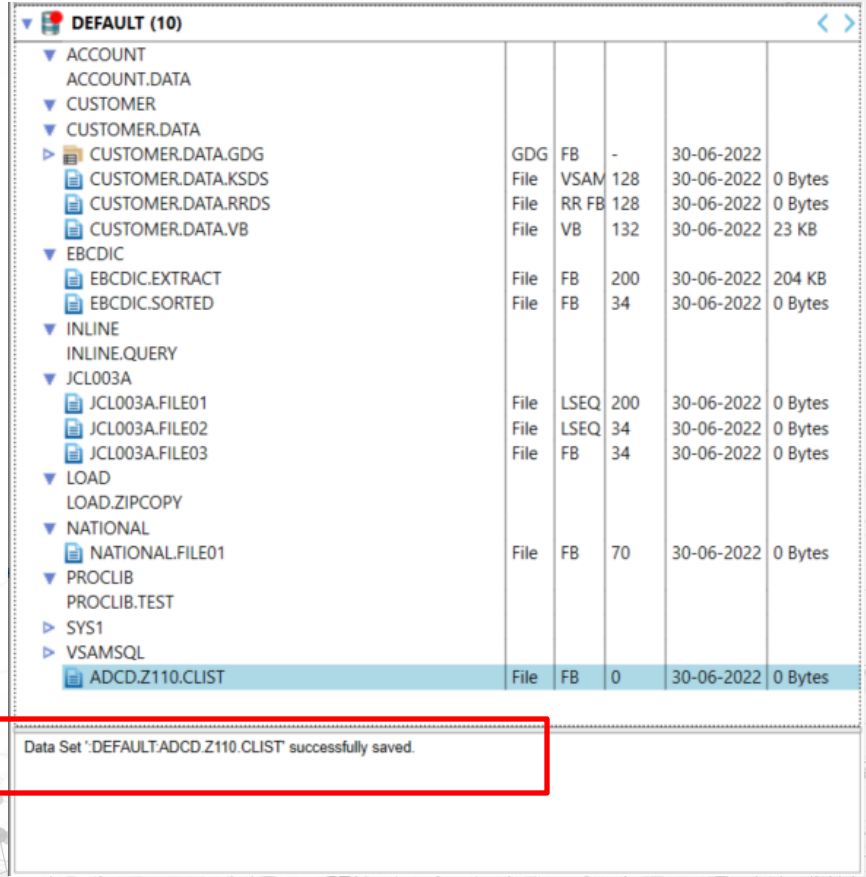
- Step 1: Launch the Catalog Explorer through e.g. the desktop shortcut
- Step 2: From the toolbar, select File → New or press + , and add the datasets (File, PDS, or GDG).
  - For this demonstration, we will use GDG as an example
  - Add the Name of Dataset
  - Select Type of Dataset from the drop-down menu
  - Press Ok



Catalog Explorer – How to save created datasets

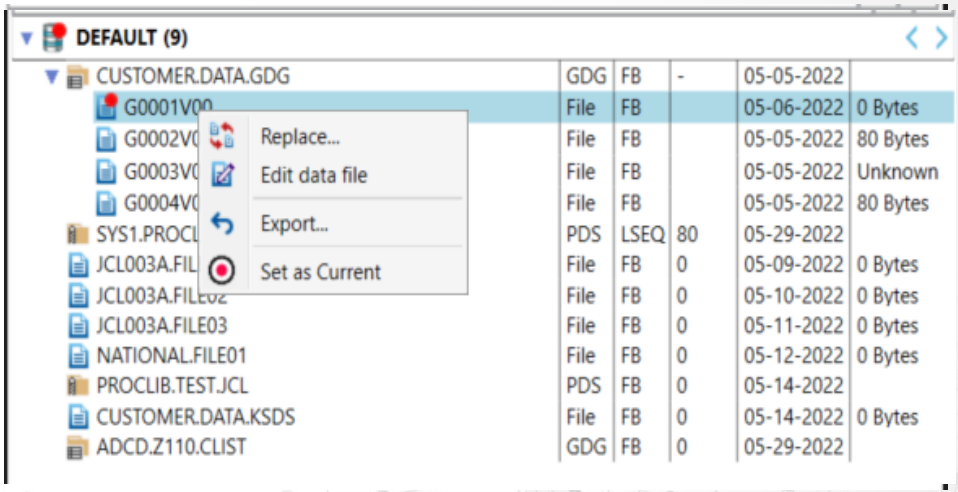
- Step 3: Press Save. The dataset will get created in the DefaultVolume folder and will be visible in the Catalog Explorer.

If you select Edit→ Activity Log → Info, then Activity Log shows all the actions taken by the user, such as creating datasets.



Catalog Explorer – How to add generation in the existing GDG

- Right click on the existing GDG, context menu will appear, select Add Generation
  - It will add generation
  - The latest added generation is always set as current by default
  - It can be changed by right click on the selected generation and selecting 'set as current' from the context menu
  - Previously G0004V00 was set as current, and now it's G0001V00 in CUSTOMER.DATA.GDG dataset



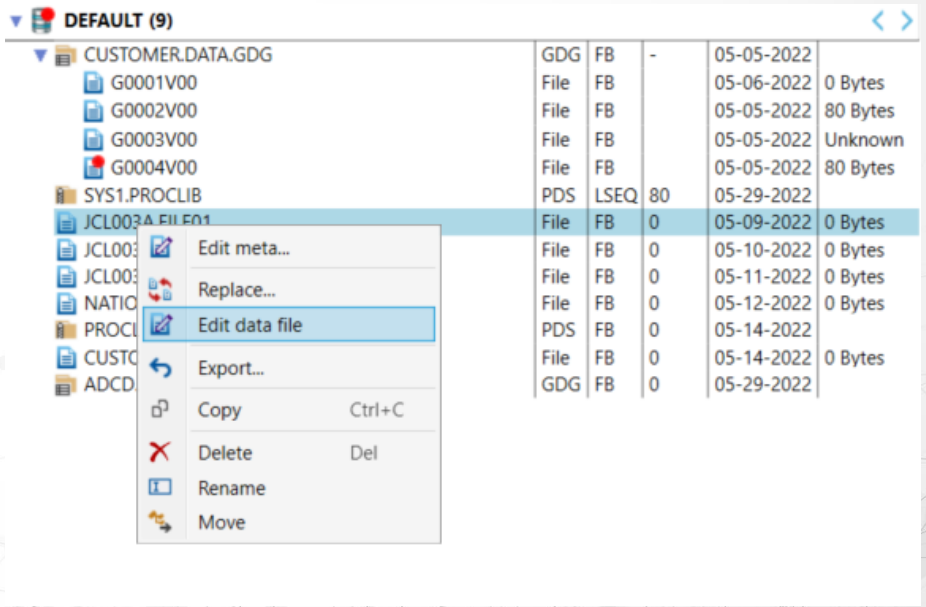
Record Editor

Record Editor – Introduction

- Record Editor is a GUI-based tool that allows users to view and edit file contents and view and modify certain metadata.
- Record Editor deals with record-based files. This tool aims to let the user work with data files as a sequence of records to provide Mainframe-like functionality.
- Record Editor provides the following features:
  - Display data file contents as records
  - Edit records in binary format (hexadecimal)
  - Display records in terms of fields
  - Edit records field-wise according to declaration description
  - Search records as per encoding
  - Search within fields
  - Display certain metadata
  - Modify record length and the encoding

Record Editor – How to launch

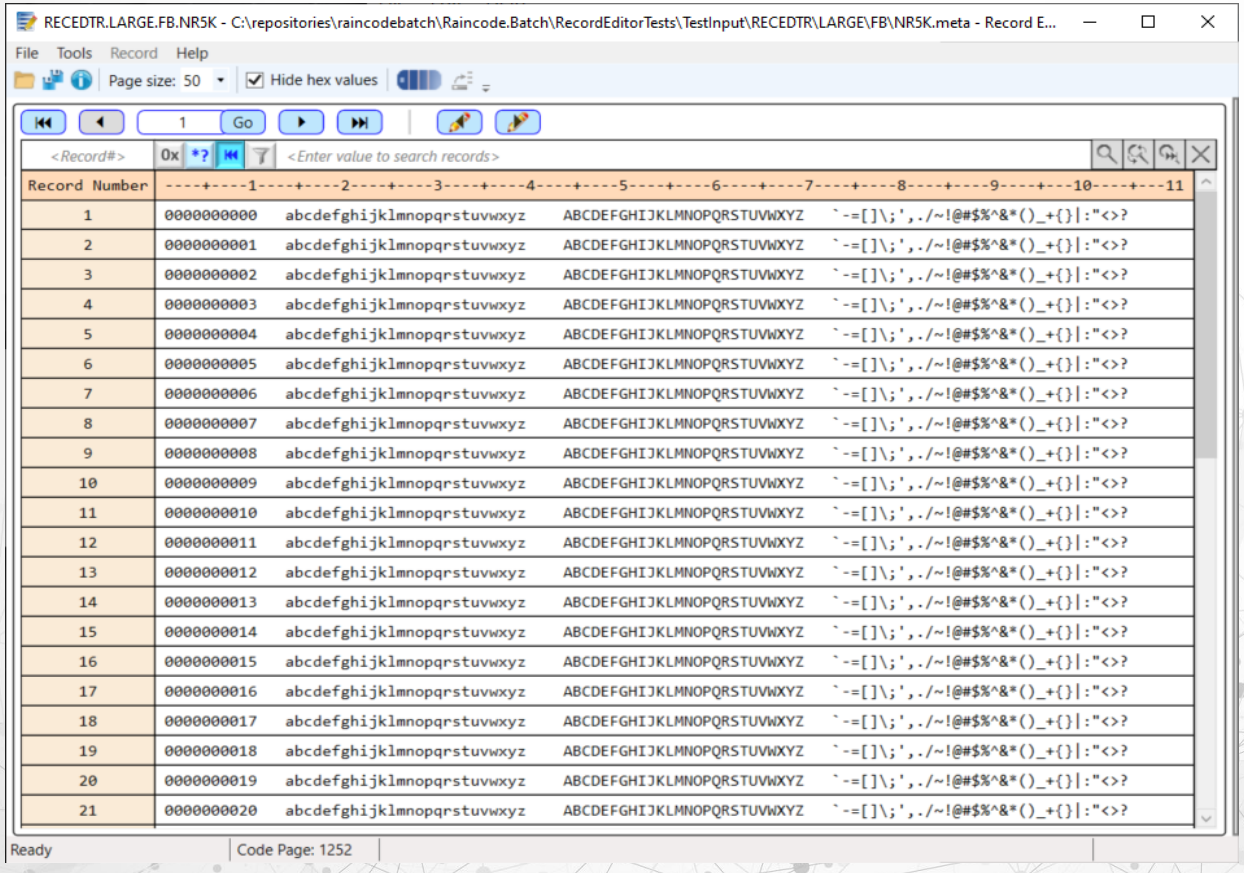
- Raincode Editor can be launched in two ways:
  - By double-clicking on the “Record Editor” shortcut placed at the desktop
  - From the Catalog Explorer:
    - Select the file and right-click, the context menu will appear
    - Select Edit data file to launch Record Editor



RAINCODE BEO Advanced Showcase

Record Editor – How to load data file

- Step 1: Select Tool->Open from the toolbar, a dialog box will appear.
- Step 2: Select the appropriate metadata file. Record Editor will load the corresponding data file.



Refer to the Record Editor documentation for more details.

RAINCODE BEO Advanced Showcase

Automatic Restart

Automatic Restart – Concepts

- Mass update programs must perform periodic commits for various reasons.
  - To make sure that the transaction log of the database is not flooded.
  - To make sure that locks are periodically released.
  - To limit the amount of rollback in case of an error.
- Intermediate commits make it more difficult to restart a job after an abend.
- Automatic Restart optionally performs certain actions behind the scenes.
  - Taking periodic commits.
  - Saving the state of the program with each commit.
  - Repositioning cursors and file pointers at restart.

Automatic Restart – Configuration

- Automatic Restart is configured by means of XML stored in a configuration file.

```
<?xml version="1.0" encoding="utf-8"?>
<Configuration>
    <ProcessingOptions>
        <ProcessingOption AutomaticRestart="true"
                          AutomaticCheckpoints="true"
                          AutomaticCursorPositioning="true"
                          PaceCheckpoints="false"
                          TriggerCount="5"
                          RestoreWorkingStorage="PGMSTART"
                          ProgName=".*"
                          JobName=".*"
                          StepName=".*">
            <Cursor Package="COB001" Name="CUR01" />
        </ProcessingOption>
    </ProcessingOptions>
    <CursorRepositioningOptions>
        <CursorRepositioningOption CursorPackage="COB001" CursorName="CUR01" RepositioningMethod="COLUMN">
            <Columns>
                <Column Name="AccountNumber" />
            </Columns>
        </CursorRepositioningOption>
    </CursorRepositioningOptions>
</Configuration>
```

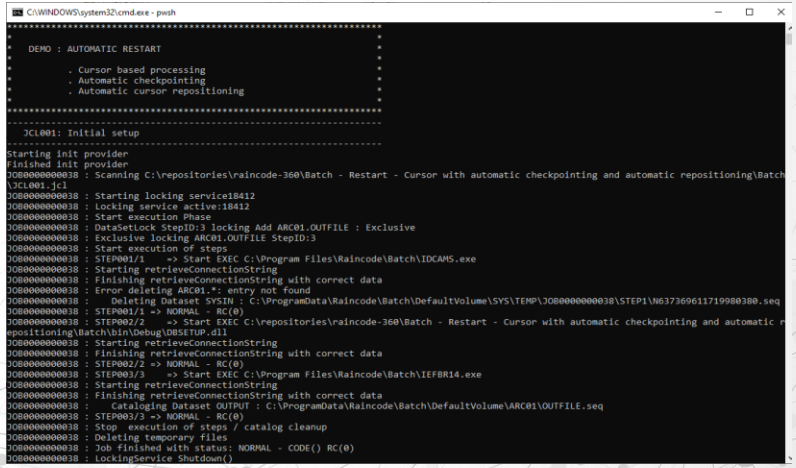
- Meaning of the parameters in this particular example:
  - The program COB001 is controlled by Automatic Restart.
  - Commits are taken every five updates
  - Working storage is restored at the restart
  - The main cursor CUR01 is repositioned at the restart

Automatic Restart – Demos

- There are three Demos, each accomplishing the same task with a different configuration.
  - Demo 1: Cursor based processing and automatic repositioning
  - Demo 2: Cursor based processing and Manual Cursor repositioning
  - Demo 3: File based processing
- In each demo, the main program is started and purposefully abended by forcing an error. After that, the main program is restarted, at which point it will run until successful completion.
- Each demo produces its own set of output files or ‘snapshots’.
  - Snapshot 1: before the execution of the main program
  - Snapshot 2: after the main program has abended
  - Snapshot 3: after the restart and successful end of the main program

Automatic Restart – Script

- Open the appropriate demo in Visual Studio.
- Start the main script by typing ‘run’ at the Developer Command Prompt.
- The script starts by running the JCL002, which executes the program COB001.
- COB001 updates a total of 20 accounts in the database.
- Automatic Restart is configured to perform an automatic checkpoint after every 5 updates.
- An abend is forced just before the 4th commit (TRMBEFORCKP=3).
- JCL002 is submitted with the necessary restart parameters (-Restart and –RestartJobId).
- The restarted job runs until completion.

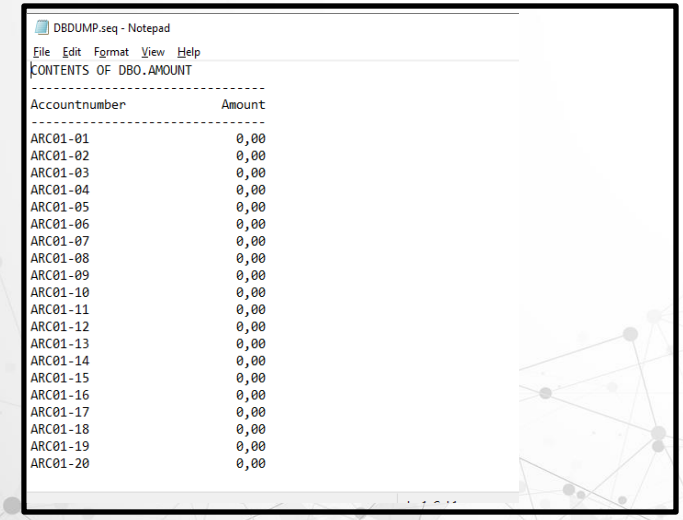


RAINCODE 360 Advanced Showcase

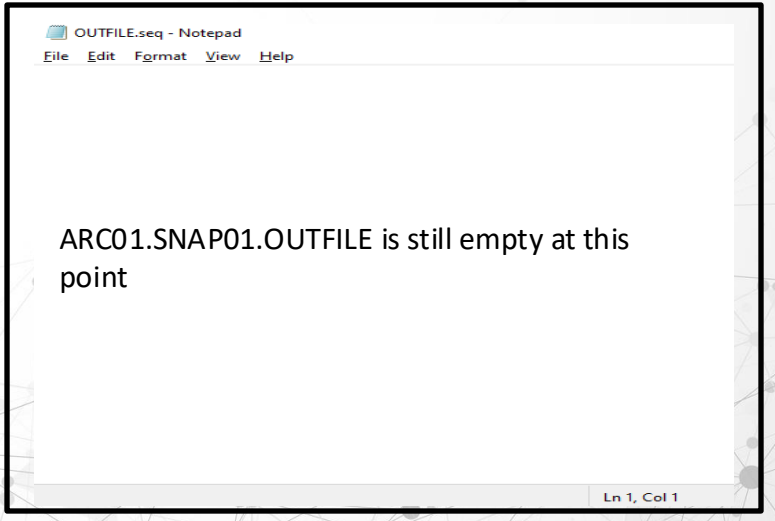
Automatic Restart – Output (1)

- Snapshot 1 is taken before COB001 is started.
  - DBDUMP reflects the contents of the database at the time of the snapshot.
  - OUTFILE is a sequential file produced by COB001 and is still empty at this point.

ARC0(x).SNAP01.DBDUMP



ARC0(x).SNAP01.OUTFILE



RAINCODE 360 Advanced Showcase

Automatic Restart – Output (2)

- SNAP02: after abend, another snapshot is taken.

ARC0(x).SNAP02.DBDUMP lists the state of the database at this point.

DBDUMP.seq - Notepad

File Edit Format View Help

CONTENTS OF DBO.AMOUNT

Accountnumber	Amount
ARC01-01	0,10
ARC01-02	0,10
ARC01-03	0,10
ARC01-04	0,10
ARC01-05	0,10
ARC01-06	0,10
ARC01-07	0,10
ARC01-08	0,10
ARC01-09	0,10
ARC01-10	0,10
ARC01-11	0,10
ARC01-12	0,10
ARC01-13	0,10
ARC01-14	0,10
ARC01-15	0,10 ← Rollback
ARC01-16	0,00
ARC01-17	0,00
ARC01-18	0,00
ARC01-19	0,00
ARC01-20	0,00 ← abend

Ln 25, Col 1

ARC0(x).SNAP02.OUTFILE is truncated after the 15th account

OUTFILE.seq - Notepad

File Edit Format View Help

COB001 UPDATES

Accountnumber	Amnt Before	Amnt After
ARC01-01	0,00	0,10
ARC01-02	0,00	0,10
ARC01-03	0,00	0,10
ARC01-04	0,00	0,10
ARC01-05	0,00	0,10
ARC01-06	0,00	0,10
ARC01-07	0,00	0,10
ARC01-08	0,00	0,10
ARC01-09	0,00	0,10
ARC01-10	0,00	0,10
ARC01-11	0,00	0,10
ARC01-12	0,00	0,10
ARC01-13	0,00	0,10
ARC01-14	0,00	0,10
ARC01-15	0,00	0,10

The lines containing accounts 16 through 20 were actually written by COB001 but are removed by the TRMBEFORCKP after the abend

Ln 12, Col 80

Automatic Restart – Output (3)

- SNAP03: Creates a new snapshot (after the successful ending).

ARC0(x).SNAP03.DBDUMP

DBDUMP.seq - Notepad

File Edit Format View Help

CONTENTS OF DBO.AMOUNT

Accountnumber	Amount
ARC01-01	0,10
ARC01-02	0,10
ARC01-03	0,10
ARC01-04	0,10
ARC01-05	0,10
ARC01-06	0,10
ARC01-07	0,10
ARC01-08	0,10
ARC01-09	0,10
ARC01-10	0,10
ARC01-11	0,10
ARC01-12	0,10
ARC01-13	0,10
ARC01-14	0,10
ARC01-15	0,10
ARC01-16	0,10
ARC01-17	0,10
ARC01-18	0,10
ARC01-19	0,10
ARC01-20	0,10 ← Final Commit

ARC0(x).SNAP03.OUTFILE

OUTFILE.seq - Notepad

File Edit Format View Help

COB001 UPDATES

Accountnumber	Amnt Before	Amnt After
ARC01-01	0,00	0,10
ARC01-02	0,00	0,10
ARC01-03	0,00	0,10
ARC01-04	0,00	0,10
ARC01-05	0,00	0,10
ARC01-06	0,00	0,10
ARC01-07	0,00	0,10
ARC01-08	0,00	0,10
ARC01-09	0,00	0,10
ARC01-10	0,00	0,10
ARC01-11	0,00	0,10
ARC01-12	0,00	0,10
ARC01-13	0,00	0,10
ARC01-14	0,00	0,10
ARC01-15	0,00	0,10
ARC01-16	0,00	0,10
ARC01-17	0,00	0,10
ARC01-18	0,00	0,10
ARC01-19	0,00	0,10
ARC01-20	0,00	0,10

Ln

Files with National Characters

Files with National Characters

- National character strings are declared as PIC N(n) USAGE NATIONAL.
- Each character is represented by two or four bytes.
- Numeric edited variables can also be declared with the USAGE NATIONAL.
- Supported operations are:
  - computations,
  - output to the console,
  - viewing values in the debugger,
  - reading and writing to a database.

Files with National Characters

- To compile programs with the embedded national characters, **MOVE N'世界！' TO REC** , the compilation command line must contain the following option: **StringSourceFileEncoding = utf-8**

- National characters are encoded in UTF-16 big endian format.

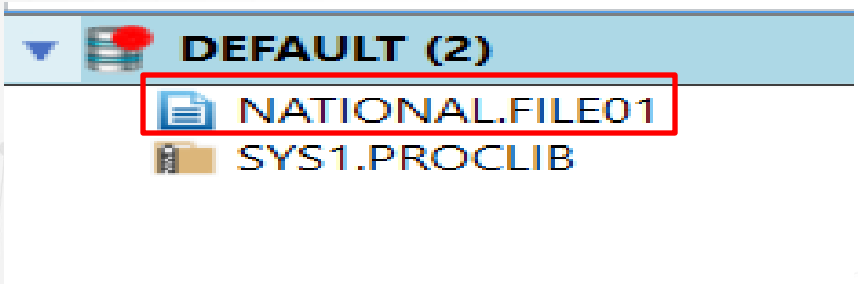
Note: If the console cannot display Unicode characters, change the language to Japanese in [configuration/time and language/language/administrative language settings/change system locale] to make the Japanese characters visible. This requires rebooting of your machine.

Files with National Characters – Demo

- This demo shows how national characters can be written to files.
- Demonstrates how the Record Editor can be used to view these files.
- This demo reads English and Japanese descriptions from a table file.

Files with National Characters – How to demonstrate

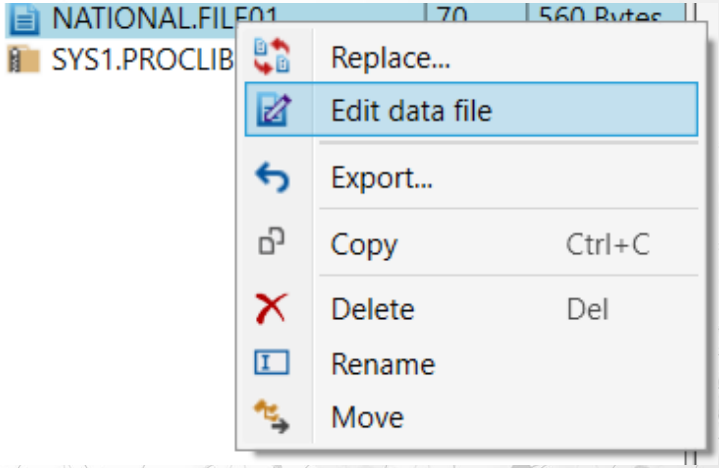
- Step 1: Open the solution from the folder Batch – Files with National Character.
- Step 2: Click on Start Raincode Debugger to execute the project.
- Step 3: To view the output file contents, double click the Catalog Explorer shortcut placed on the desktop.



NATIONAL.FILE01 is the output file. It contains both character fields and national fields.

Files with National Characters – How to demonstrate

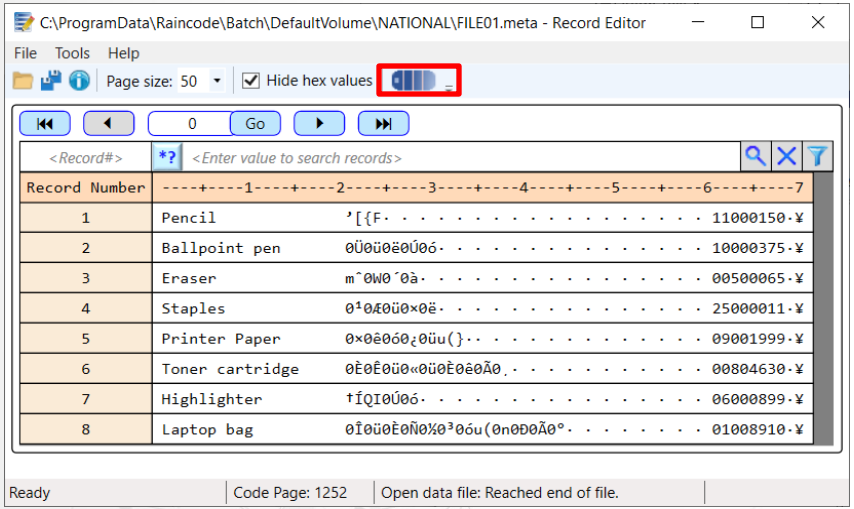
- Step 4: Right-click on the output file in the Catalog Explorer and select Edit data file to open it in the Record Editor.
  - You can also directly open the File in the Record Editor by double-clicking the Record Editor shortcut placed on the desktop and navigating to the desired location. By default, all the output files are created at C:\ProgramData\Raincode\Batch\DefaultVolume.



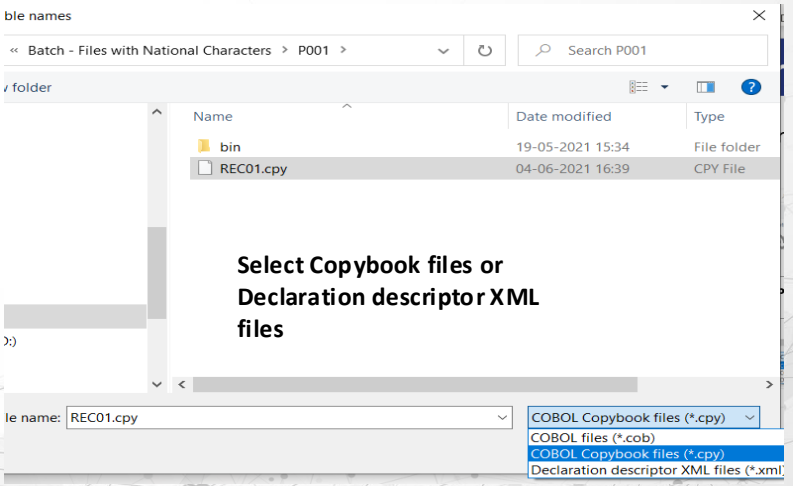
Files with National Characters – How to demonstrate

- Step 5: Switch to field view using the declaration descriptor or the copybook to make Japanese characters visible.

1.



2.

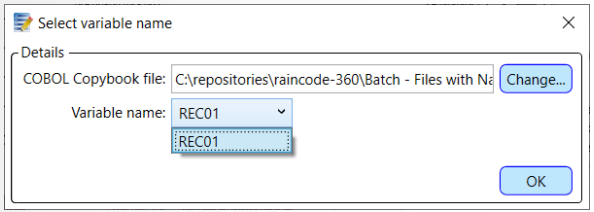


Run **gendd.bat** in the solution directory if you want to update *DeclarationDescriptors.xml*

RAINCODE BEO Advanced Showcase

Files with National Characters – How to demonstrate

3.



4.

C:\ProgramData\Raincode\Batch\DefaultVolume\NATIONAL\FILE01.meta - Record Editor

File Tools Help

Page size: 50 Hide hex values Variable name: REC01

0 Go

<Record#> *? <Enter value to search records>

Record Number	REC01-NM-ENG	REC01-NM-JPN	REC01-QTY	REC01-PRICE	REC01-CURRENCY
1	Pencil	鉛筆	110	001.50	¥
2	Ballpoint pe	ボールペン	100	003.75	¥
3	Eraser	消しゴム	005	000.65	¥
4	Staples	ステープル	250	000.11	¥
5	Printer Pape	プリンター用紙	090	019.99	¥
6	Toner cartri	トナーカートリッジ	008	046.30	¥
7	Highlighter	蛍光ペン	060	008.99	¥
8	Laptop bag	ノートパソコン用の	010	089.10	¥

Ready Code Page: 1252

RAINCODE 360 Advanced Showcase

FBA with EBCDIC

FBA with EBCDIC

- This demo produces a dataset of type FBA that is encoded in the EBCDIC
- FBA is a print format, normally 133 bytes in length with 132 printable characters, where the first character is a carriage control, which determines the action of the print head after the line is printed.
  - The valid carriage control characters are:
    - **1**: new page
    - **0**: two new lines
    - **+**: no new line
    - **" "**: new line
- In this demo the character **0** is produced with the WRITE AFTER 2 statement in the COBOL.
- In this demo the data and the carriage control character are encoded in the EBCDIC.
- The following parameter in the COBOL build parameters of the project will ensure that the output of the program is encoded in EBCDIC:
  - :StringRuntimeEncoding=IBM037

FBA with EBCDIC

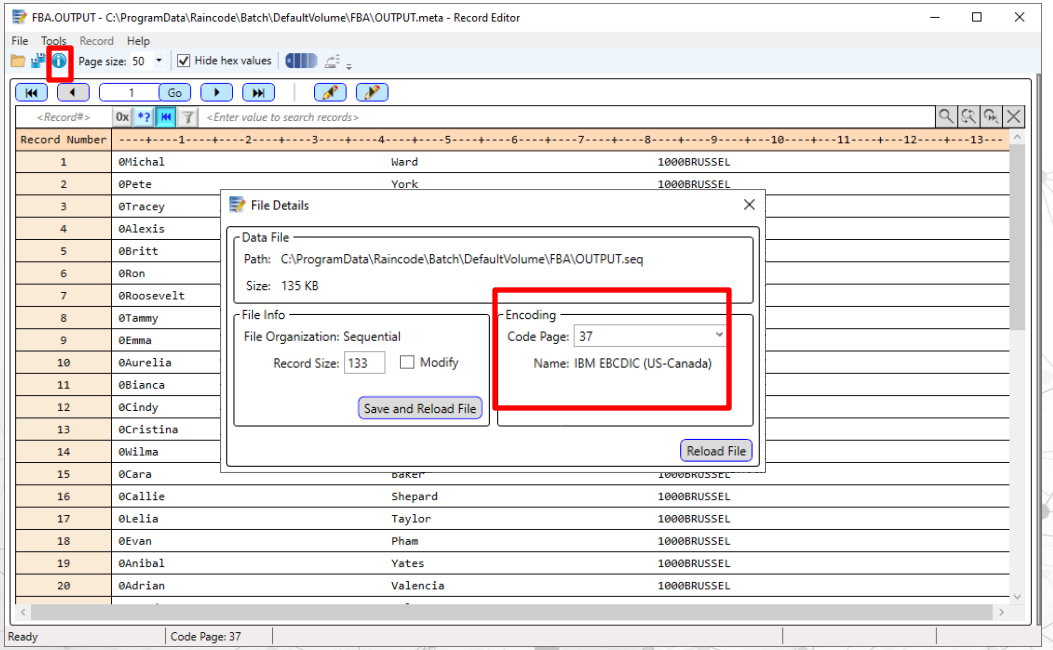
- In this demo, however, we must also ensure that the **0** at the beginning of each line is also encoded in EBCDIC. We set the defaultCodePage in the catalog to EBCDIC, which is not the case in the default configuration file that is shared by most of the demos.
- For this reason, we provide a private catalog configuration file in the solution folder with the following line:
  - <catalogConfiguration dbConnectionDataProvider="RcDbConnections.csv" defaultCodePage="IBM037" ...
- Because our program uses a database connection, we also provide a private copy of the connection file **RcDbConnections.csv** that can be pointed to by the catalog configuration.

FBA with EBCDIC

- The submit utility will look for these files in the bin folder, the current folder from where it is executed. To make sure that it finds these files, we will copy them from the solution folder to the bin folder with a post-build event:
  - copy /Y "$(SolutionDir)\Raincode.Catalog.xml" . && copy /Y "$(SolutionDir)\RcDbConnections.csv"
- The **&&** part is needed so that both files can be copied with a single command.
- Finally, we need to set the **Catalog Configuration** field of the common JCL properties of our project as follows so that it points to the private catalog configuration file in the bin folder:
  - Raincode.Catalog.xml

FBA with EBCDIC – How to demonstrate and see output

- Step 1: Open the solution from the folder Batch – FBA with EBCDIC.
- Step 2: Start the Raincode Debugger
- Step 3: Navigate to the folder C:\ProgramData\Raincode\Batch\DefaultVolume\FBA and double-click on the **OUTPUT. meta**.
- Step 4: The dataset **FBA.OUTPUT** will open in the **Record Editor**



Click on ⓘ to see that the code page is 37 (EBCDIC)

RAINCODE BEO Advanced Showcase

zTrieve- Compiled and Interpreted

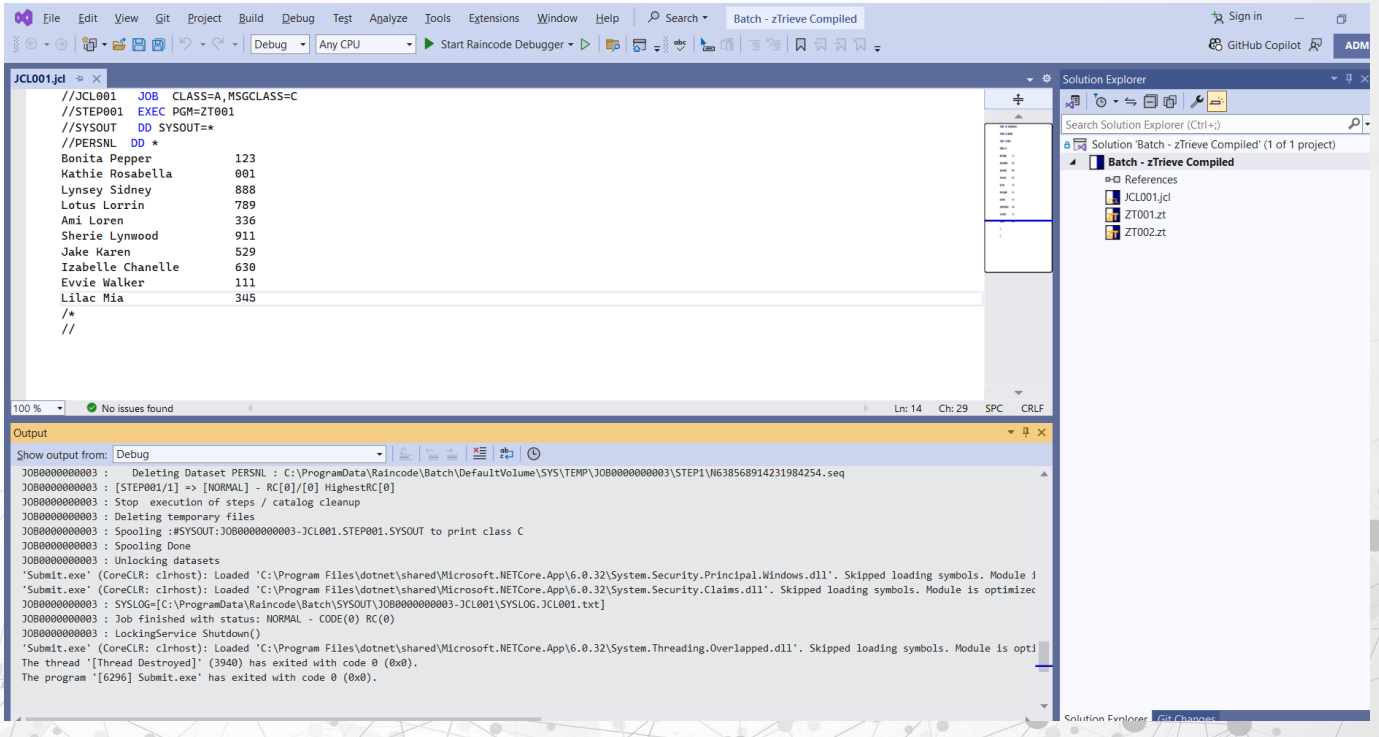
zTrieve-Introduction

- zTrieve reproduces the exact functionality of Easytrieve®.
- zTrieve offers two modes: the **intermediate** and the **compiled** mode.
- The Easytrieve® program is read from the JCL on the fly in the intermediate or interpreted mode.
- On the other hand, the compiled mode will generate standalone DLLs. Further, this DLL can be executed in two ways: via the command line or as part of a JCL.

zTrieve- Compiled

- Open the solution from the folder *C:\repositories\raincode-360\Batch - zTrieve Compiled*

- Start Raincode Debugger to execute the program. It will compile the program, create a DLL, and execute the DLL as part of a JCL.

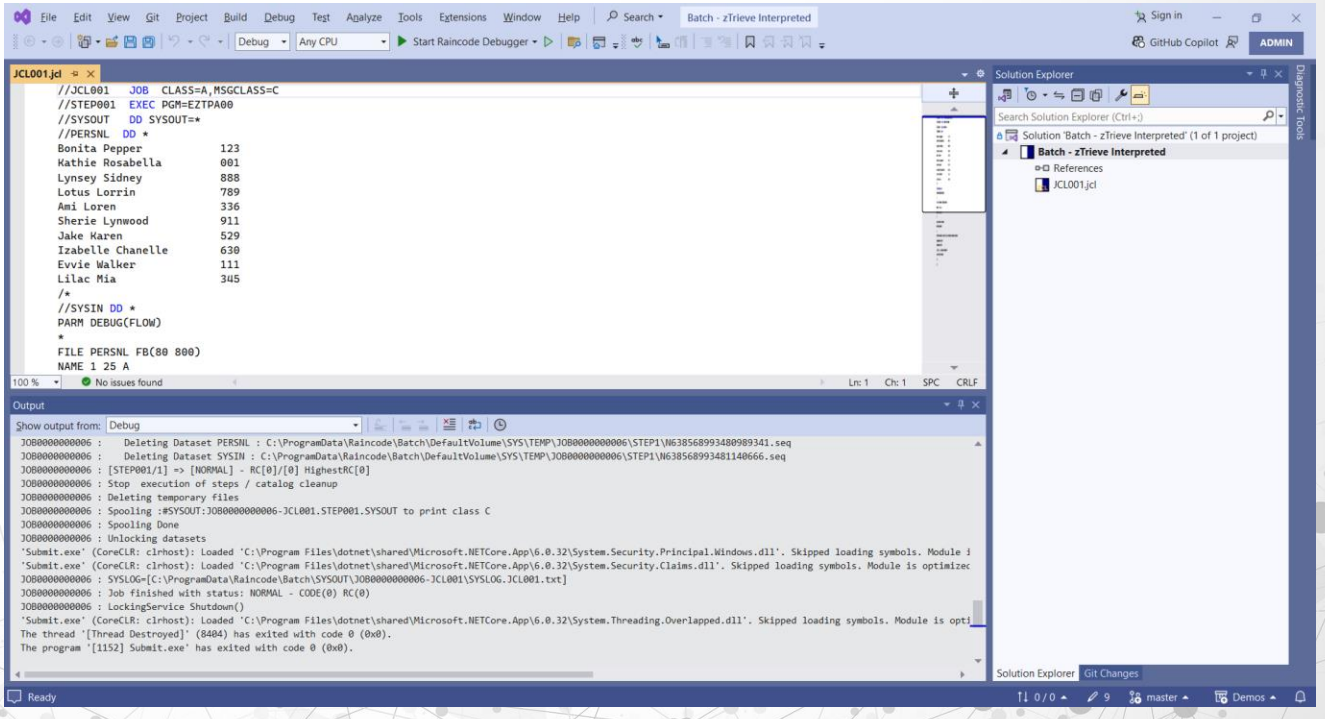


zTrieve- Interpreted

- Open the solution from the folder *C:\repositories\raincode-360\Batch - zTrieve Interpreted*

- Start Raincode Debugger to execute the program.

- In this example, PGM=EZTPA00 interprets the program in SYSIN



Legacy with MultiDatabase Access

Introduction

- This demo showcases connecting COBOL programs to the BANKDEMO and SecondaryDBDemo databases within a single SQL Server instance via IKJEFT01 in a single JCL workflow. It reads data from multiple databases and writes the output to a file.
- The SQL Server connection relies on the following elements:
  - Plan name
    - Defined in the **Raincode.catalog.xml**:

      <programDBConnectionDefinitions>

      <dbConnection JobId="." >DEFAULTPLAN</dbConnection >*

      </programDBConnectionDefinitions>
  - Connection String
    - Defined in the **RcDbConnections.csv**

      **,SqlServer,Data Source=<datasource>;Uid=<user>;Pwd=<password>;Initial Catalog=BankDemo;*

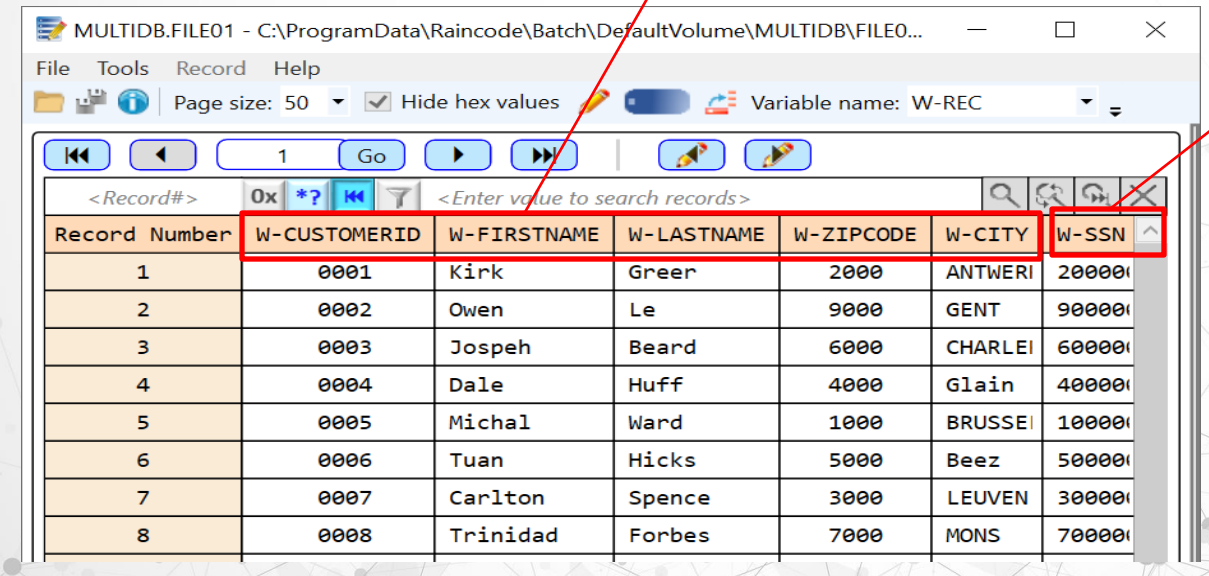
RAINCODE 360 Advanced Showcase

How to run

- Open the solution from the folder Batch – Legacy with Multi Database Access
- Start the Raincode Debugger
- Output files are located at C:\ProgramData\Raincode\Batch\DefaultVolume\MULTIDB

Fetched from the BANKDEMO.

Fetched from the *SecondaryDB Demo*



Record Number	W-CUSTOMERID	W-FIRSTNAME	W-LASTNAME	W-ZIPCODE	W-CITY	W-SSN
1	0001	Kirk	Greer	2000	ANTWERI	20000(
2	0002	Owen	Le	9000	GENT	90000(
3	0003	Jospeh	Beard	6000	CHARLEI	60000(
4	0004	Dale	Huff	4000	Glain	40000(
5	0005	Michal	Ward	1000	BRUSSEI	10000(
6	0006	Tuan	Hicks	5000	Beez	50000(
7	0007	Carlton	Spence	3000	LEUVEN	30000(
8	0008	Trinidad	Forbes	7000	MONS	70000(

Unload MultiDatabase

Introduction

- This demo showcases how to use the DSNTIAUL utility to copy data from multiple tables across different databases into a sequential file.
- The SQL server connection relies on the following elements:
  - Plan name
    - Defined in the **Raincode.catalog.xml**:

      <programDBConnectionDefinitions>

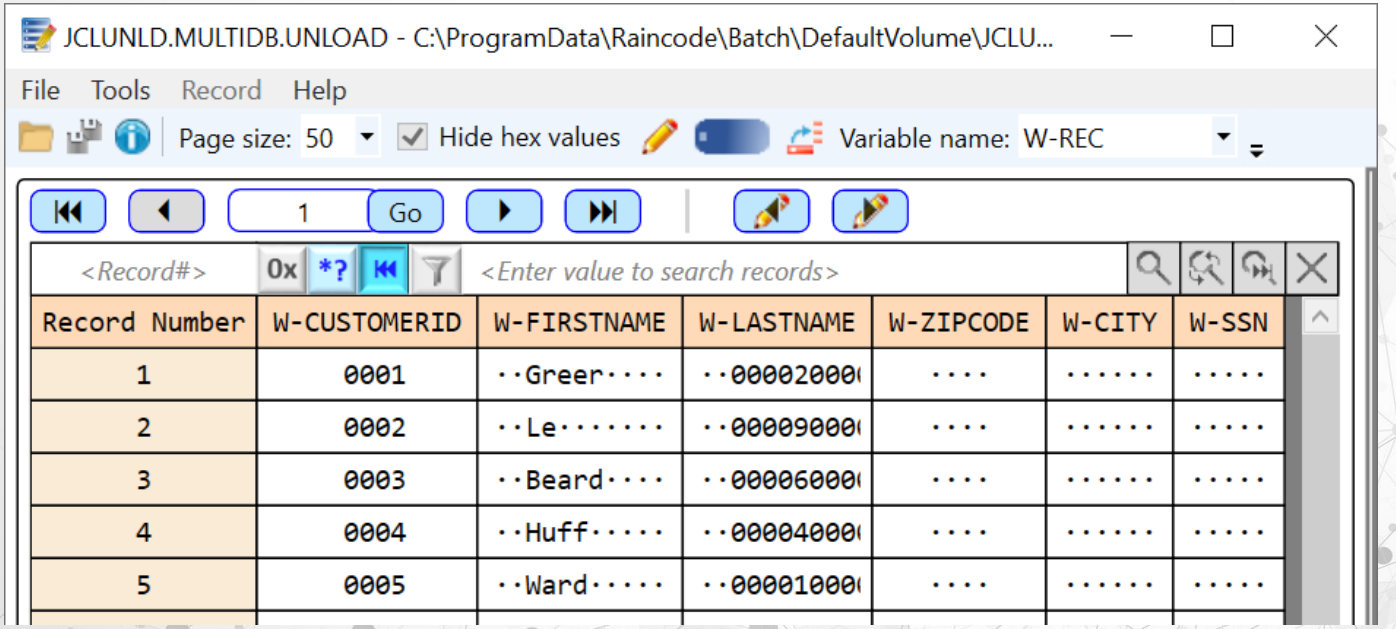
      <dbConnection JobId="." >DEFAULTPLAN</dbConnection >*

      </programDBConnectionDefinitions>
  - Connection String
    - Defined in the **RcDbConnections.csv**

      **,SqlServer,Data Source=<datasource>;Uid=<user>;Pwd=<password>;Initial Catalog=BankDemo;*

How to run

- Open the solution from the folder Batch – Unload with MultiDatabase
- Start the Raincode Debugger
- Output files are located at C:\ProgramData\Raincode\Batch\DefaultVolume\JCLUNLD\MULTIDB



Record Number	W-CUSTOMERID	W-FIRSTNAME	W-LASTNAME	W-ZIPCODE	W-CITY	W-SSN
1	0001	··Greer····	··00002000(	····	·····	·····
2	0002	··Le·····	··00009000(	····	·····	·····
3	0003	··Beard····	··00006000(	····	·····	·····
4	0004	··Huff·····	··00004000(	····	·····	·····
5	0005	··Ward·····	··00001000(	····	·····	·····

W-CUSTOMERID and W-SSN are fetched from the *SecondaryDBDemo*

W-LASTNAME and W-ZIPCODE are fetched from the *BANKDEMO*

RAINCODE 360 Advanced Showcase

zBridge

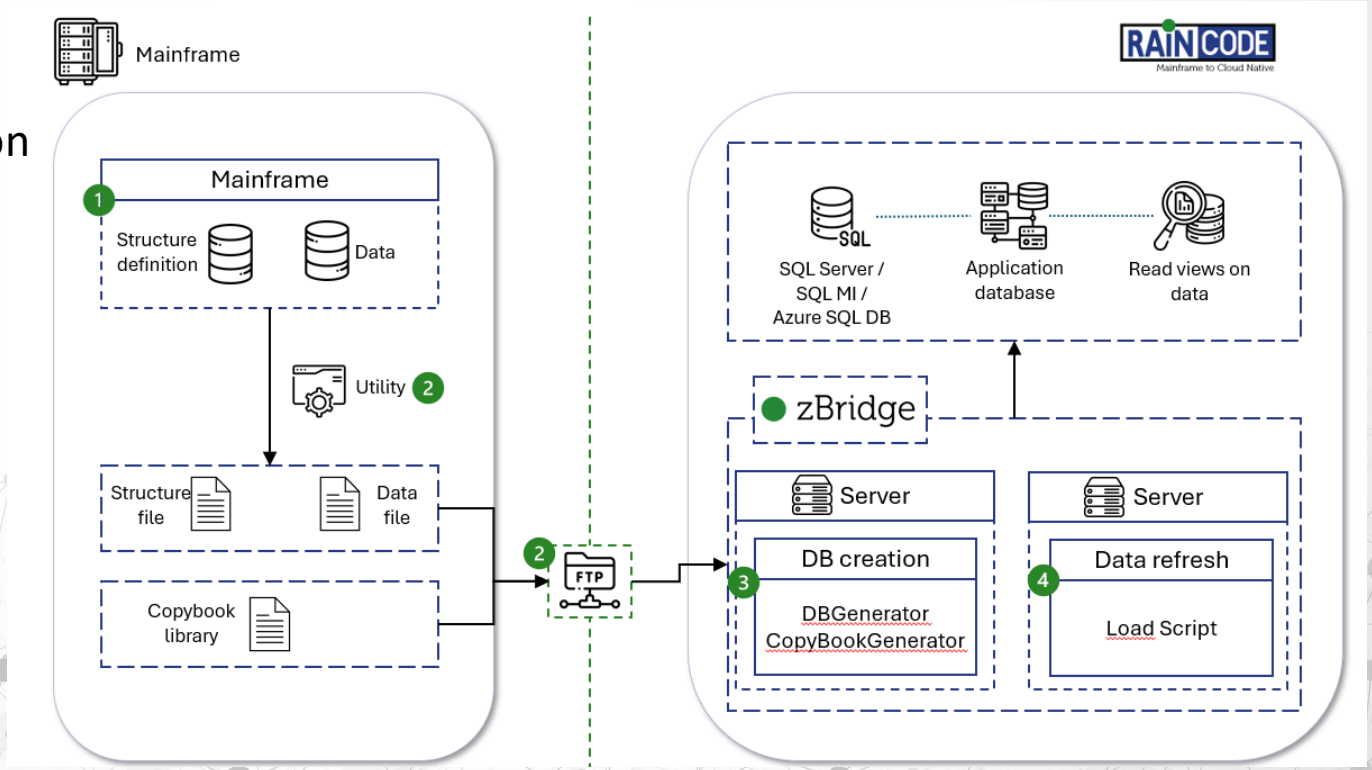
zBridge-Introduction

- zBridge is an innovative solution from Raincode that exposes mainframe data as SQLServer views.
- It transforms complex, non-relational datasets into easily accessible relational formats and eliminates the need for deep data restructuring or long, laborious cleanups.
- This, in turn, allows organizations to gain value from their data without fully migrating or disrupting existing systems.

zBridge-Introduction

Using zBridge is a four-step process:

1. Non-Relational Data Extraction
2. Data Transfer
3. Automated DB Creation
4. Data loading



RAINCODE 360 Advanced Showcase

zBridge- How to run

To run this demo, follow these steps:

- Run the script `generateTable.ps1` to produce a table creation script named `table_creation.sql` based on the catalog configuration (C:\Raincode360\Repositories\Demos\zBridge\zBridgeFile\catalog\Raincode.Catalog.xml). There is no need to execute the `table_creation.sql ` script since the table creation was already taken care of during this Raincode360 VM provisioning.

- Next, run the script `generate view.ps1` to generate the SQL view creation script named `ACCOUNT_view.sql` based on the copybook. There is no need to execute `ACCOUNT_view.sql` script since the view creation was already taken care of during this Raincode360 VM provisioning.

zBridge- How to run

- Finally, run the script `file.LoadData.ps1` to load the data into the `[VsamSql].[REC_ACCOUNT]` table. The view `[VsamSql].[REC_ACCOUNT_V]` can be used to query the data in a more user-friendly format.

```
select top(5) [RID], [W_ACCOUNT_ID], [W_CUSTOMERID_ID], [W_AMOUNT]
from [REC_ACCOUNT_V]
```

100 %

Results | Messages

	RID	W_ACCOUNT_ID	W_CUSTOMERID_ID	W_AMOUNT
1	1	BE00-1000-0000-1438	832	+00001064.80
2	2	BE00-1000-0000-2643	70	+00004541.89
3	3	BE00-1000-0000-3557	922	+00002802.74
4	4	BE00-1000-0000-4464	818	+00002631.55
5	5	BE00-1000-0000-4488	899	+00005565.00

Raincode QIX Emulator

- Demos:
  - Raincode QIX
  - How to debug a CICS application
  - Running a QIX program from a command line
  - Redefining LINK
  - Redefining SEND and RECEIVE
  - Legacy with database access
  - Asynchronous Processing with START/ RETRIEVE
  - VSAM – C# calling legacy with VSAM access using CICS LINK
  - *Create REST API from Legacy Program*
  - *Create REST API from Legacy Program –Rainbox Generated*
  - *Online Internal Reader*
  - *CWA*

Raincode QIX Emulator

- Raincode QIX is a CICS emulator for the .NET and the Azure Platforms
- It is the ideal companion for the Raincode Legacy Compilers
- In addition to your existing COBOL and PL/I applications, QIX also allows you to develop new transactions in C#
- Provides a robust, distributed, and scalable architecture
- Support for dynamically added processing servers

Raincode QIX Emulator

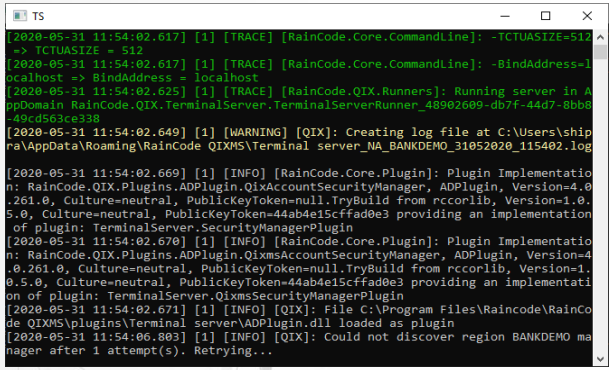
- Usage of the **Raincode QIX Emulator**
- Six CICS transactions written in COBOL
- Six BMS maps
- One transaction written in PL/I (screenless)
- Programming Features:
  - Obtaining and passing a COMMAREA
  - Embedded SQL
  - SQL Server database
- CICS LINK, XCTL, SEND and RECEIVE
- Application Features:
  - Validation, error messages, highlighting of errors
  - Cursor-sensitive fields
  - Navigation via PF-Keys
  - List/selection screens with filtering

Raincode QIX – Starting the Raincode Bank Demo

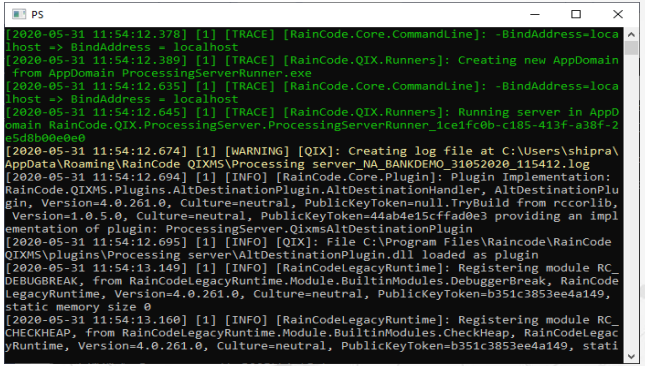
- To start the demo, launch the “Start CICS demo” on the desktop.

Three additional windows will open up (with a short delay)

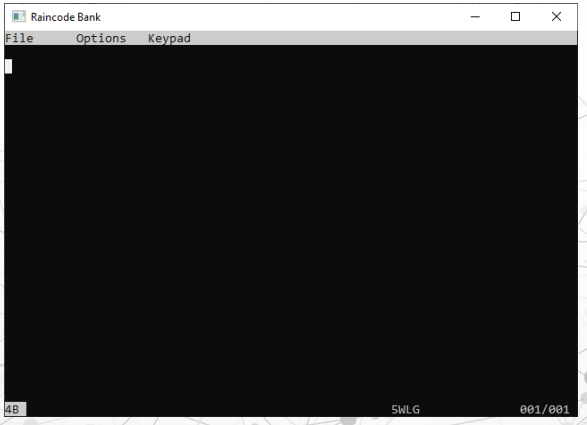
- Terminal Server (TS)



- Processing Server (PS)



- Raincode Bank (in 3270 terminal emulator, connects to TS)

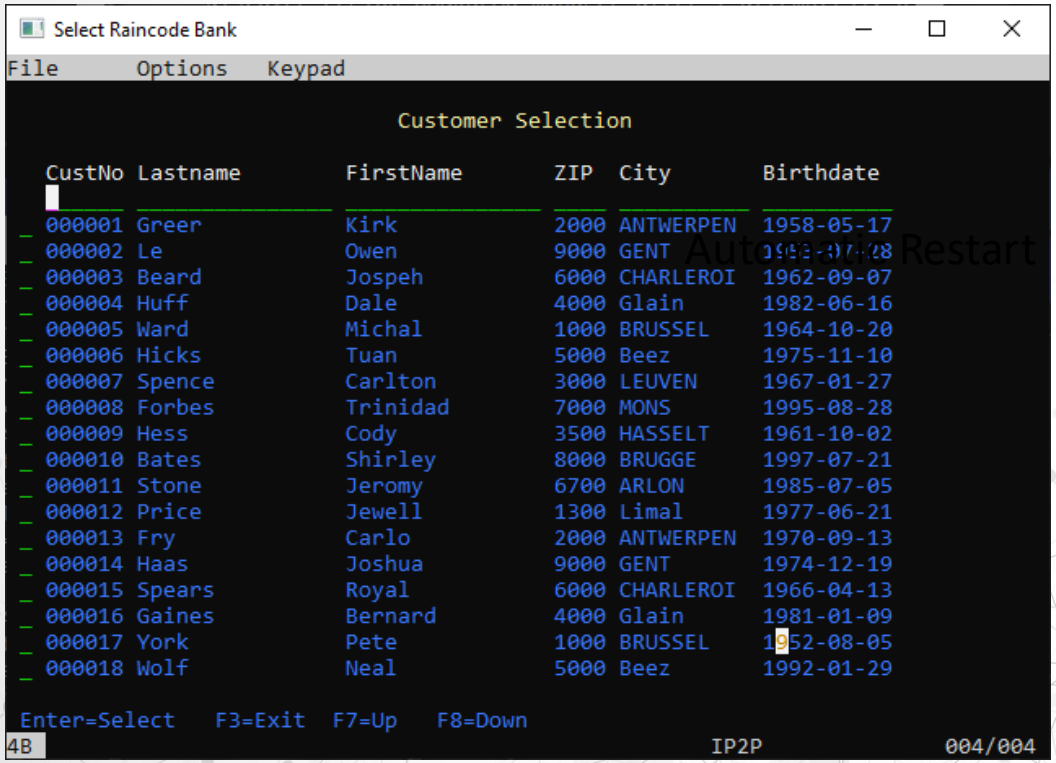


TS/PS are minimized by default. Together, they implement the Raincode QIX emulator.

RAINCODE 360 Advanced Showcase

Raincode QIX – Raincode Bank Demo (3270 terminal)

- The Raincode Bank window is a 3270 terminal emulator session that allows you to interact with the QIX.
- Type C004 and press Enter to start the application



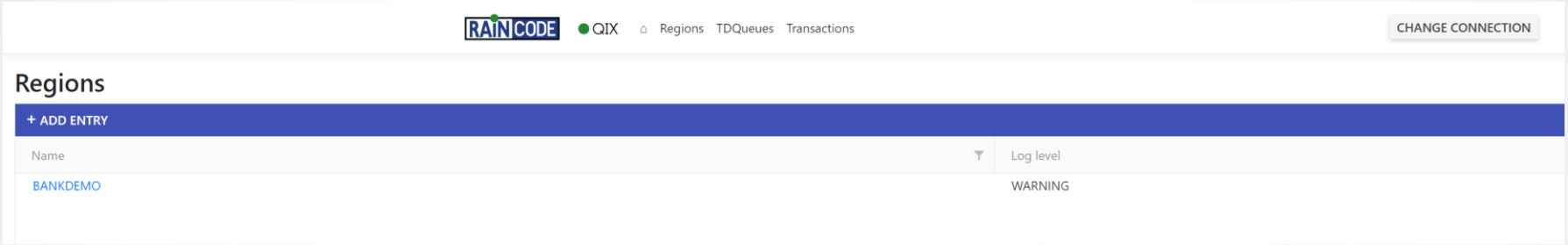
- *Use the filter keys above the list to search for names and/or zip codes*
- *Position the cursor and press Enter to select a beneficiary account from a list.*
- *A transaction between two accounts can be created.*

You can also see the regions and transaction details in the Raincode Console.

RAINCODE 360 Advanced Showcase

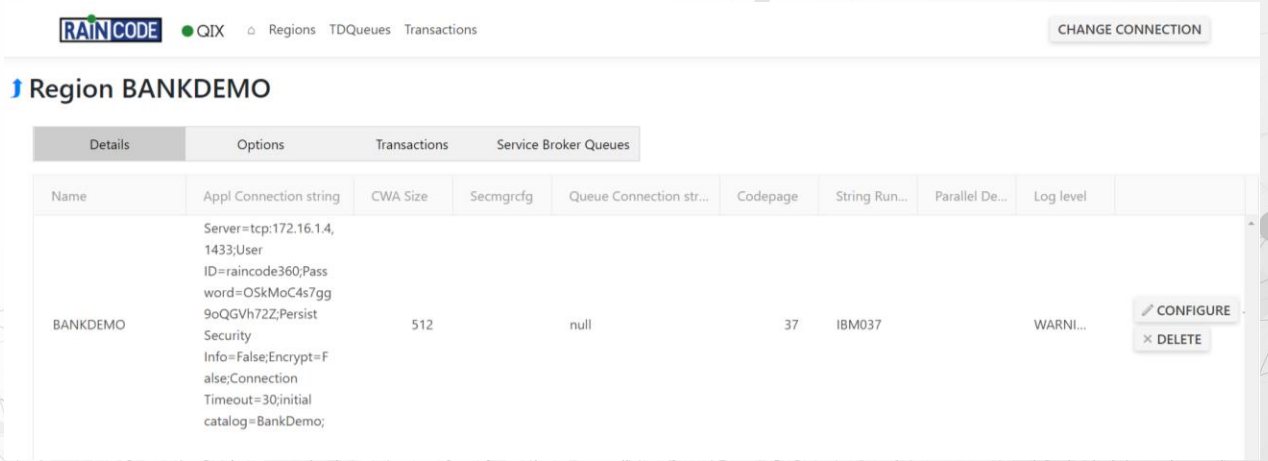
Raincode Bank Demo (Raincode Console)

BANKDEMO is the region which appears in the Raincode Console - QIX Panel when you execute the [Raincode Bank Demo].



Once you click the region name, it will provide you with more detailed information about that region like:

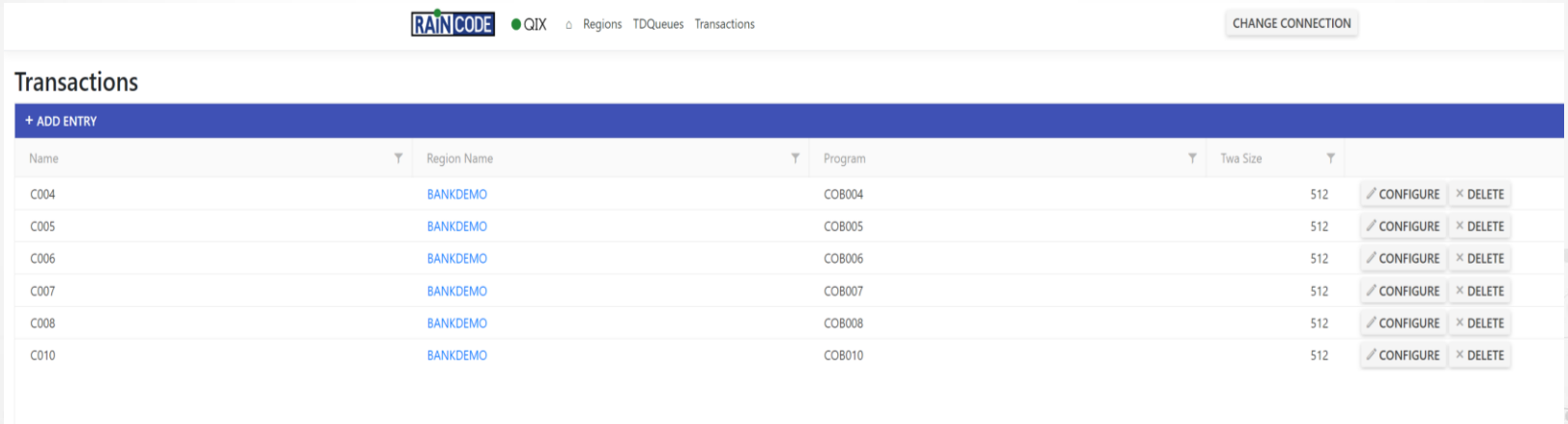
- Details
- Options
- Transactions
- Service Broker Queues



RAINCODE 360 Advanced Showcase

Raincode Bank Demo (Raincode Console)

- The below transactions are configured while creating the Raincode Bank demo. Through the Raincode console, you can add a new transaction or configure the existing transaction.



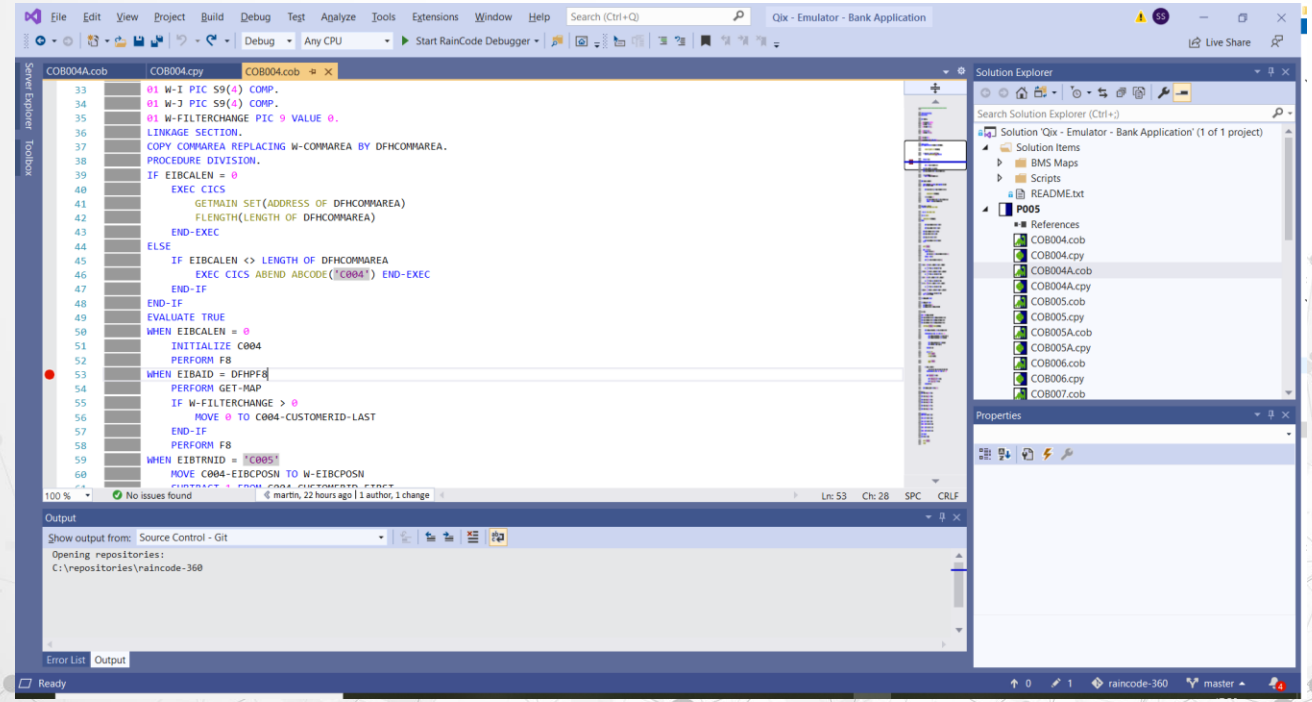
RAINCODE 360 Advanced Showcase

How to debug a CICS application

RAINCODE 360
Advanced Showcase

How to debug a CICS application

- Step 1: Open the solution from the folder Qix - Emulator - Bank Application.
- Step 2: Place a breakpoint at line 53 of program COB004.cob (`WHEN EIBAID = DFHPF8`) by clicking in the left margin. A red dot will appear.

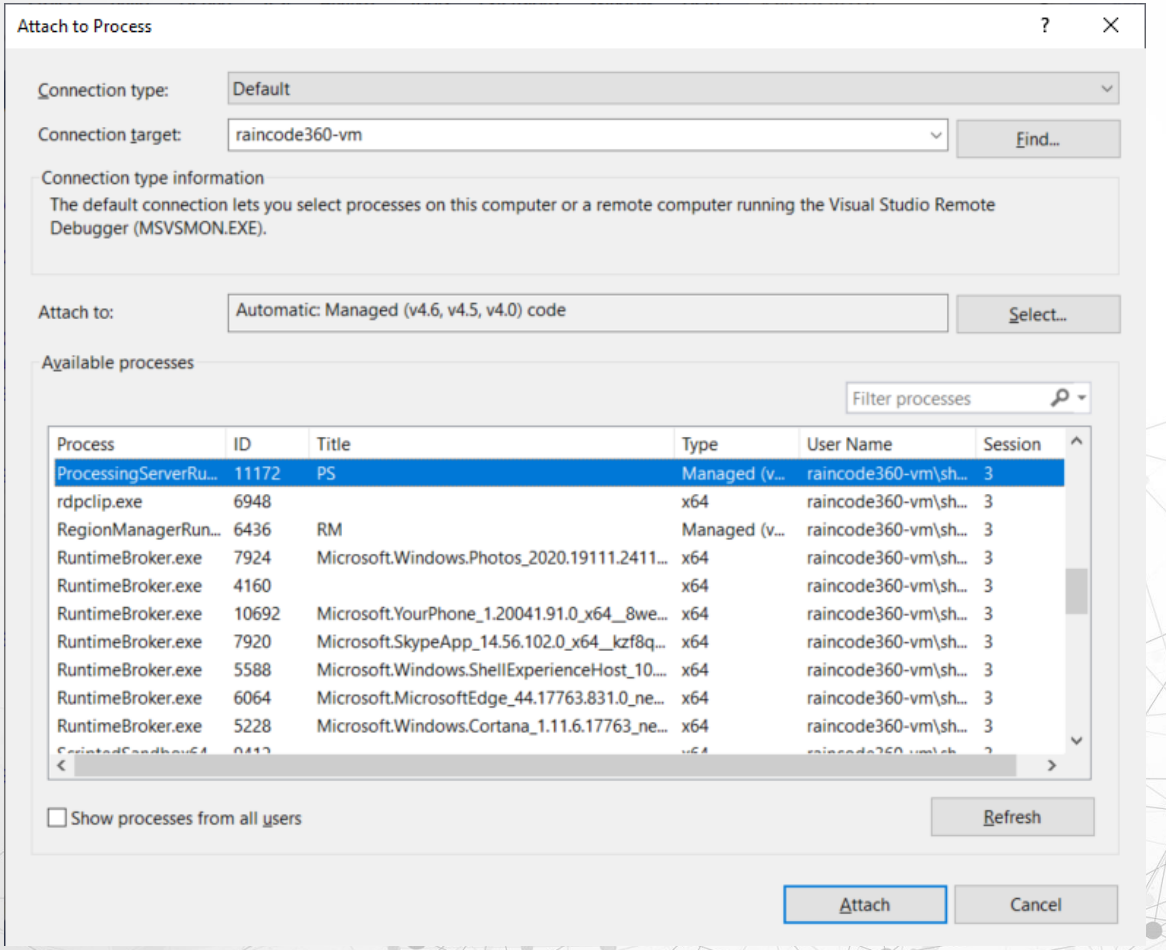


- *Make sure that QIX is running and connected*
- *Start the application in the 3270 terminal emulator window*

RAINCODE 360 Advanced Showcase

How to debug a CICS application

- Step 3: In the Visual Studio, select Debug → Attach to Process (Ctrl+Alt+P)

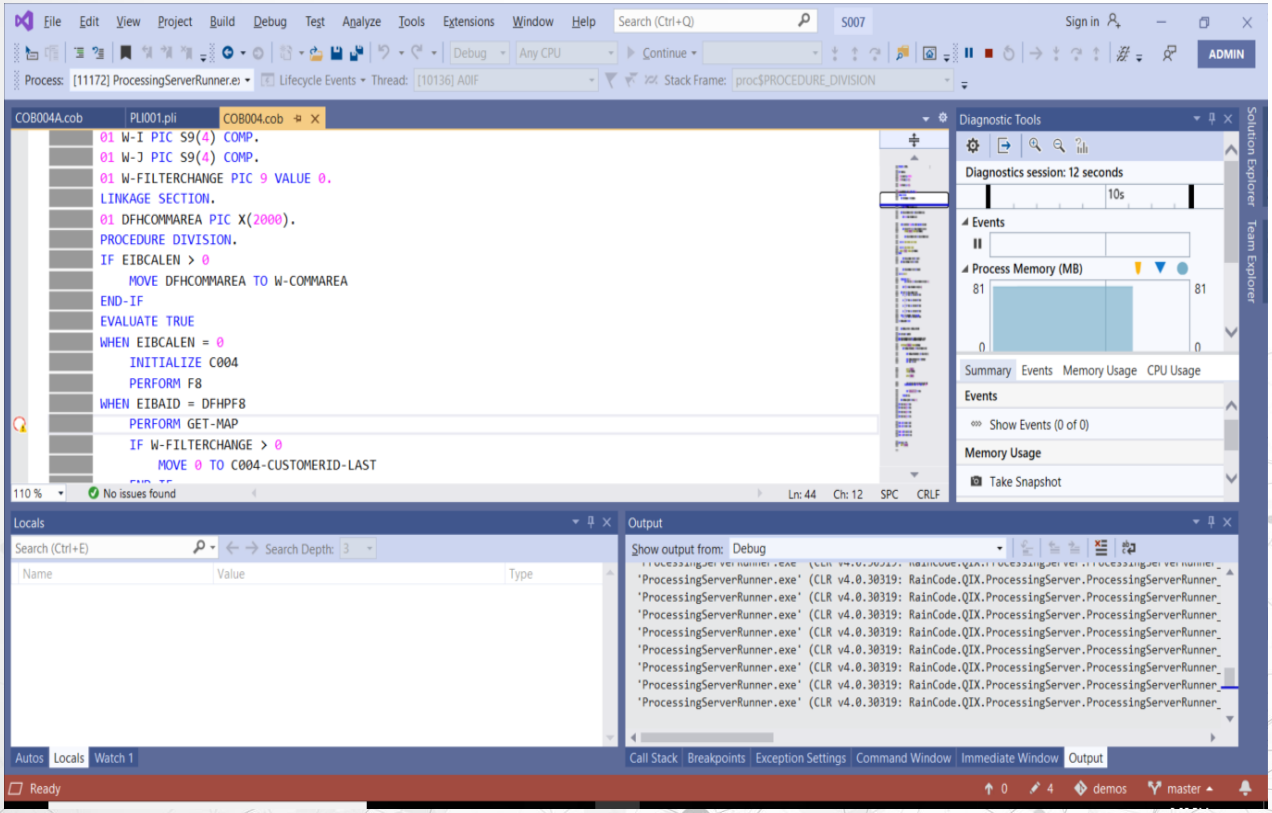


Select `ProcessingServerRunner.exe` from the list of processes and click on the Attach button

RAINCODE 360 Advanced Showcase

How to debug a CICS application

- Visual Studio switches to debug mode.

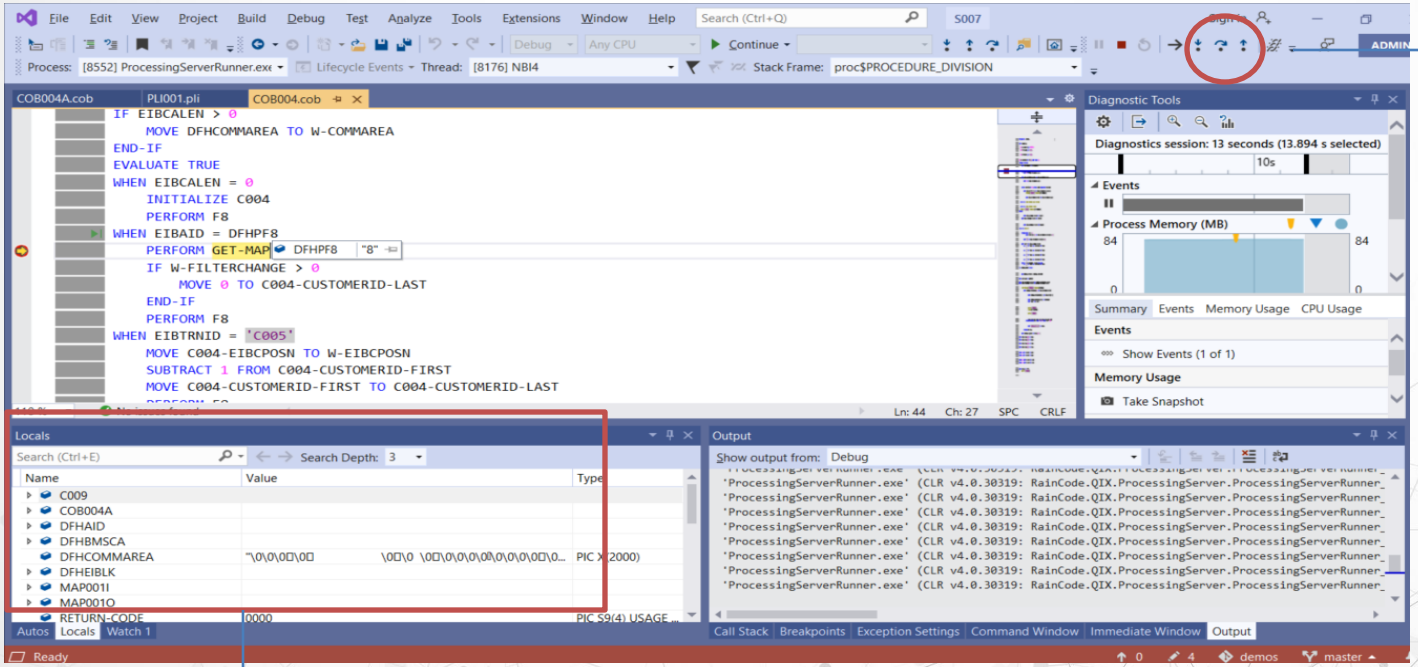


Because of the pseudo-conversational nature of the application, the program COB004.cob is not running at this point

RAINCODE 360 Advanced Showcase

How to debug a CICS application

- Step 4: Press **F8** on the main screen of the Raincode Bank application to scroll down. This will activate COB004.cob, and the execution will stop at the breakpoint.



Use the `Step` buttons to step through the program

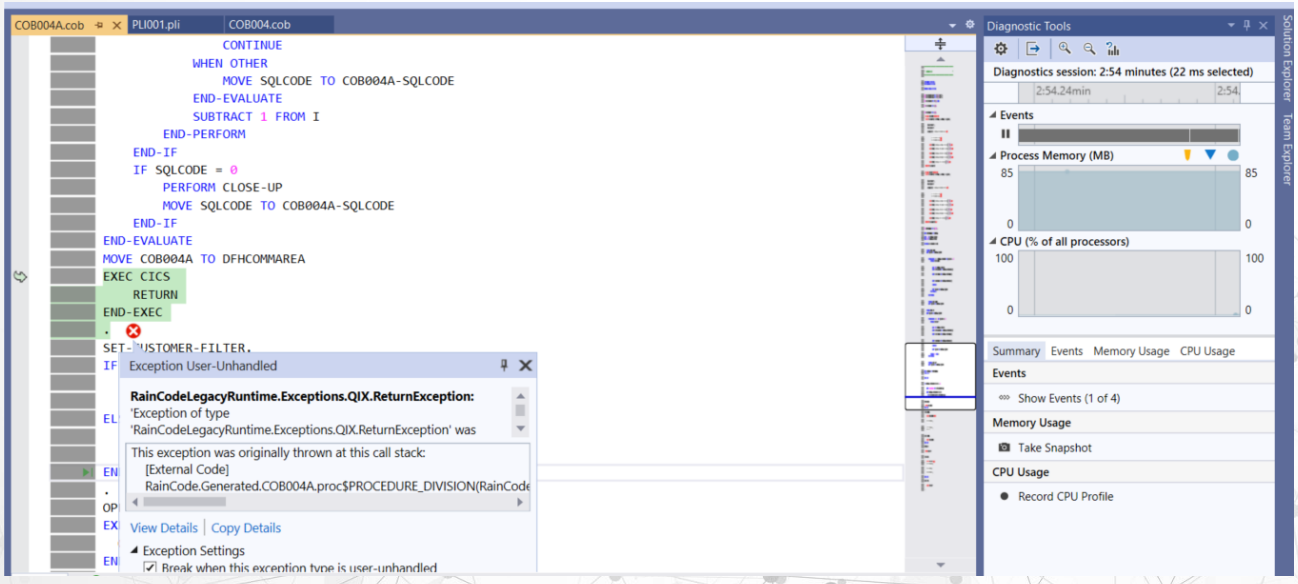
Hover over any variable in the source text of COB004.cob to see its contents.

*Check the **Locals** window in Visual Studio to see a list of local variables and their corresponding values*

RAINCODE 360 Advanced Showcase

How to debug a CICS application

- Stepping over the 'EXEC CICS RETURN' will trigger an exception. This is normal.
- Pressing “Continue” will end the program without detaching the debug session.
- Continue debugging by pressing **F8** or **Enter** in the Raincode Bank application.



Stop debugging by pressing the stop icon in the debug toolbar

RAINCODE 360 Advanced Showcase

Running a QIX program from a command line

Running a QIX program from a command line

- This demo runs a COBOL/QIX program from the command line without a runner
- Step 1: Open the solution from the folder QIX -Legacy – Run from the Command Line

There are two ways to RUN the program:

- Step 2a: Place a breakpoint on the last line of MAIN.Cob and start the Raincode debugger
- Step 2b: Run `run.bat` from the command line

Redefining LINK

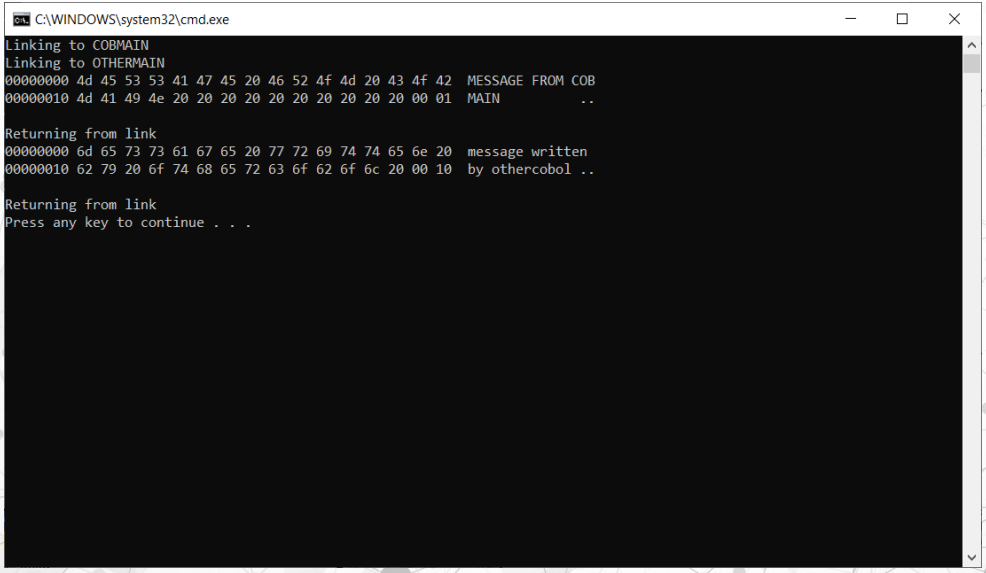
Redefining LINK – Introduction

- Demonstrate how the behavior of CICS commands can be overwritten using Qix factories.
- The new link command writes a message plus a hex dump of the commarea before and after the actual link.
- The original link mechanism is invoked by the base.Execute() in CustomLink.cs.
- For more details, see Qix-Factory.

Redefining LINK – How to run and see output

- Step 1: Open the solution from folder Qix - Factories - Redefining LINK
- Step 2: Select start without debugging (Ctrl + F5).

  It will avoid automatic breakpoints and will also keep the console window open after the program ends
- You can check the messages in the console window



Redefining SEND and RECEIVE

Redefining SEND and RECEIVE – Introduction

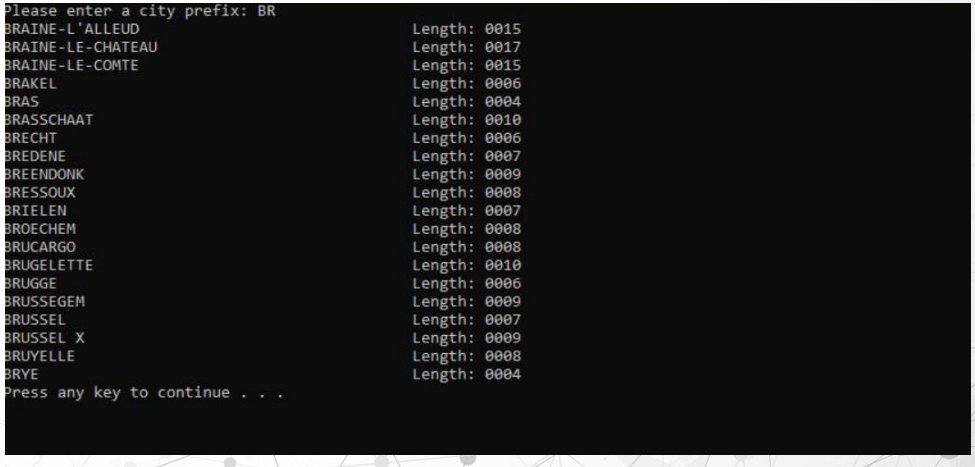
- Redefine the behaviour of the standard CICS commands SEND and RECEIVE by providing custom factories.
- SEND and RECEIVE are traditionally used to send and receive text messages to and from the terminal.
- The behavior of these commands is changed so that they can send and receive data structures to and from an in-memory queue.
- The main program asks the user for a city prefix and calls a subroutine (via CICS LINK) to retrieve all cities matching this prefix from the database.
- The subroutines push a variable number of results on the internal queue.
- The main routine reads the results from the queue and shows them on the screen.

Redefining SEND and RECEIVE – How to run and see output

- Step 1: Open the solution from the folder Qix - Factories - Redefining SEND and RECEIVE
- Step 2: Select start without debugging (Ctrl + F5).

  It will avoid automatic breakpoints and will also keep the console window open after the program ends.

- Enter a prefix of a Belgian city ('BR' for Brussels will work just fine)



Legacy with the database access

Legacy with the database access – Introduction

- C# calling legacy with the database access using CICS LINK
- Demonstrates how a COBOL program can be called from a C# main program (a proxy)
- The C# program uses the CICS LINK mechanism to invoke the COBOL Program
- Demonstrates how to prepare an execution context with a database connection at the C# level
- In this demo, the legacy program fetches an item (minimum zip code) from the database, displays it on the console and exits.

Legacy with database access – How to demonstrate

- Step 1: Open the solution from the folder Qix - Legacy with Database Access - Csharp Main
- Step 2: Select Start without debugging (Ctrl + F5).

  It will avoid automatic breakpoints and will keep the console window open after the program ends

Asynchronous Processing with START / RETRIEVE

Asynchronous Processing with START / RETRIEVE– Introduction

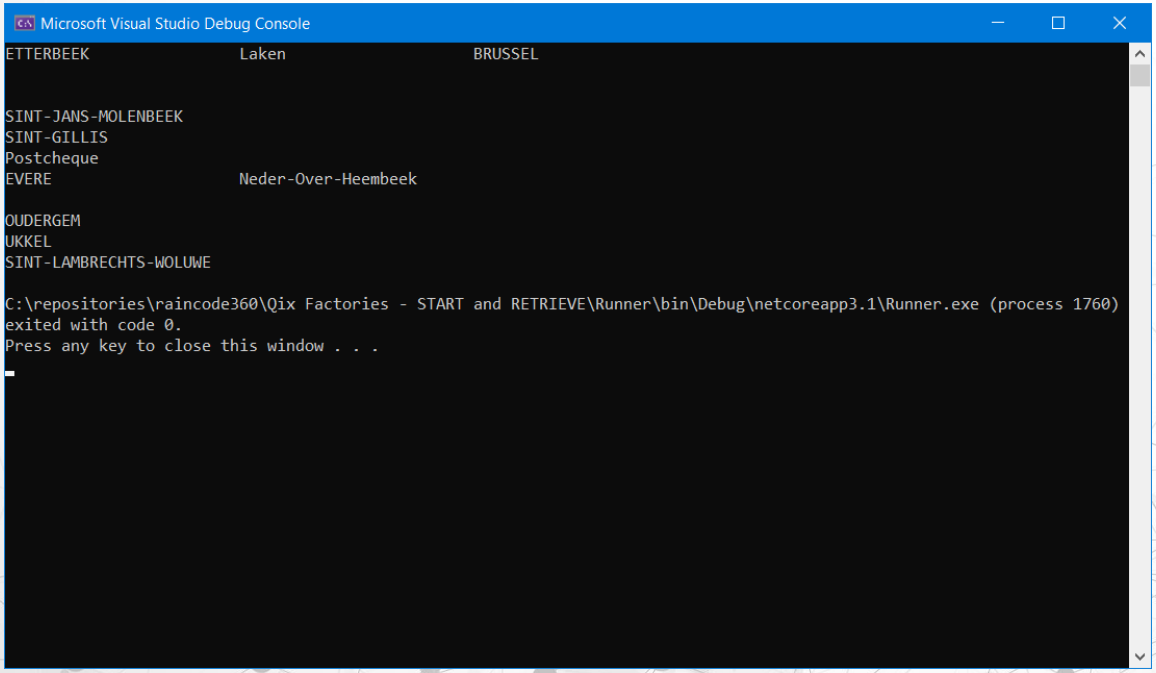
- This demo implements the CICS START and RECEIVE commands with Qix factories
- The START command implements a 'fire-and-forget' mechanism by starting a new asynchronous transaction: **START TRANSID('name') FROM(data)**, whereby **data** can be any part of working storage.
- If CICS cannot execute the new transaction immediately, it will be queued to start at a later point in time. The original transaction does not wait for the new transaction to be started. The return status of the command indicates that the new transaction has been successfully queued.
- The transaction receives the data with the following command: **RETRIEVE INTO(data)**, whereby **data** refers to the working storage of the transaction.

Asynchronous Processing with START / RETRIEVE– Introduction

- PRACTICAL APPLICATIONS OF START AND RECEIVE
  - The START and RECEIVE commands are often used to create a listener for incoming data on a queue.
  - The listener will launch a separate task to process the data whenever data is received.

Asynchronous Processing with START / RETRIEVE- How to demonstrate

- Step 1: Open the solution from the folder Qix Factories - START and RETRIEVE
- Step 2: Select Start without debugging (Ctrl + F5).
- Expected output:



- The program launches ten separate tasks to retrieve the cities associated with certain zip codes.
- The output can appear out of order depending on which task finishes first.

RAINCODE 360 Advanced Showcase

VSAM - C# calling legacy with VSAM access using CICS LINK

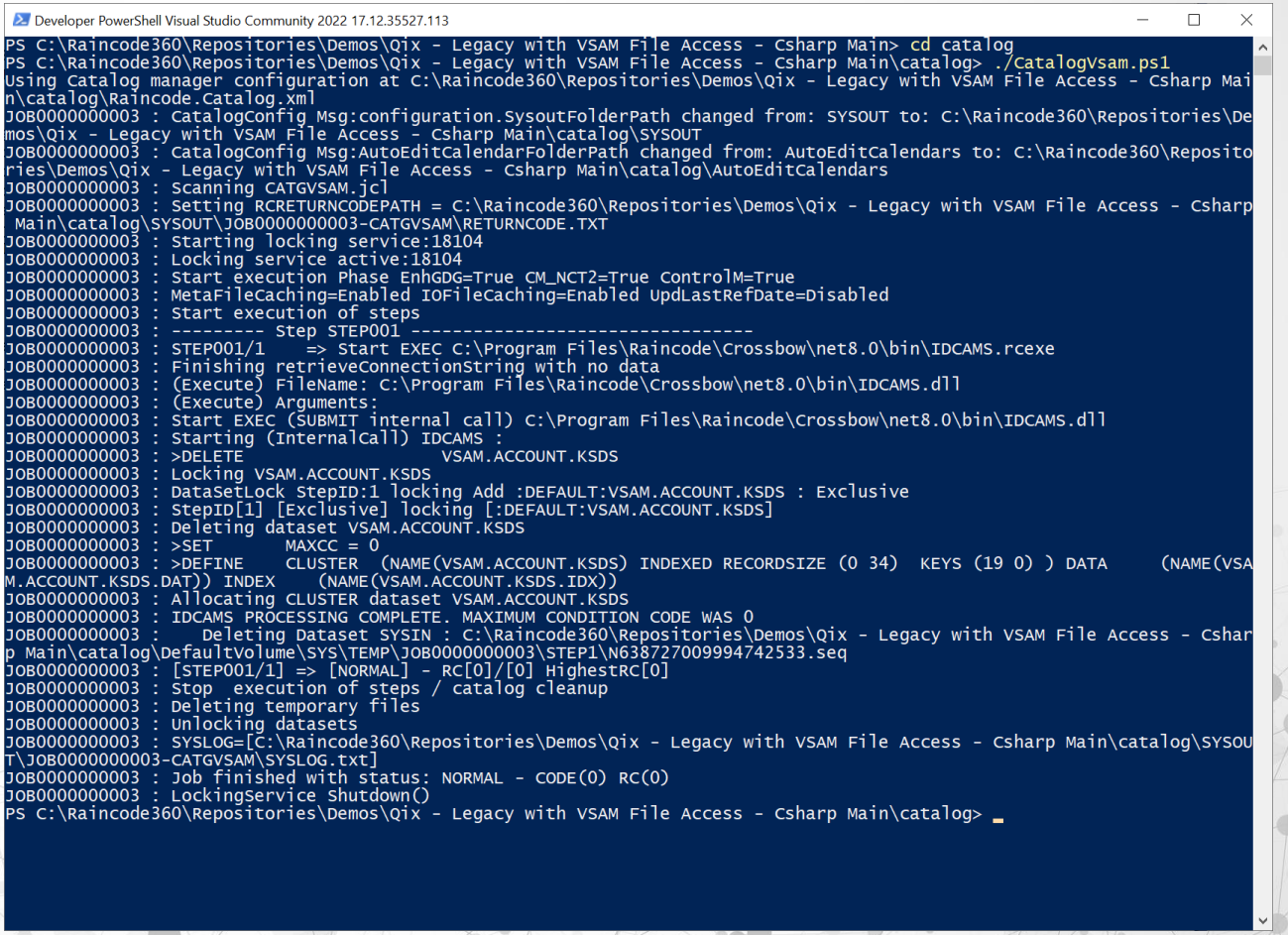
RAINCODE 360
Advanced Showcase

C# calling legacy with VSAM access using CICS LINK– Introduction

- The program demonstrates how to Write and Read VSAM files from CICS modules.
  - The legacy program COBWRITE fetches account information from the database and writes it to the VSAM file.
  - The legacy program PLIREAD reads and displays the VSAM file for a particular record on the console.

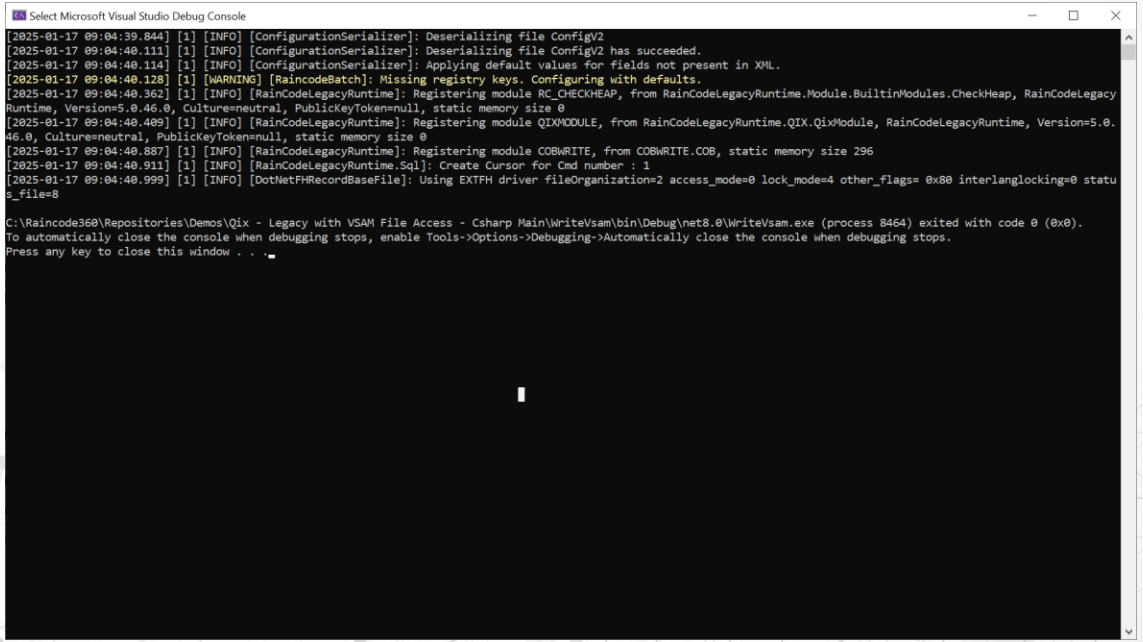
C# calling legacy with VSAM access using CICS LINK– How to demonstrate

- Step 1: Open the Powershell console via Tools->Command Line->Developer Powershell.
- Step 2: Navigate to the catalog folder using the command: *cd catalog.*
- Step 3: Execute the script *CatalogVsam.ps1* to catalog the empty VSAM file .



C# calling legacy with VSAM access using CICS LINK– How to demonstrate

- Step 4: Rebuild the solution by selecting Build->Rebuild solution from the toolbar.
- Step 5: In the solution explorer, right-click *WriteVsam* project and select *Set as Startup Project.*
- Step 6: Run the *WriteVsam project* by clicking the ***play*** button on the toolbar. This process will write the VSAM file with the records fetched from the database.



Note:

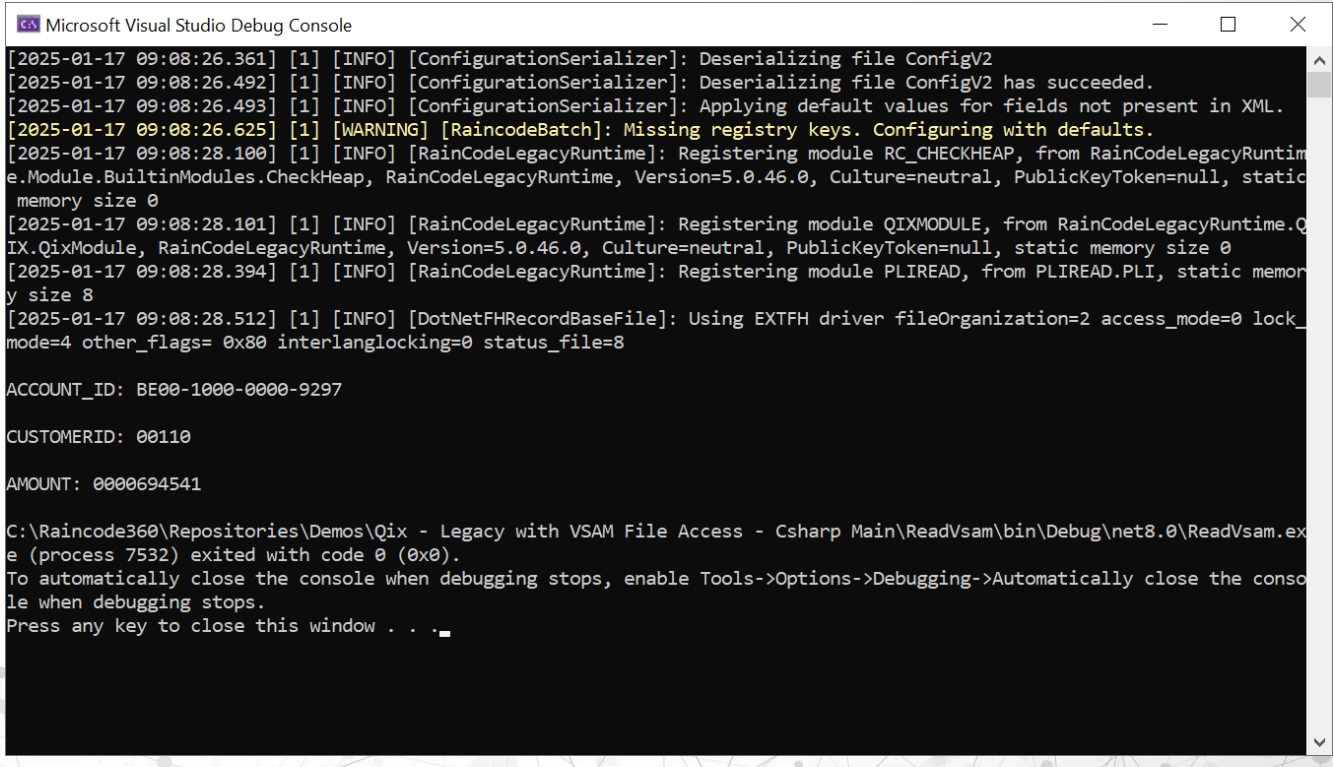
Ignore the `*RainCodeLegacyRuntime.Exceptions.QIX.ReturnException*` if you encounter it during debugging. This is not a real exception, but it is expected.

RAINCODE 360 Advanced Showcase

C# calling legacy with VSAM access using CICS LINK– How to demonstrate

- Step 7: In the solution explorer, right-click *ReadVsam* project and select *Set as Startup Project.*

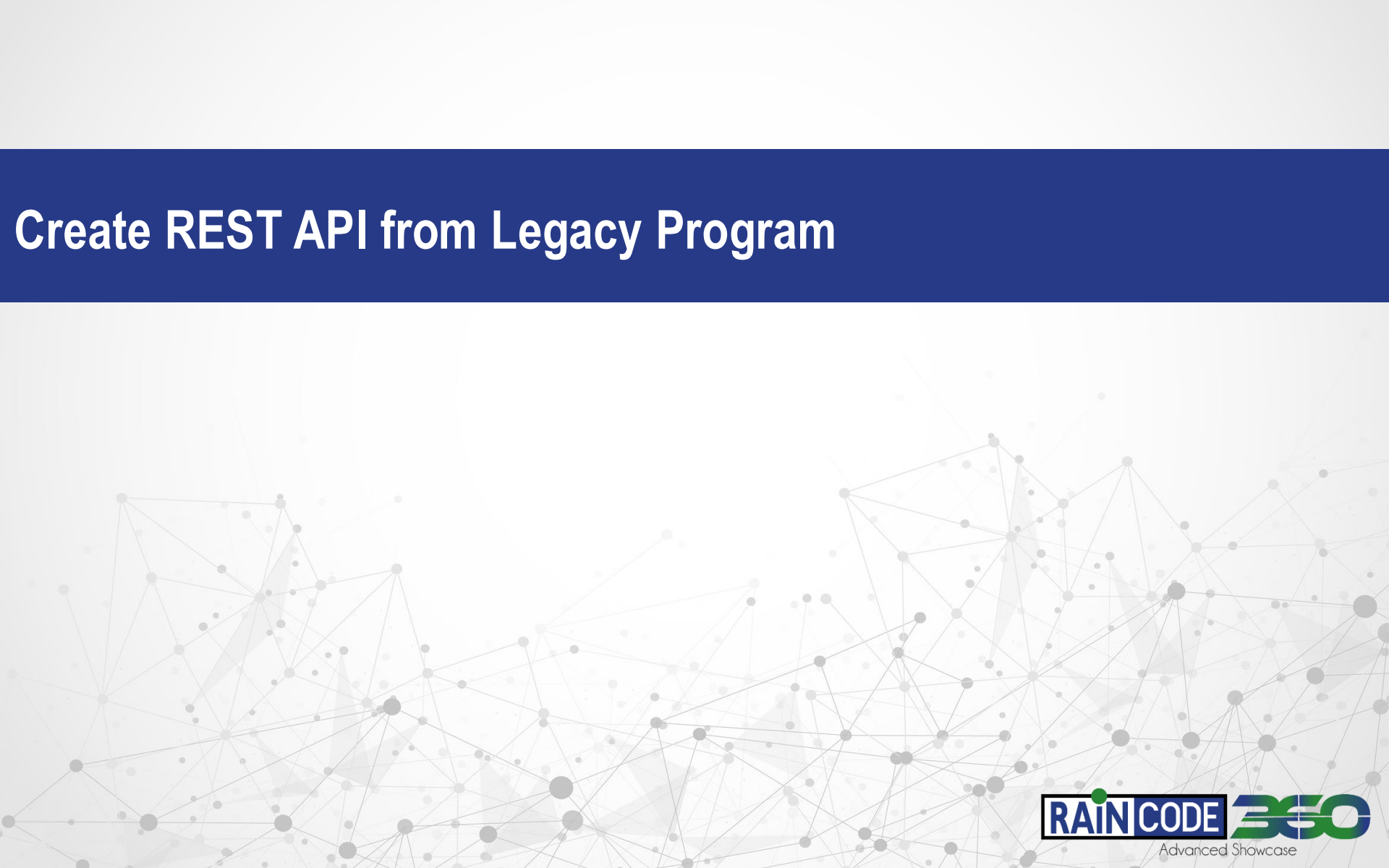
- Step 8: Run the *ReadVsam* project by clicking the **play** button on the toolbar; it will read the VSAM file using the key "BE00-1000-0000-9297" and display the file content on the console.



Note:
Before re-running *WriteVsam*, ensure to run the script *CatalogVsam.ps1*; otherwise, you will encounter a *duplicate record QIX exception*.

RAINCODE 360 Advanced Showcase

Create REST API from Legacy Program



Create REST API from Legacy Program - Introduction

- The program demonstrates how to create and invoke a REST API on top of the CICS module.

- **Prerequisite:**

  - In the solution explorer, right-click on Legacy project and select Build. This process will perform the following:
    - Compiles the PLI001 and COB004A modules using the Raincode PLI and COBOL compiler.
    - Executes the script *CopyDlls*, which transfers the generated DLLs to the LegacyDll folder of the *RCMainframeAP*I project.

    Note:

    This step must be repeated whenever there is a change in the PLI001 or COB004A modules.

  - Open PowerShell via Tools->Command Line->Developer PowerShell and execute the *Helper.ps1* script. This script performs the following:
    - Generates the helper classes for PLI001 and COB004A modules using the Raincode PLI and COBOL compilers.
    - It also copies the generated *helperclasses* to the **HelperClas**s folder of the **RCMainframeAPI** project.

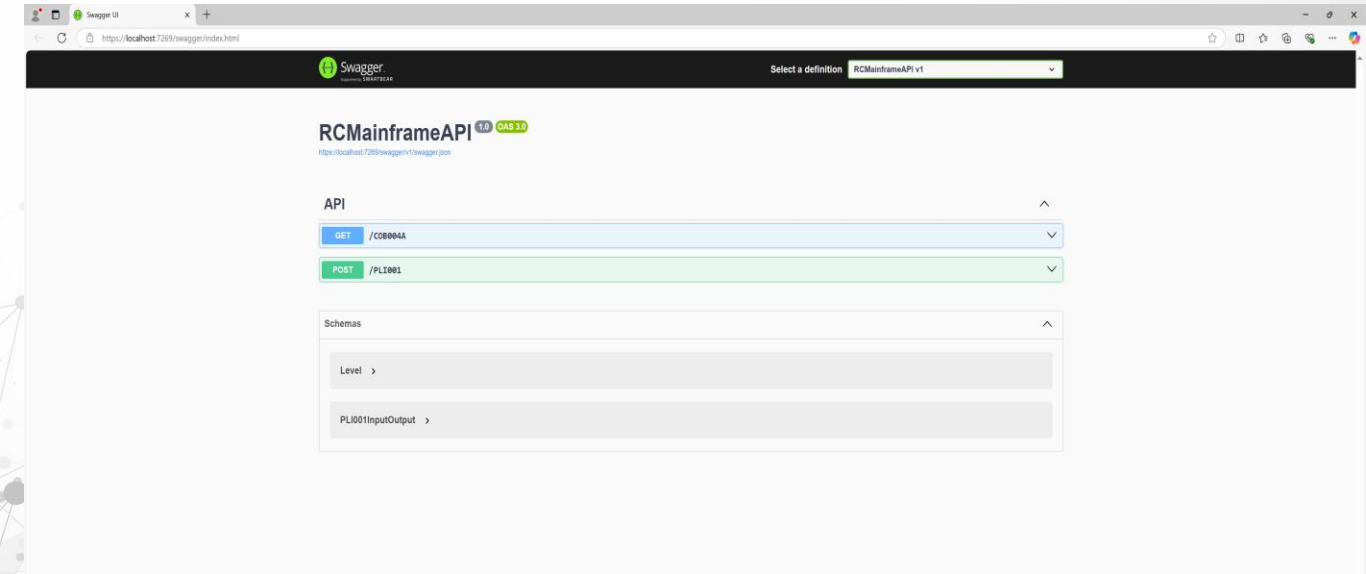
Note:

This script is to be rerun when there is a change in the copybook layout of PLI001 or COB004A modules.

RAINCODE 360 Advanced Showcase

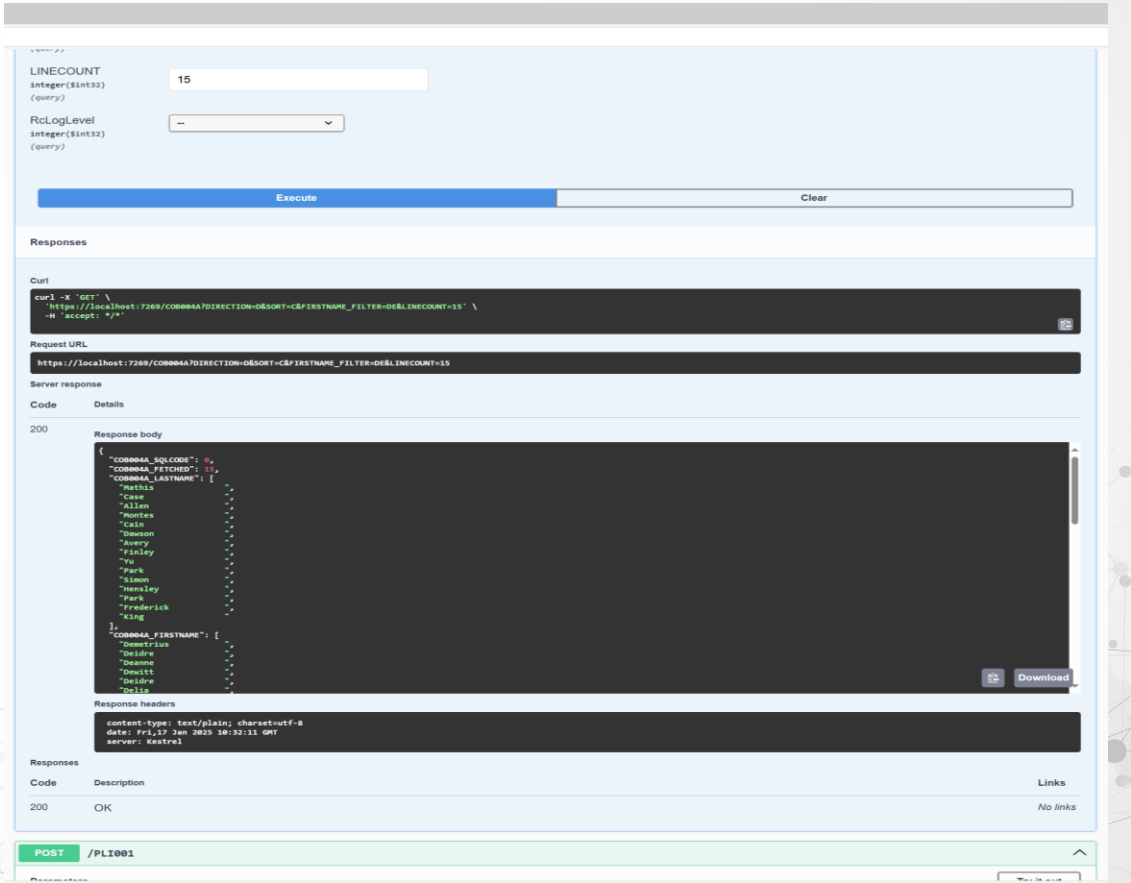
Create REST API from Legacy Program – How to demonstrate

- Step 1: In the solution explorer, right-click the **RCMainframeAPI** project and select *Set as Startup Project*.
- Step 2: Execute the project by pressing *Ctrl+F5* or navigating to the debug menu and selecting *Start Without Debugging*.
- Step 3: A browser with Swagger UI for the GET/COB004A and POST/PLI001 methods will open.



Create REST API from Legacy Program – How to demonstrate

- Click on **Try it out** and enter the following inputs for GET /COB004A method
  - DIRECTION - D
  - SORT - C
  - FIRSTNAME_FILTER - DE
  - LINECOUNT - 15
- Click **Execute.**



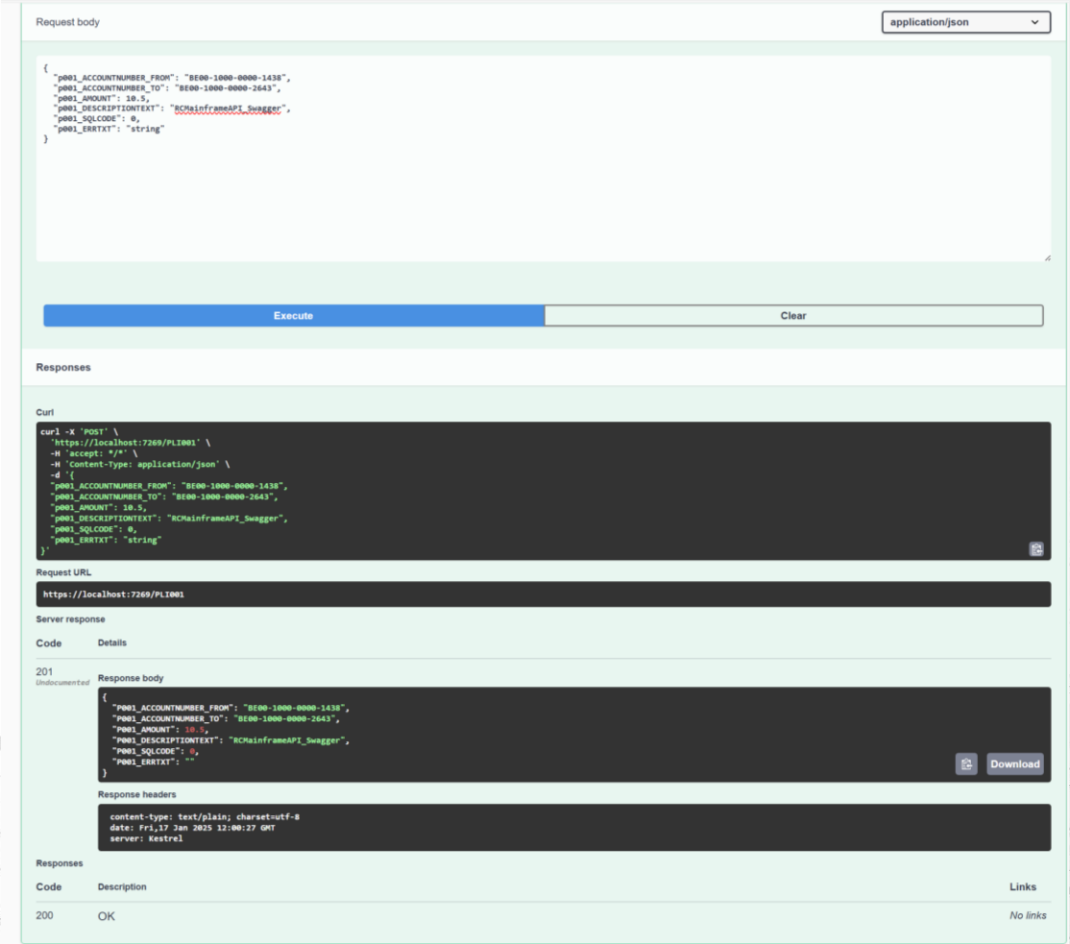
The response will display a list of customer names that starts with **DE.**

RAINCODE 360 Advanced Showcase

Create REST API from Legacy Program – How to demonstrate

- Click on **Try it out,** provide the request body as shown in the screenshot for the PLI001 method, and click **Execute**.

- The server response will display the result with value 201 indicating success.



Create REST API from Legacy Program –Rainbox Generated

Create REST API from Legacy Program – Rainbox Generated

- This demo shows how to generate the REST API from a CICS module with the help of **Rainbox** and invoke it using the Postman API Client.

- **Rainbox** is a tool which is used to generate REST APIs from legacy COBOL and PL/I programs.

- **Steps to generate an API:**
  - Configure rainbox.xml
    - Commarea or Channel mappings
    - Endpoint details (single or multiple)
    - File configuration settings
  - Make sure COBOL and PL/I are compiled successfully, and generated DLLs are available in the assembly path for Rainbox to access.
  - Invoke Rainbox.exe to generate the REST wrapper:
    - *"C:\Program Files\Raincode\Rainbox\rainbox.exe" "-ConfigurationFile=rainbox.xml" "-OutputDir=rest" "-AppNamespace=Rainbox.Demo" "-AssemblyPath=.\output" "-SourcePath=." "-LogLevel=TRACE"*
  - Rainbox looks for the DB connection string (*RAINBOX_DB_CONNECTION_STRING) from the environment variable.*

Create REST API from Legacy Program – Rainbox Generated

- Steps:
  - Run the script **install_postman.ps1** to install Postman, which will be used to call the API (this is a one-time process)
  - Execute the **run.ps1** script, which performs the following steps:
    - Invokes the Raincode's COBOL compiler to compile the COBOL programs COBACCT and COBCUST; generating DLL files in the **output** directory
    - Invokes the Rainbox by providing the configuration file **rainbox.xml** to generate a REST API containing multiple endpoints (/COBCUST and /COBACCT)
      - The generated API project (**Box.csproj**) will be created in the **rest** directory.
    - Builds and launches the REST API, which will be running at Localhost:5000

Create REST API from Legacy Program –Rainbox Generated

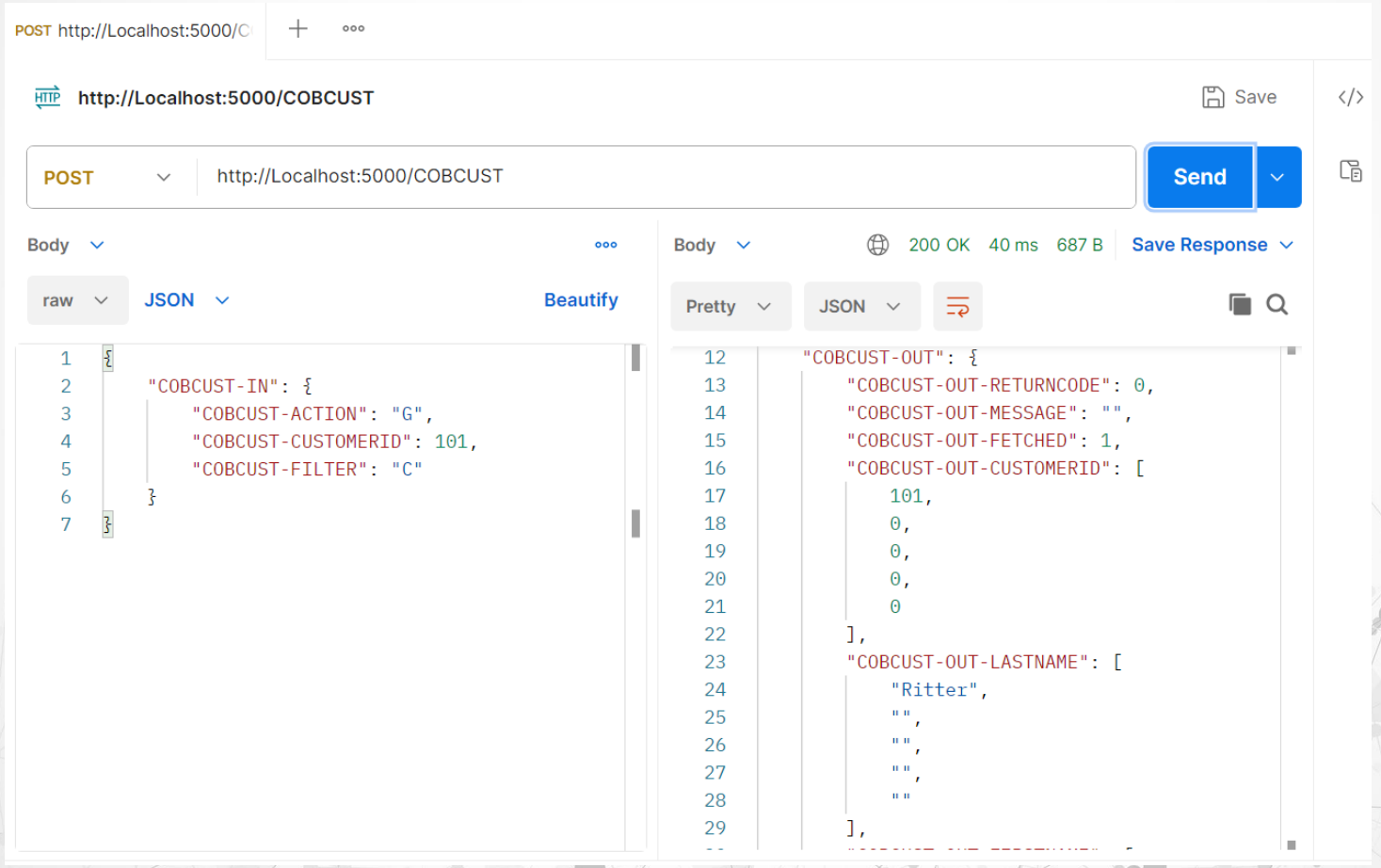
- Trigger a series of requests from Postman as follows
- COBCUST:
  - GET request to the endpoint *http://Localhost:5000/COBCUST/Schema* to understand the schema
  - POST request to the endpoint *http://Localhost:5000/COBCUST* with the following body to read the customer details

```
{
        "COBCUST-IN": {
                "COBCUST-ACTION": "G",
                "COBCUST-CUSTOMERID": 101,
                "COBCUST-FILTER" : "C"
        }
}
```

  - POST request to the endpoint *http://Localhost:5000/COBCUST* with the following body to update the customer's first name

```
{
        "COBCUST-IN": {
                "COBCUST-ACTION": "U",
                "COBCUST-CUSTOMERID": 101,
                "COBCUST-FILTER" : "C",
                "COBCUST-FIRSTNAME": "Emma-Test"
        }
}
```

POST COBCUST (read the customer details)



RAINCODE BEO Advanced Showcase

Create REST API from Legacy Program –Rainbox Generated

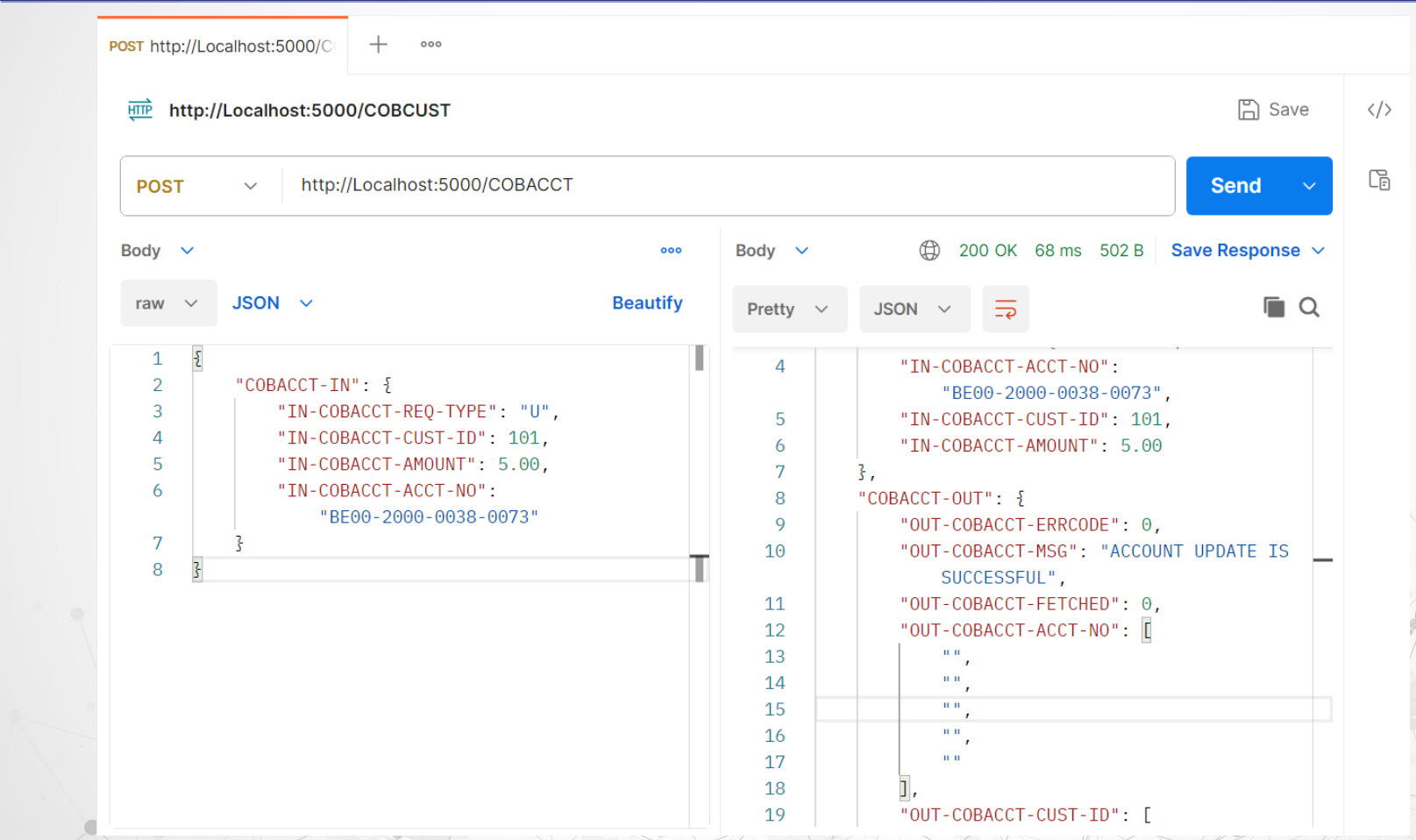
- COBACCT:
  - GET request to the endpoint *http://Localhost:5000/COBACCT/Schema* to understand the schema
  - POST request to the endpoint *http://Localhost:5000/COBACCT* with the following body to read the account details

```
{
        "COBACCT-IN": {
                "IN-COBACCT-REQ-TYPE": "G",
                "IN-COBACCT-CUST-ID": 101
        }
}
```

  - POST request to the endpoint *http://Localhost:5000/COBACCT* with the following body to update the amount of the account

```
{
    "COBACCT-IN": {
            "IN-COBACCT-REQ-TYPE": "U",
            "IN-COBACCT-CUST-ID": 101,
            "IN-COBACCT-AMOUNT": 5.00,
            "IN-COBACCT-ACCT-NO": "BE00-2000-0038-0073"
    }
}
```

POST COBACCT (update the amount of the account)



RAINCODE 360 Advanced Showcase

Online Internal Reader

RAIN CODE 360
Advanced Showcase

Online Internal Reader- Introduction

- This program demonstrates how a Batch can be started from an online CICS application using SPOOL commands

- It highlights the following:
  - Providing a custom QixFactory to the QIX Emulator
  - Utilization of CICS commands, such as CICS SPOOLOPEN, SPOOLWRITE, and SPOOLCLOSE
  - Utilization of BMS maps
  - Execution of a JCL from a CICS application

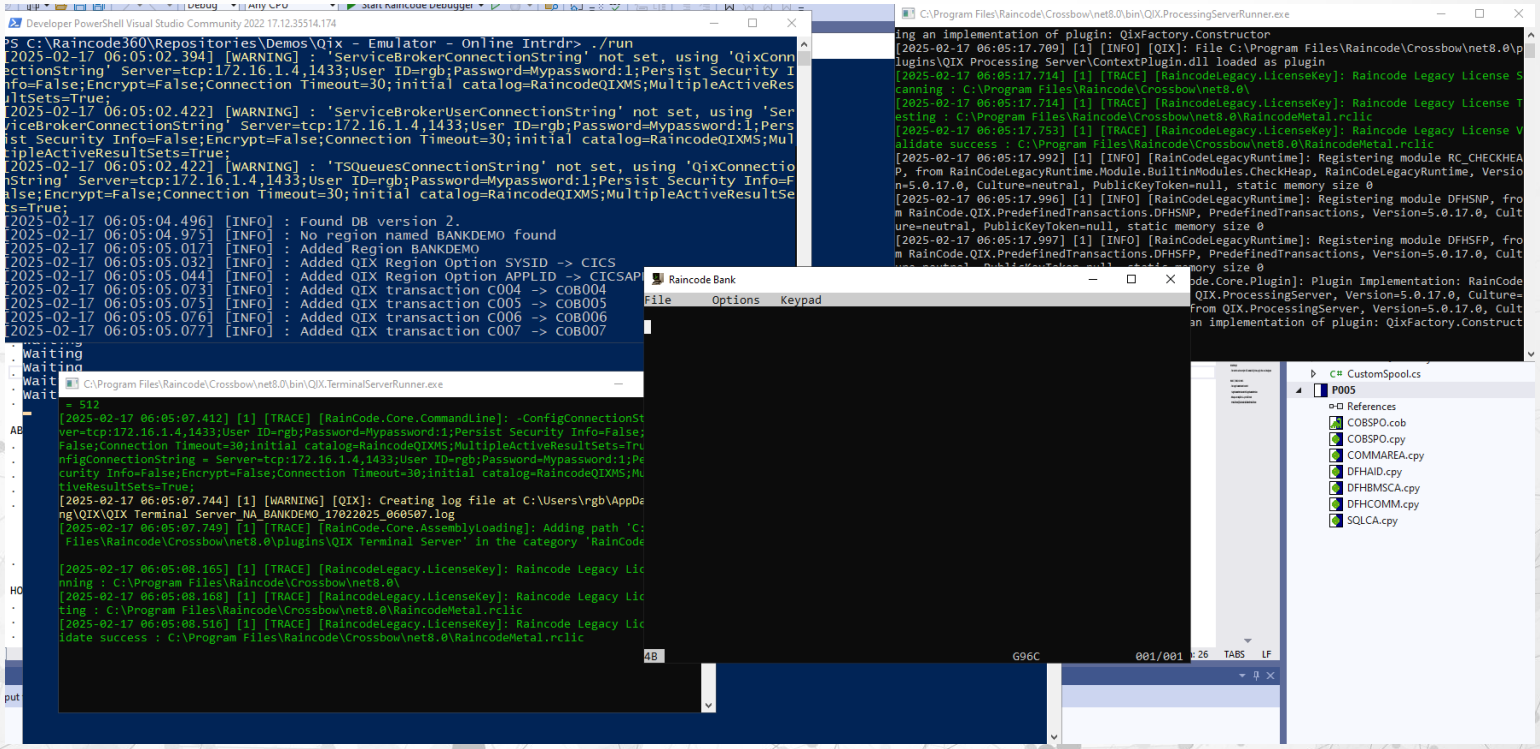
- The CICS writes a file in a designated folder

- A watchdog picks up the file and starts a JCL job

- The generated JCL filters the customers from the database based on birth years within a specified range (start date and end date) and writes them into the dataset (UNLOAD.CUST.SEQ)

Online Internal Reader- Steps to run

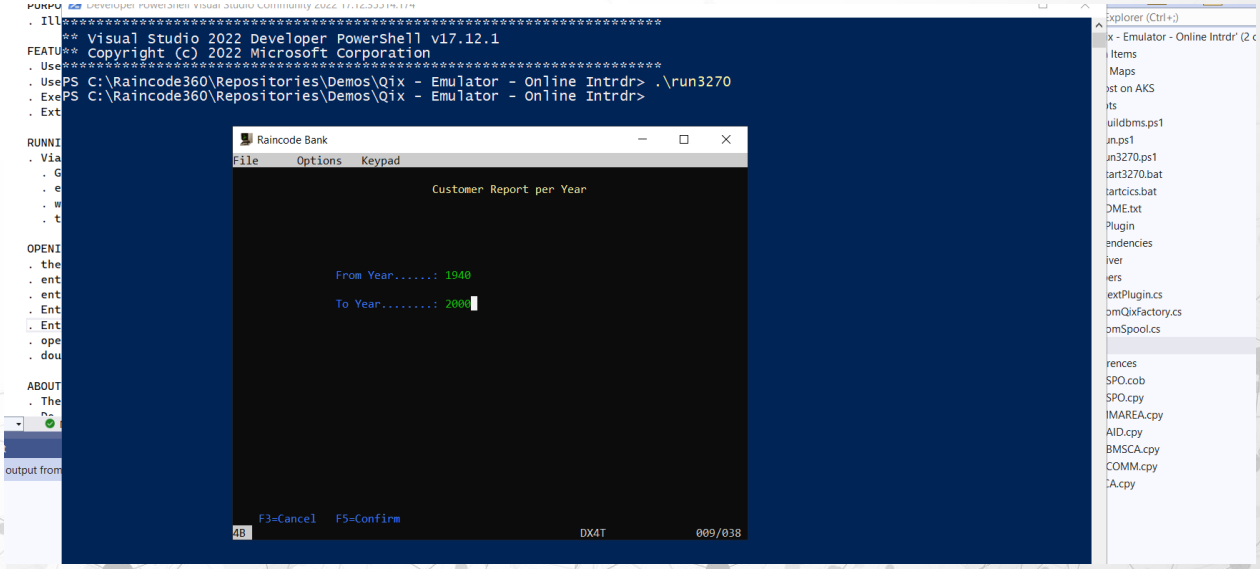
- Open Powershell via Tools->Command Line-> Developer PowerShell
- Execute the *run.ps1* script and wait for the QIX windows and the 3270 emulator to launch. The bank application will start automatically



RAINCODE 360 Advanced Showcase

Online Internal Reader- Opening Additional Terminal Windows

- The QIX server (Processing Server and Terminal Server) supports multiple terminals while running. To open an additional terminal window:
  - In PowerShell, run: .\run3270
  - Enter `**CSPO**` to start the application
  - Enter a start year (greater than 1940)
  - Enter an end year (less than 2000)
  - Open Windows Explorer and navigate to: `"C:\ProgramData\Raincode\Batch\DefaultVolume\UNLOAD\CUST."`
  - Double-click **SEQ.meta** to open the dataset in the Record Editor



RAINCODE 360 Advanced Showcase

Raincode QIX Emulator - CWA

RAINCODE 360
Advanced Showcase

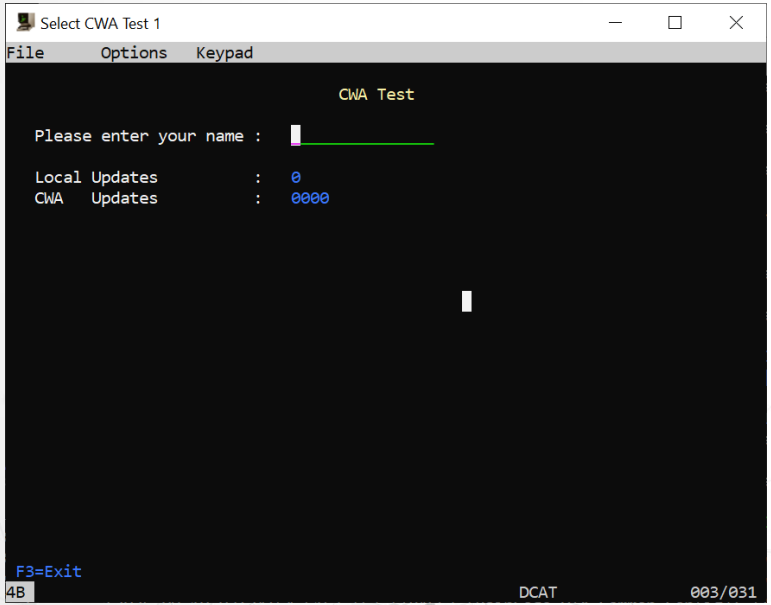
Raincode QIX Emulator – Common Work Area – Purpose and Concept

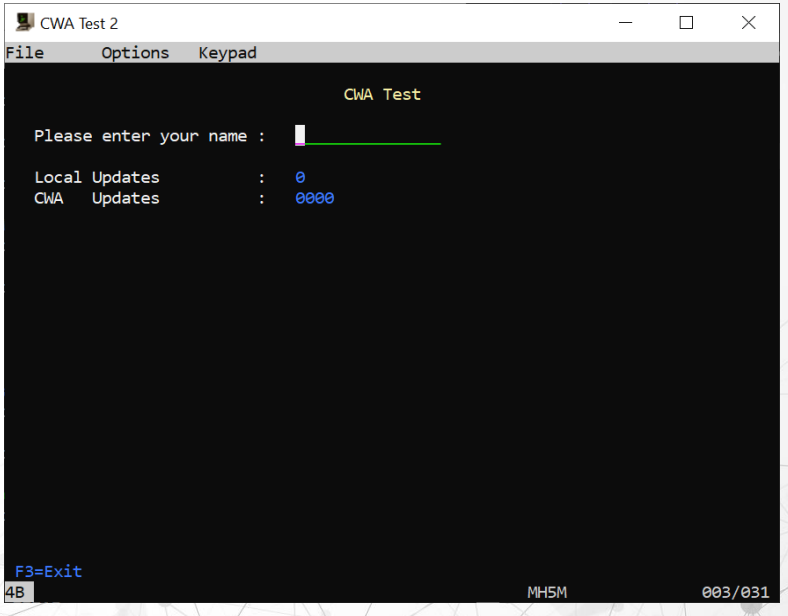
- Raincode's CICS implementation supports shared memory through the Common Work Area (CWA). CICS provides access to a CWA by returning a pointer to it when invoking the ADDRESS CWA command.
- However, in QIX, programs from the same region may run on different processing servers or even on separate machines, which makes direct memory sharing impractical.
- To solve this problem, the contents of the CWA are stored in SQLServer and are loaded into memory at the beginning of the transaction. Any updates done to it during the lifetime of the transaction are written back into the database when the transaction finishes.
- The idea is to run a single transaction that both reads and writes the CWA twice in two separate emulator windows.

To keep things simple, we limit ourselves to a single processing server and a single terminal server running on the same machine.

Raincode QIX Emulator – Running the demo

- Run the *run.ps1* PowerShell script from any directory.
- When the two terminal windows appear, type **CWA1** in each of them.

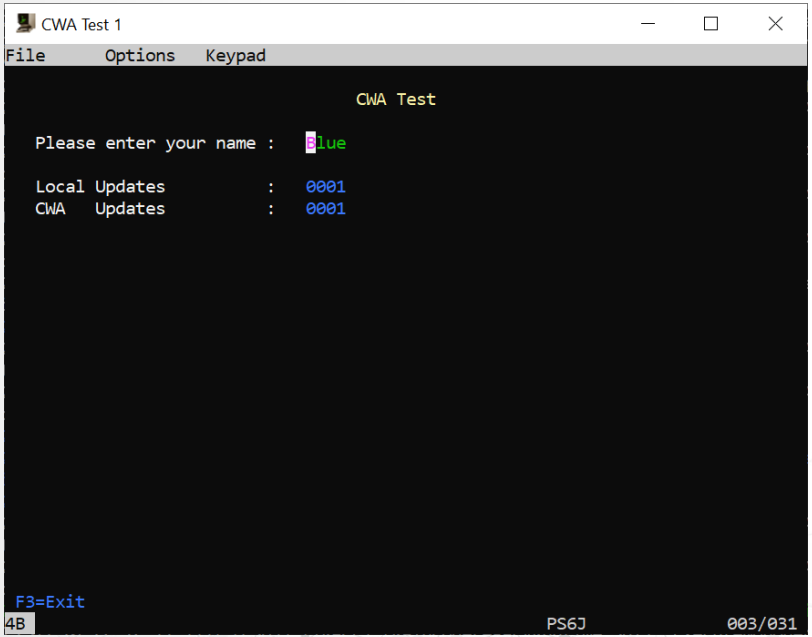


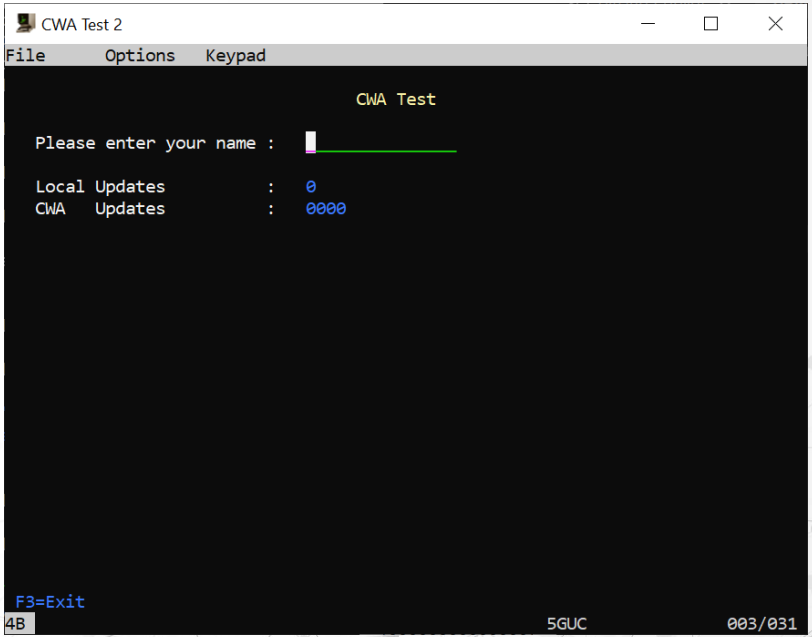


RAINCODE 360 Advanced Showcase

Raincode QIX Emulator – Running the demo

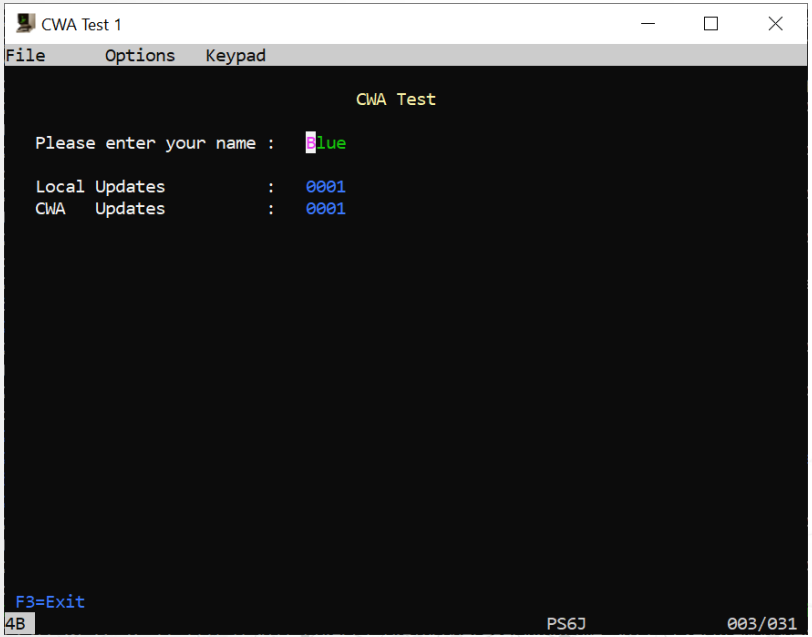
- In the **CWA Test 1** window, type any sequence of characters and press Enter.

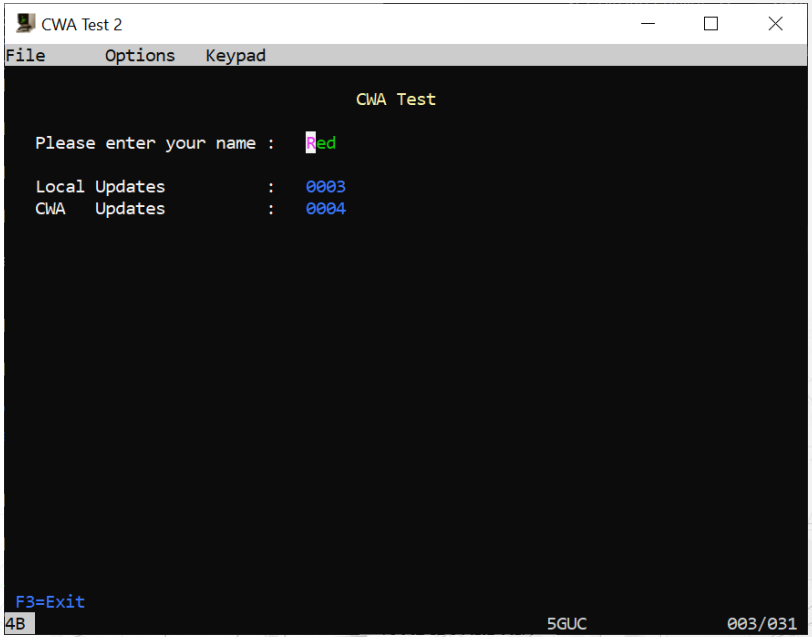




Raincode QIX Emulator – Running the demo

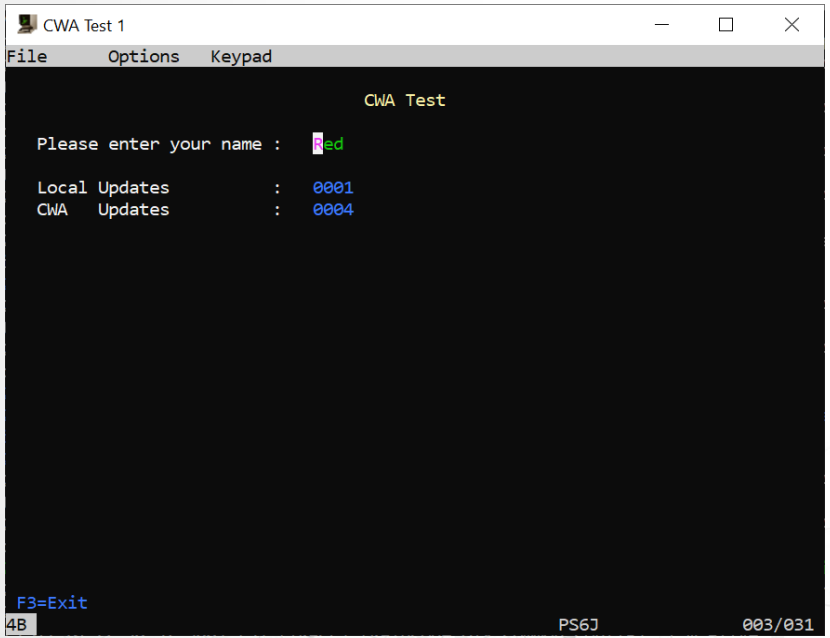
- In the **CWA Test 2** window, press Enter and then type something.

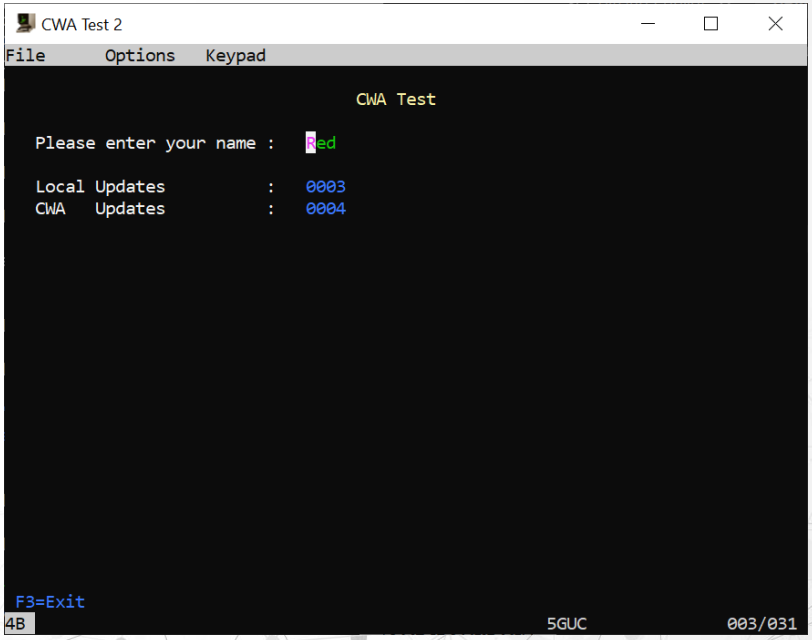




Raincode QIX Emulator – Running the demo

- When you return to the **CWA Test 1** window, the value entered in the **CWA Test 2** window will be displayed



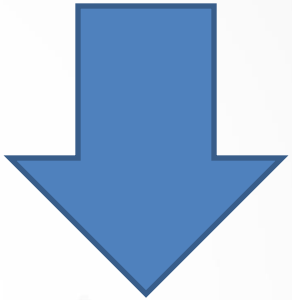


Caching

- Demos:
  - SQL Level Caching
  - Program Caching
  - Standalone SQL Conversion
  - SQL Caching - Singleton Statements

Caching – Introduction

- Keep a copy of slow-moving data closer to the application
- Can be in-process, in-machine, or even on a separate machine
  - Provided it is closer and faster than the original data provider (database)



- A reimplementation of a (simpler) database
- Even if slow-moving, caches can be invalidated
- Deciding that something is slow-moving is application-dependent
- Changes to the application

SQL Level Caching

- Test the performance of SQL Level Caching
- Illustrates the use of static SQL
- Type **.\run** from the powershell command line to execute the demo

- SQL Caching Benchmark

```
Test 1: Static SQL with no caching
Test 2: Static SQL with SQL caching

ELAPSED TIMES
-----------------------------------------
              SQLs       SQLs        Time
Test          Total      Cached      (s)
-----------------------------------------
Test 1        500          0         6.08
Test 2        500        400         0.26
```

The use of static SQL requires that the COBOL program names are unique for the database that they connect to.

When the cache is active, the first 50 SQLs will connect to the database to fill up the cache.

Test 1 and Test 2 execute the same compiled code. Only difference is Test 2 uses the **-plugin** argument of rclrun.

When launching the program from the menu, it will run in the cached mode.

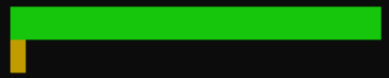
RAINCODE 360 Advanced Showcase

Program Caching

- Test the performance of Program Caching
- Illustrates the use of static SQL

- Caching Benchmark

```
Test 1: Static SQL with no caching
Test 2: Static SQL with program caching

ELAPSED TIMES
-----------------------------------------
             SQLs       SQLs         Time
Test         Total      Cached       (s)
-----------------------------------------
Test 1         500          0         5.90
Test 2         500        400         0.27
```



The use of static SQL requires that the COBOL program names are unique for the database that they connect to.

When the cache is active, the first 50 SQLs will connect to the database to fill up the cache.

Test 1 and Test 2 execute the same compiled code. Only difference is Test 2 uses the **-plugin** argument of rclrun.

When launching the program from the menu, it will run in the cached mode.

SQL Caching – Singleton Statements – Introduction

- The following pattern:

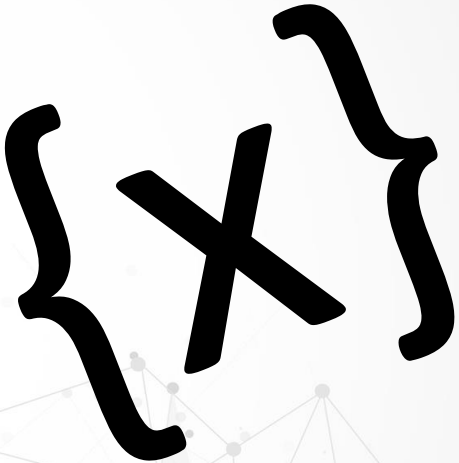
```
SELECT F1, F2, F3
 FROM T
 INTO :HT1, :HT2, :HT3
 WHERE K1=:HK1 AND
       K2=:HK2 AND
       K3=:HK3
```

- VSAM-like use of SQL
- Probably cacheable
  - Chatty (one record at most)
  - Read (as opposed to write)
  - Simple criterion (typical of a primary key)

- Recognizing that a SQL statement that is cacheable is not trivial
  - SQL is way too versatile for comfort
  - Even more so if the analysis must happen at runtime
- But our compilers perform this at compile-time
- At runtime, it is merely an attribute to consult
  - Together with the table, source fields, target fields

How to implement a cache for singletons?

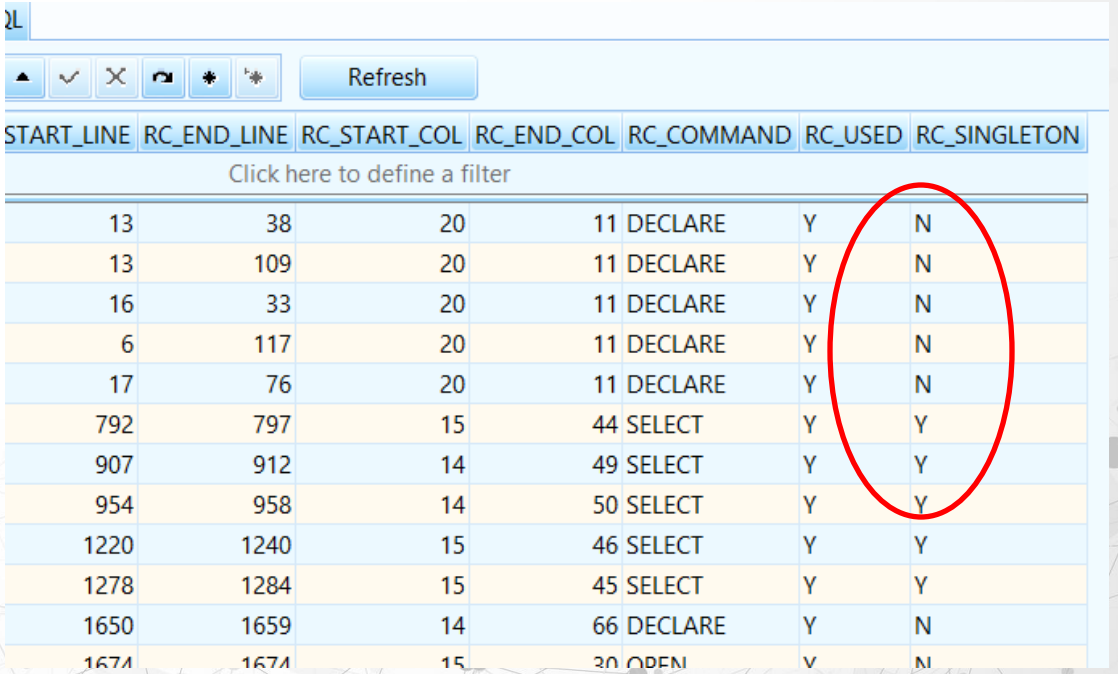
- When executing a SQL statement:
  - Check for the singleton attribute
  - Ensure that the table is cacheable (slow-moving)
  - Ensure that the Kn tuple defines a primary key
- If all these conditions are fulfilled
  - Redirect the query to the cache
- In all other cases, revert to default behaviour
  - Send the actual statement to the database



- We do not supply an actual cache implementation!
  - Many different design decisions to make

Benefits

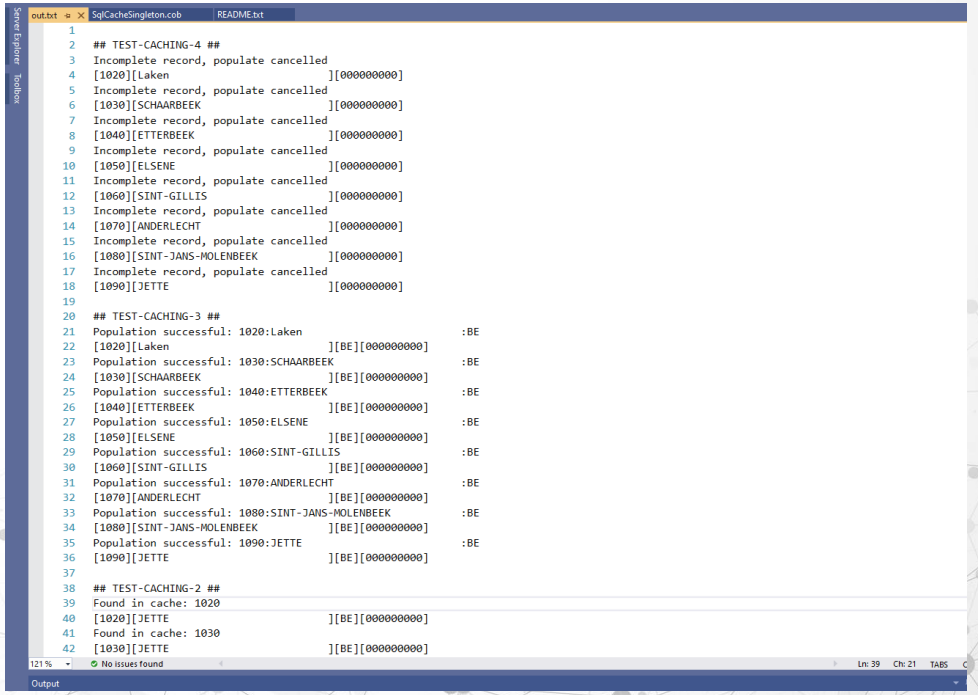
- No change to the application code
  - The version running on the mainframe remains unaltered
- Database agnostic
  - SQL Server or DB2 (through HIS)
- Scalability
  - One must visit tables, not statements
- Information present in the repository



RAINCODE 360 Advanced Showcase

SQL Caching – Singleton Statements

- Demonstrate SQL Caching via a plugin
- Demonstrate that statements that do not retrieve all the columns of the table should not populate the cache
- Illustrates the use of static SQL
- After the script is executed, it will write the output to a file named **out.txt**



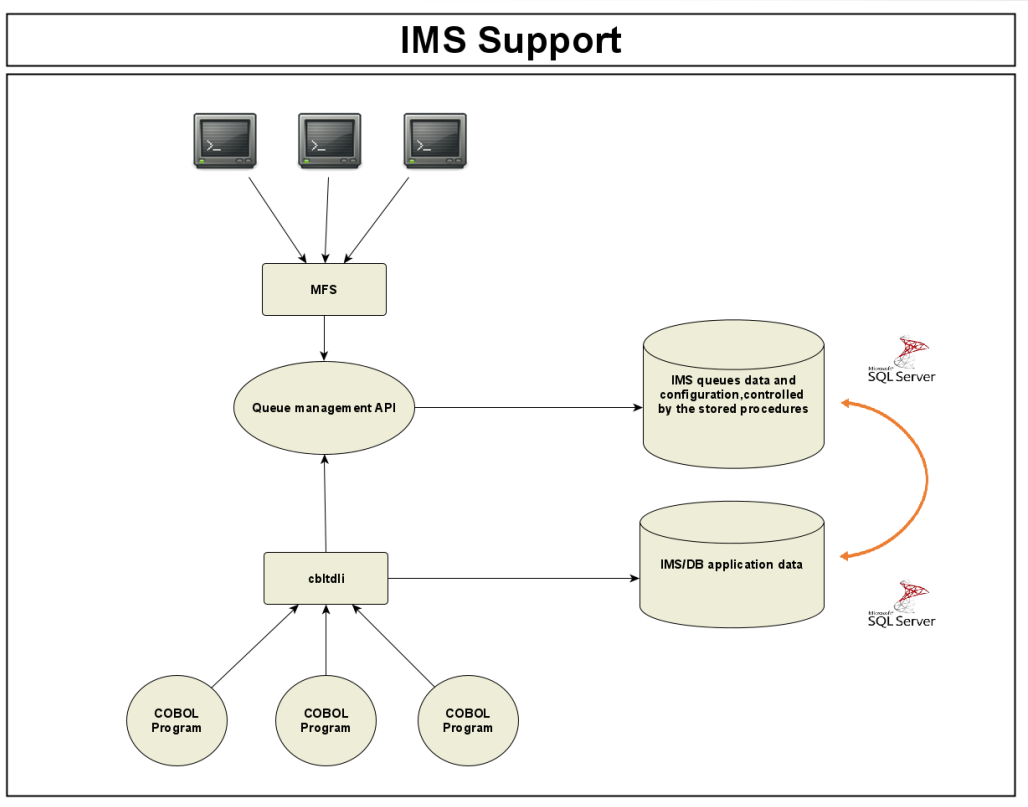
RAINCODE BEO Advanced Showcase

IMSql

IMSql – Introduction

- IMSql rehosts IMS/DB and IMS/TM systems on .NET and SQL Server. Mainframe applications compiled for .NET can interact with IMSql in the same manner as they would interact with IMS.

- IMSql is non-transformational, which means it keeps the sources (COBOL, PL/I) as-is with the IMS-specific CBLTDLI, PLITDLI, and EXEC DLI calls unaltered. This way, new sources are not introduced, which plays a crucial role in program maintainability.



IMSql – Introduction

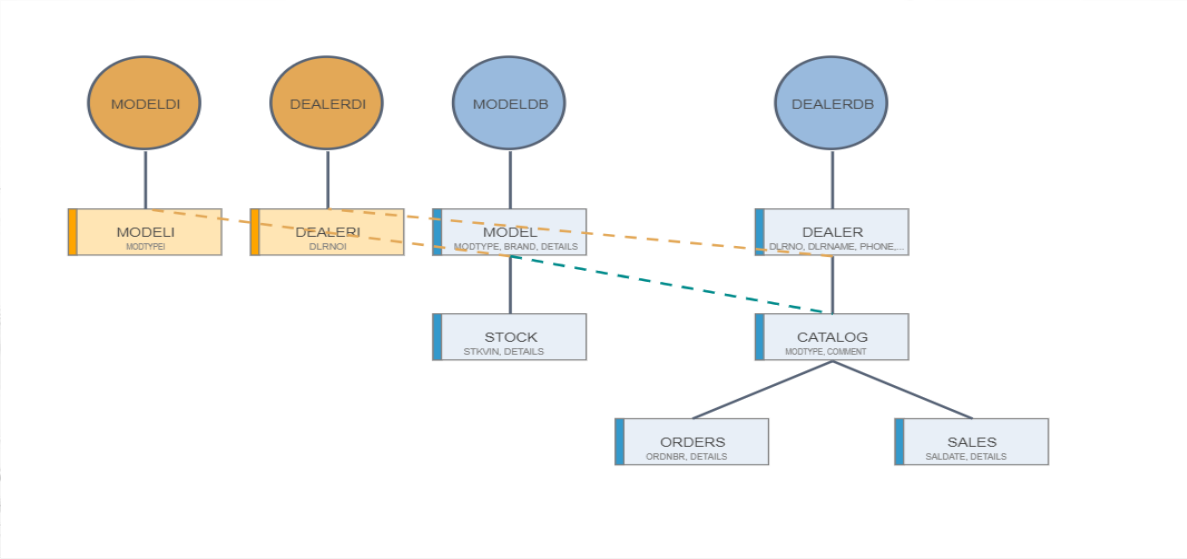
- IMS/DB
  - Data Migration
    - The new DB is created using the DBD's
    - A table is created for each physical segment
    - Data is not transformed; IMS segments are stored as raw data
- Program compilation
  - Entity Framework *DbContext* is generated for each DBD
  - Untransformed programs (COBOL and PL/I) are compiled as usual
  - The rcpsb utility translates the PSB into XML files that are loaded at runtime
- IMS/TM
  - The IMS/TM is translated using SQL Server Queues and Messages
  - The *rcmfs* utility translates the MFS files into XML files that are loaded at runtime

IMSql- Demos

- Introduction
- Batch
- Online
- Checkpoint/Restart

IMSql – Demo – Introduction

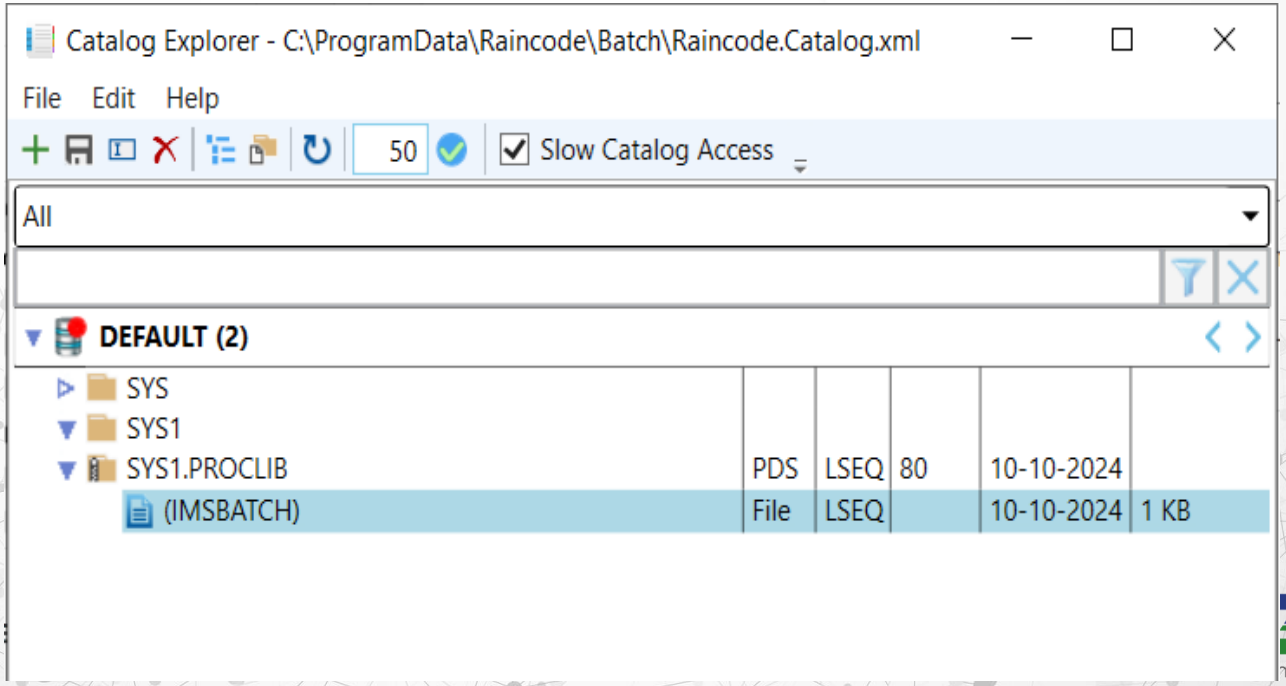
- For the demo we use two DBDs (MODELDB and DEALERDB)
  - In the graph below, DBDs are presented by circle and segments by rectangle. Physical links by plain lines, logical links by dotted lines
- Each DBD has a primary index, respectively MODELDI and DEALERDI
- DEALERDB.CATALOG is a logical child of MODELDB.MODEL



IMSql – Demo – Batch

- Step 1: Open the solution from the folder IMSql\DealerBatch
- Step 2: Make 'ADDPROC' as the startup project by right clicking on the project and selecting 'Set as Startup Project'
- Step 3: Click on 'Start Raincode Debugger', this creates the PROC IMSBATCH in the catalog, which is one time task

Expected Result:



IMSql – Demo – Batch

- Step 1: Open the solution from the folder IMSql\DealerBatch
- Step 2: Make 'BESTSALE' as the startup project by right clicking on the project and selecting 'Set as Startup Project'
- Step 3: Click on 'Start Raincode Debugger', this executes BESTSALE.JCL and the program BESTSALE.cbl searches for the DEALER with the biggest number of SALES segments

- **Expected Result:**

Job must execute successfully and

SYSOUT Should have this information

```
NBR RECORD   : 00041
BEST DEALER : DEAL00030  / 00011
END TIME 2024/10/10 14:28:29:83
final PCB STATE:
PCB STATE :
    DB-LEVEL   :  0
    DB-KFBA    : 00000 ->
    DB-SEGMENT:
    DB-STAT    : GB
    DB-PROC    : GIRD
```

IMSql – Demo – Batch

- Step 1: Open the solution from the folder IMSql\DealerBatch
- Step 2: Make 'EXTRACT-STAT' as the startup project by right clicking on the project and selecting 'Set as Startup Project'
- Step 3: Click on 'Start Raincode Debugger', this executes EXTRACT-STAT.JCL and the program DLRSTAT.cbl counts the number of segments of each type for the first 101 segments of the DBD

- **Expected Result:**

Job must execute successfully and

SYSOUT Should have this information

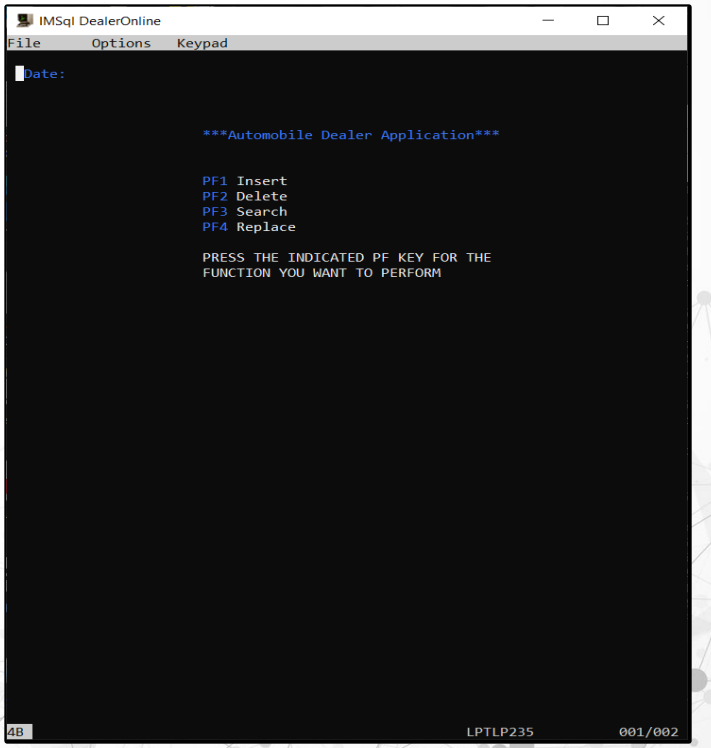
```
NBR RECORD   : 00101
NBR DEALER   : 00022
NBR CATALOG  : 00042
NBR ORDERS   : 00018
NBR SALES    : 00019
END TIME 2024/10/10 14:17:01:77
final PCB STATE:
PCB STATE :
    DB-LEVEL   :  3
    DB-KFBA    : 00050 -> DEAL00010 CARV00010
    DB-SEGMENT: ORDERS
    DB-STAT    :
    DB-PROC    : GIRD
```

IMSql – Demo – Online

- The IMS/TM is translated using SQL Server Queues and Messages
- This application search/insert DEALER, CATALOG segments
- COBOL/PLI programs using call CBLTDLI, PLITDLI and EXEC DLI

IMSql – Demo – Online

- Step 1: Double-click on "**Start IMSql 3270 demo**" shortcut placed on the desktop
  - Starts two servers: terminal server and processing server
  - Starts WC3270: 3270 Emulator
- Step 2: Enter username and password (demo/demo)
- Step 3: Enter the command "/FOR OMMENU"
- Step 4: Use function keys to navigate to the different screens



The transactions details can also be seen through [Raincode console]

RAINCODE 360 Advanced Showcase

IMSql – Demo – Online

Possible values for the online screens

- Insert (PF1)
  - Insert a new Dealer (PF2)
    - Insert new values
  - Insert a new Model (PF3)
    - Insert new values
- Delete (PF2)
  - Delete a dealer (PF2)
    - DDDD00005
  - Delete a model (PF3)
    - BMWV01323
- Search (PF3)
  - Search for a dealer (PF2)
    - ID Number =DDDD00001
  - Search for a model (PF3)
    - Model Type = CARV00001
  - Display catalogs (PF4)
    - ID Number = DDDD00050
- Replace (PF4)
  - Update a model (PF4)
    - Type = CARV00004

      Brand = <new data>

      Details = <new data>

IMSql – Online (Raincode Console)

Terminals

Name	User	Host	Conversation	ConnectionTime
LPTLP207	NOBODY	raincode360-vm	b1e732a7-849f-ec11-b402-002248584216	09/03/2022 08:41:04
LPTLP229	NOBODY	raincode360-vm	7fb88d4a-cd9e-ec11-b402-002248584216	08/03/2022 10:48:30
LPTLP235	NOBODY	raincode360-vm	2dbd91c4-5ba3-ec11-b404-002248584216	14/03/2022 05:58:28
LPTLP365	NOBODY	raincode360-vm	9f550257-969b-ec11-b401-002248584216	04/03/2022 08:37:35
LPTLP456	NOBODY	raincode360-vm	ada99bf7-959f-ec11-b402-002248584216	09/03/2022 10:45:00

LPTLP235 is the same terminal (at the bottom of the 3270 emulator) which starts when you run the IMSql online demo.

RAINCODE 360 Advanced Showcase

IMSql – Online (Raincode Console)

Conversations

conversation_handle	local_service	far_service	state_desc	
bc16cf0d-969f-ec11-b402-002248584216	imsql://DEALER/TRANS/dinsert	https://imsql.raincode.com/IMS/D... vm/TerminalService	CONVERSING	ENDDIALOG
b916cf0d-969f-ec11-b402-002248584216	https://imsql.raincode.com/IMS/D... vm/TerminalService	imsql://DEALER/TRANS/dinsert	CONVERSING	ENDDIALOG
ef75ed5b-c89e-ec11-b402-002248584216	https://imsql.raincode.com/IMS/D...	https://imsql.raincode.com/IMS/D... vm/TerminalService	DISCONNECTED_INBOUND	ENDDIALOG
a2550257-969b-ec11-b401-002248584216	https://imsql.raincode.com/IMS/D...	https://imsql.raincode.com/IMS/D... vm/TerminalService	DISCONNECTED_INBOUND	ENDDIALOG
8f1793f3-689f-ec11-b402-002248584216	imsql://DEALER/TRANS/dinsert	https://imsql.raincode.com/IMS/D... vm/TerminalService	CONVERSING	ENDDIALOG
8c1793f3-689f-ec11-b402-002248584216	https://imsql.raincode.com/IMS/D... vm/TerminalService	imsql://DEALER/TRANS/dinsert	CONVERSING	ENDDIALOG

Details of the conversation: When you choose to insert a new dealer.

IMSql – Online (Raincode Console)

Message Queues

Queues

QueryNotificationErrorsQueue (0)
EventNotificationErrorsQueue (0)
ServiceBrokerQueue (0)
TerminalServer (0)
ProcessingServer (0)
IMS_PROC_DEALER (1)
IMS_DEALER_raincode360-vm (2)
IMS_DEALER (0)

When you click on IMS_DEALER_raincode360-vm (2), it will take you to the details of the pending messages

Messages

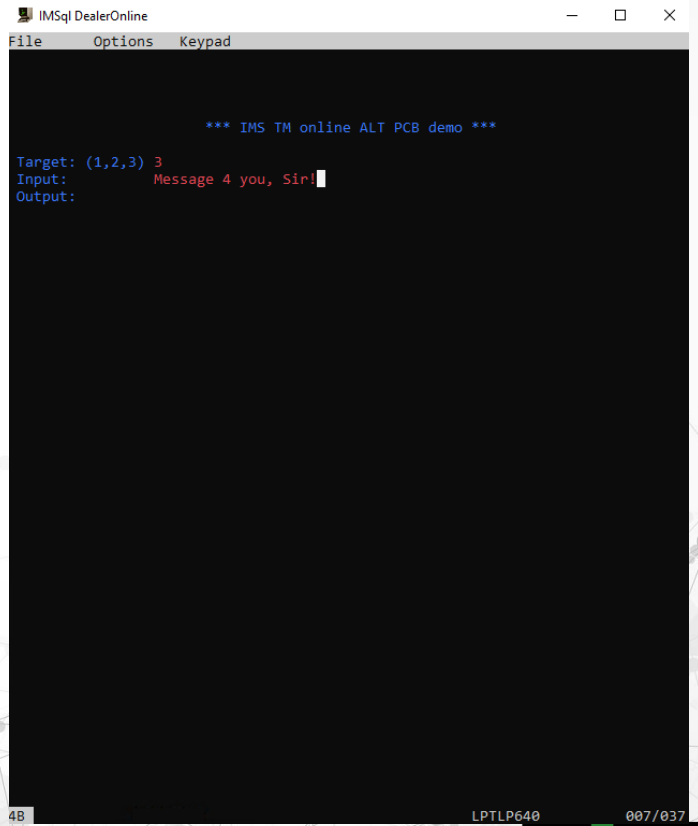
message_enqueue_time	service_name	service_contract_name	message_type_name
2022-03-08T08:46:46.853	https://imsql.raincode.com/IMS/DEALER/raincode360-vm/TerminalService	ProcessingServiceContract	ErrorMessage
2022-03-14T05:26:45.967	https://imsql.raincode.com/IMS/DEALER/raincode360-vm/TerminalService	ProcessingServiceContract	OutputMessage

IMSql – Demo – Online – Alternate PCB

- The alternate PCB can have as output message destination
  - transaction code names (`NAME=`)
  - logical terminal names (`LTERM=`)
- In both cases, IMSql uses a program as a destination
  - In COBOL or PLI
  - In C#
- All programs use call CBLTDLI, PLITDLI and EXEC DLI as normal

IMSql – Demo – Online – Alternate PCB

- Step 1: Double-click on the "**Start IMSql 3270 demo**" shortcut placed on the desktop
  - Starts two servers: terminal server and processing server
  - Starts WC3270: 3270 Emulator
- Step 2: Enter username and password (demo/demo)
- Step 3: Enter the command "/FOR RECMIN"
- Step 4: Enter a target destination number and some Input text.
- Step 5: The mirrored text is returned as Output.



IMSql – Demo – Online – Alternate PCB

Target destinations are:

1. A COBOL program logger.cbl registered as a transaction
   - Calls `CBLTDLI` to get the message and does a `DISPLAY` of the contents
   - Requires a PSB (LOGGER00)
2. A COBOL program termlogger.cbl registered as a terminal
   - Fundamentally, it is the same code as the logger.cbl
   - Does **not** require a PSB since it is a terminal
- A C# program logger.cs registered as a terminal
   - Construct a call to `CBLTDLI` as if it were a COBOL program
   - Does **not** require a PSB since it is a terminal

IMSql – Demo – Checkpoint/Restart

- To demonstrate the checkpoint/restart, we use a program that
  - Reads command from a GSAM file (RCD001GI)
  - Writes a log into a GSAM file (RCD001GO)
  - Set a checkpoint after every 2 commands. Checkpoint are named “D0100002”, “D0100004”, …
  - The commands can be
    - ADD = insert entry in IMS DB
    - DELETE = delete entry from IMS DB
    - UPDATE = update entry from IMS DB
    - DISPLAY = display entry
    - TADD = same as ADD, but write to console
    - ABEND = If in restart mode, do nothing else - execute a “divided by 0” to abort the program

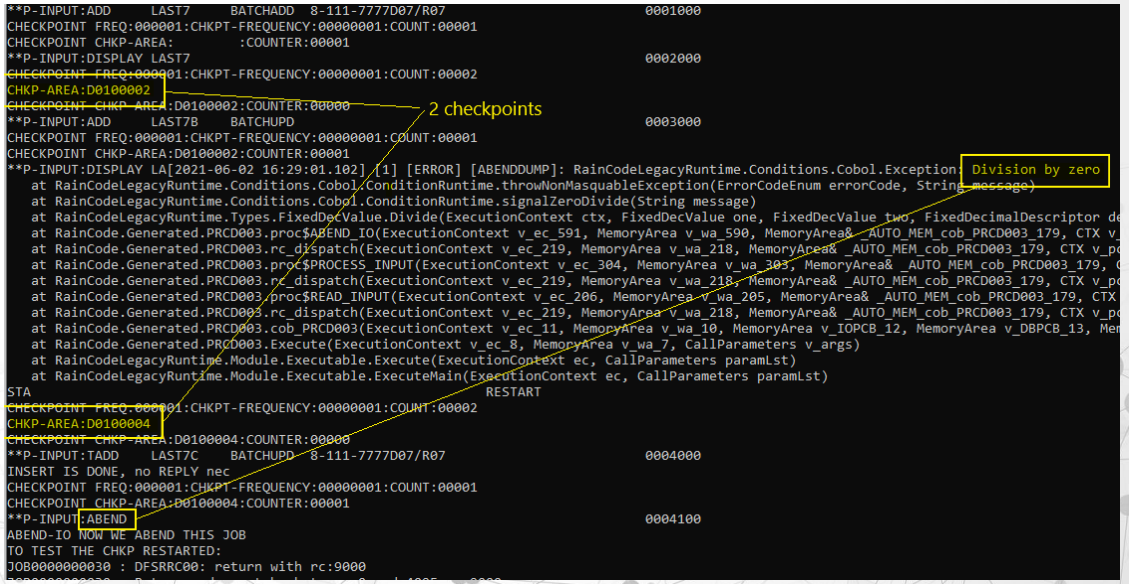
IMSql – Demo – Checkpoint/Restart

- The command file used

```
ADD     LAST7     BATCHADD  8-111-7777D07/R07                    0001000
DISPLAY LAST7                                                    0002000
checkpoint “D0100002” here
ADD     LAST7B    BATCHUPD                                       0003000
DISPLAY LASTA                                                    RESTART
checkpoint “D0100004” here
TADD    LAST7C    BATCHUPD  8-111-7777D07/R07                    0004000
ABEND                                                            0004100
checkpoint “D0100006” here
DISPLAY LAST7B                                                   0005000
DELETE  LAST7                                                    0006000
checkpoint “D0100008” here
DISPLAY LAST7                                                    0007000
TADD    LAST8     BATCHADD  8-111-8888D08/R08                    0008000
checkpoint “D0100010” here
DELETE  LAST8                                                    0009000
DISPLAY LAST8                                                    0010000
```

IMSql – Demo – Checkpoint/Restart

- Step 1: Open powershell and navigate to directory "C:\Raincode360\Repositories\Demos\IMSql\CheckpointRestart"
- Step 2: Type command **`make demo`** to run the first step of the demo:
  - Execute "RCD001.EPRCD003.JCL"
  - Run in non-restart mode
    → ABEND = div/0
  - The 6th command is ABEND
  - Two checkpoints
  - "`TADD LAST7C`" is between the last checkpoint and the ABEND. So, it's rollback (not committed to the DB)
  - Return RC = 9000 + div/0



Advanced Showcase

IMSql – Demo – Checkpoint/Restart

- Step 2: Type the command **`make restart`** to run the second step of the demo:
  - Execute “RCD001.EPRCDR03.JCL”
  - Run in Restart mode → ABEND = no op

```
//*HIDAMUPD EXEC PROC=IMSBATCH,MBR=DFSIVA64,PSB=DFSIVP64,IMSID=IVP1,
//*         SOUT='*',TIME=(60),CKPTID=LAST
```

  - (Re)start after the last checkpoint
  - Succeed with RC = 0

```
CHKPT-XRST-AREA:D0100004    :RESTARTED:Y
AFTER XRST STATUS:
IOPCB: RCPC01         L   L            Jean.Hen
FOR   READ-CHKPT-FREQ
FREQ:000001:CHKPT-FREQUENCY:00000001
AFTER READ-CHKPT-FREQ:000001
**P-INPUT:TADD    LAST7C    BATCHUPD  8-111-7777D07/R07                    0004000
INSERT IS DONE, no REPLY nec
CHECKPOINT FREQ:000001:CHKPT-FREQUENCY:00000001:COUNT:00001
CHECKPOINT CHKP-AREA:        :COUNTER:00001
**P-INPUT:ABEND                                                           0004100
ABEND-IO NOW WE ABEND THIS JOB
TO TEST THE CHKP RESTARTED:Y
NO ABEND BECAUSE WE ARE IN RESTART
CHECKPOINT FREQ:000001:CHKPT-FREQUENCY:00000001:COUNT:00002
CHKP-AREA:D0100006
```

```
**P-INPUT:DISPLAY LAST8                                                   0010000
CHECKPOINT FREQ:000001:CHKPT-FREQUENCY:00000001:COUNT:00002
CHKP-AREA:D0100012
CHECKPOINT CHKP-AREA:D0100012:COUNTER:00000
JOB00000031 : DFSRRC00: return with rc:0
```

RAINCODE 360 Advanced Showcase

IMSql – Demo – Checkpoint/Restart

RCD001GO after first run (ABEND)

```
1  *-------------------------------------------------------------------------*
   |                                                                         |
   |          ************************************************               |
   |          * TEST FOR CHECKPOINT RESTART                                  |
   |          ************************************************               |
   |                                TRANSACTION TYPE : BMP/DLI (HIDAM DB)    |
   |                                DATE       :     06/11/21 09:52:36       |
   |  PROCESS  CODE  (*1) :   DISPLAY                                        |
   |                                                (*1) PROCESS CODE        |
   |  LAST  NAME          :   LASTA                        ADD               |
   |                                                       DELETE            |
   |  FIRST NAME          :                                UPDATE            |
   |                                                       DISPLAY           |
   |  EXTENSION  NUMBER   :                                TADD              |
   |  INTERNAL  ZIP CODE  :                                                  |
   |                                                                         |
   |                                                                         |
   |  SPECIFIED PERSON WAS NOT FOUND                    SEGMENT# :  0004     |
0  *-------------------------------------------------------------------------*
1  *-------------------------------------------------------------------------*
   |                                                                         |
   |          ************************************************               |
   |          * TEST FOR CHECKPOINT RESTART                                  |
   |          ************************************************               |
   |                                TRANSACTION TYPE : BMP/DLI (HIDAM DB)    |
   |                                DATE       :     06/11/21 09:52:36       |
   |  PROCESS  CODE  (*1) :   TADD                                           |
   |                                                (*1) PROCESS CODE        |
   |  LAST  NAME          :   LAST7C                       ADD               |
   |                                                       DELETE            |
   |  FIRST NAME          :   BATCHUPD                     UPDATE            |
   |                                                       DISPLAY           |
   |  EXTENSION  NUMBER   :   8-111-7777                   TADD              |
   |  INTERNAL  ZIP CODE  :   D07/R07                                        |
   |                                                                         |
   |                                                                         |
   |  ENTRY WAS ADDED                                   SEGMENT# :  0005     |
0  *-------------------------------------------------------------------------*
```

Last checkpoint (D0100004)

First run

Second run

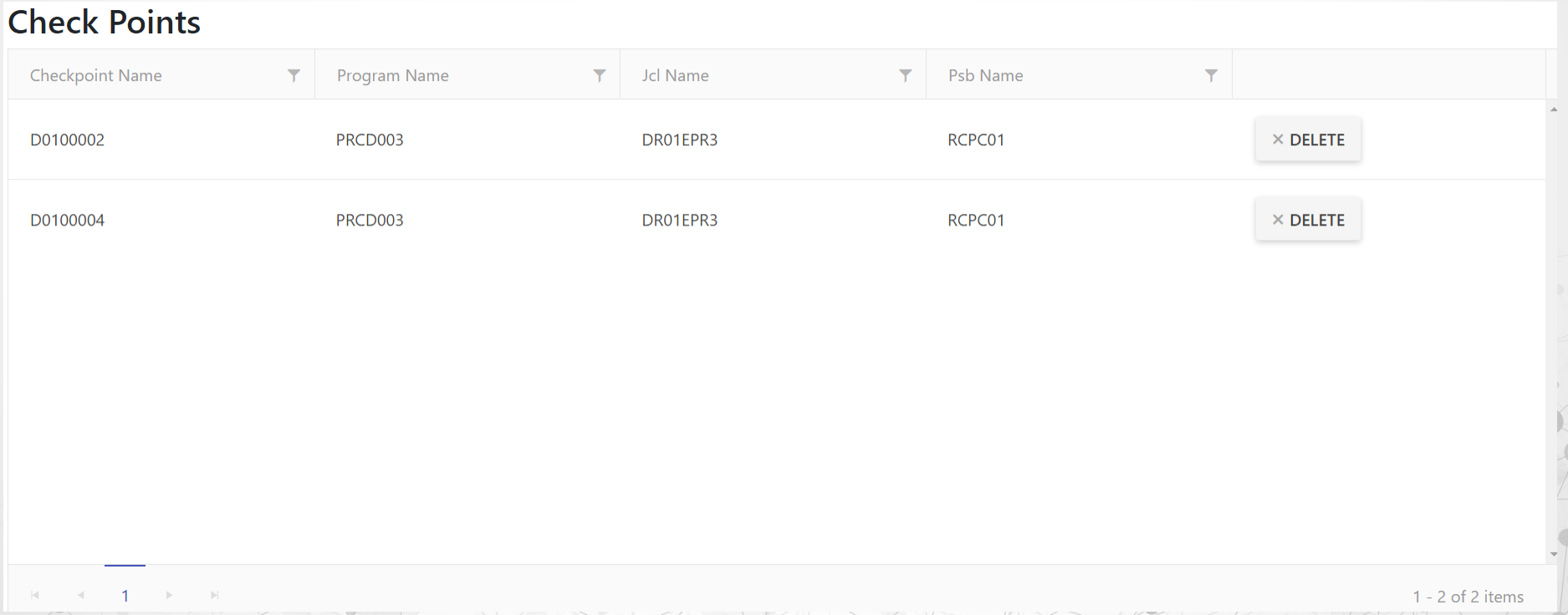
RCD001GO after second run (SUCCEED)

```
1  *-------------------------------------------------------------------------*
   |                                                                         |
   |          ************************************************               |
   |          * TEST FOR CHECKPOINT RESTART                                  |
   |          ************************************************               |
   |                                TRANSACTION TYPE : BMP/DLI (HIDAM DB)    |
   |                                DATE       :     06/11/21 09:52:36       |
   |  PROCESS  CODE  (*1) :   DISPLAY                                        |
   |                                                (*1) PROCESS CODE        |
   |  LAST  NAME          :   LASTA                        ADD               |
   |                                                       DELETE            |
   |  FIRST NAME          :                                UPDATE            |
   |                                                       DISPLAY           |
   |  EXTENSION  NUMBER   :                                TADD              |
   |  INTERNAL  ZIP CODE  :                                                  |
   |                                                                         |
   |                                                                         |
   |  SPECIFIED PERSON WAS NOT FOUND                    SEGMENT# :  0004     |
0  *-------------------------------------------------------------------------*
1  *-------------------------------------------------------------------------*
   |                                                                         |
   |          ************************************************               |
   |          * TEST FOR CHECKPOINT RESTART                                  |
   |          ************************************************               |
   |                                TRANSACTION TYPE : BMP/DLI (HIDAM DB)    |
   |                                DATE       :     06/11/21 09:54:52       |
   |  PROCESS  CODE  (*1) :   TADD                                           |
   |                                                (*1) PROCESS CODE        |
   |  LAST  NAME          :   LAST7C                       ADD               |
   |                                                       DELETE            |
   |  FIRST NAME          :   BATCHUPD                     UPDATE            |
   |                                                       DISPLAY           |
   |  EXTENSION  NUMBER   :   8-111-7777                   TADD              |
   |  INTERNAL  ZIP CODE  :   D07/R07                                        |
   |                                                                         |
   |                                                                         |
   |  ENTRY WAS ADDED                                   SEGMENT# :  0001     |
0  *-------------------------------------------------------------------------*
1  *-------------------------------------------------------------------------*
   |                                                                         |
   |          ************************************************               |
```

RAINCODE Advanced Showcase

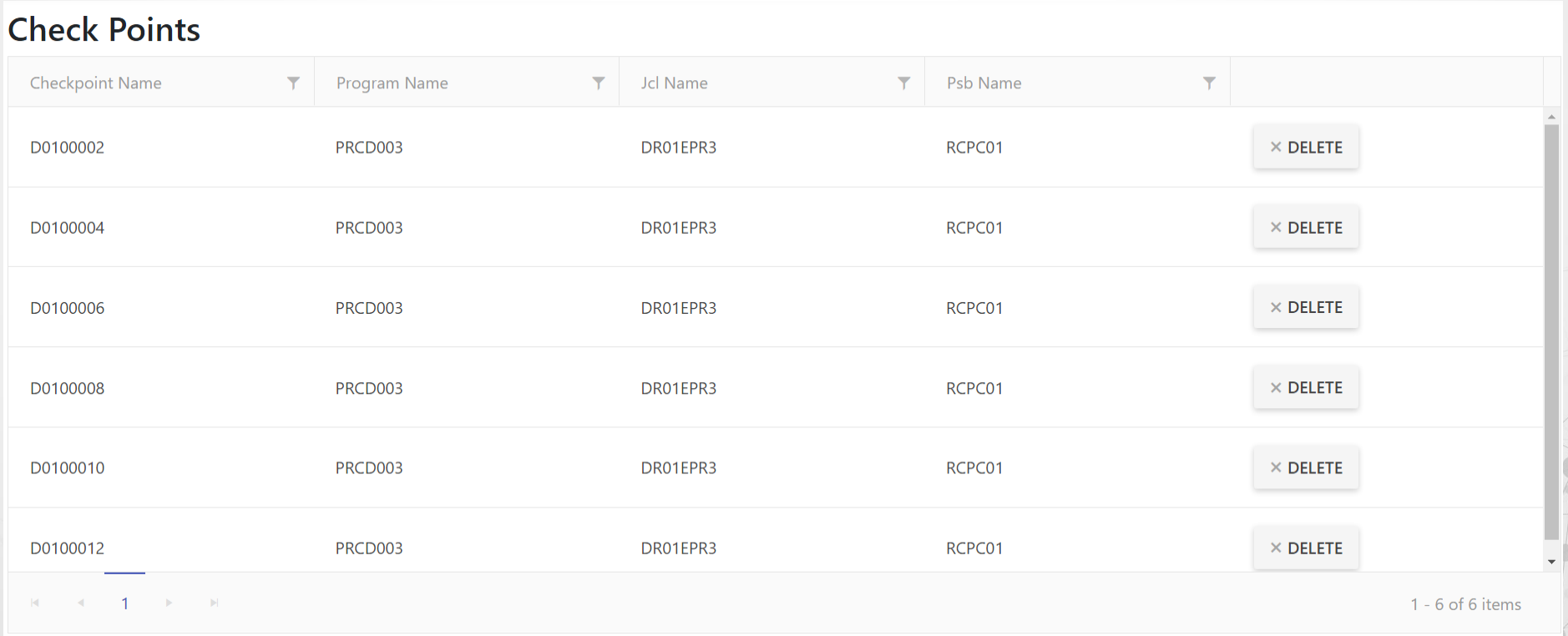
IMSql – Checkpoint (Raincode Console)

The screenshot below shows the Checkpoints created after the **first run** of the checkpoint/restart demo.



IMSql – Checkpoint (Raincode Console)

The screenshot below shows the Checkpoints created after the **second run** of the checkpoint/restart demo.

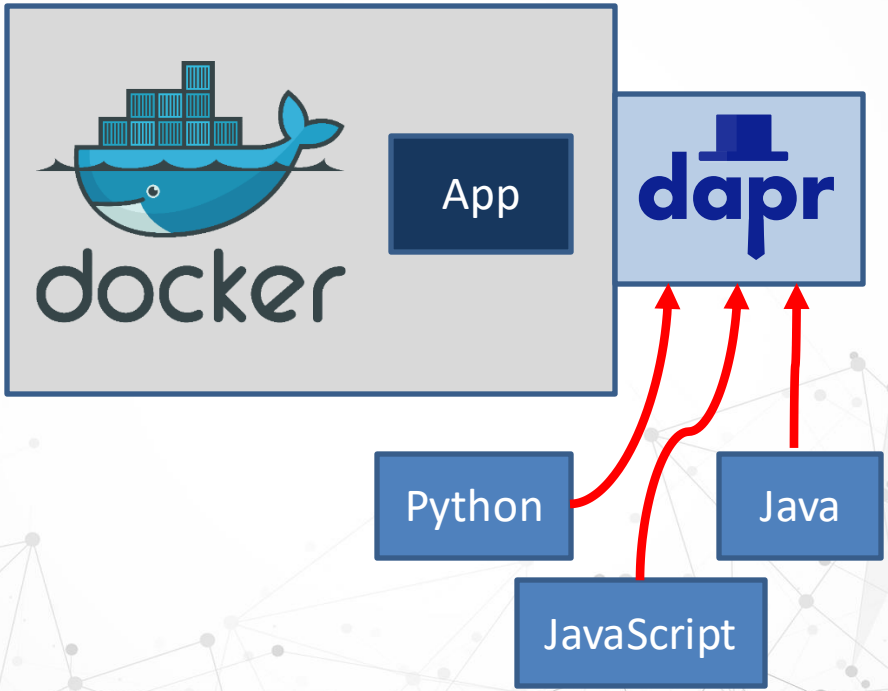


Cloud Native Deployment

More than your good old mainframe migration

- Deployment of mission-critical business logic at scale
  - Hundreds of transactions
  - Hundreds of batch jobs
- Using state-of-the-art cloud-native technologies
  - .NET 10.0
  - Azure Container Apps
    - Kubernetes + Dapr
- Without the burden of antiquated infrastructure
- Easy consumption by applications written in modern languages
- Fulfilling the promise of digital transformation

Exposing mainframe business logic for non-mainframe consumption

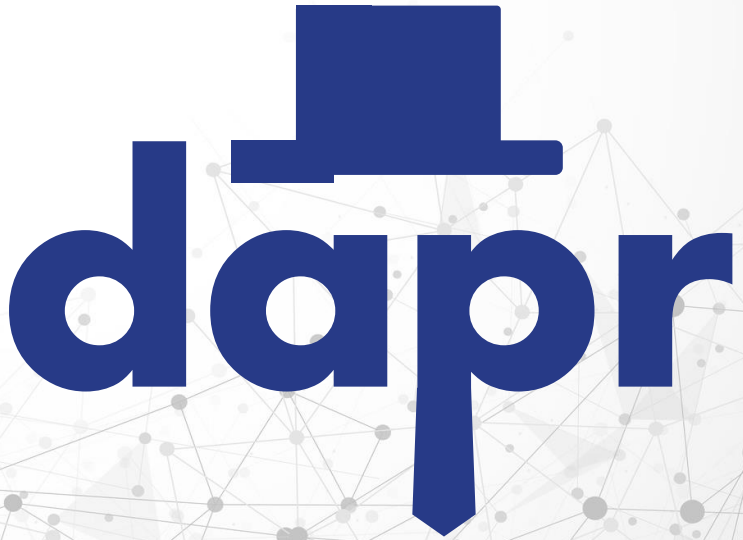


The mainframe code becomes a service
Usable from any application with a HTTP client

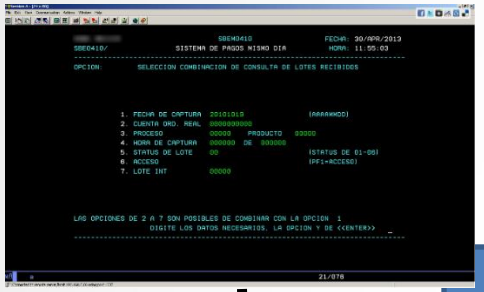
Dapr

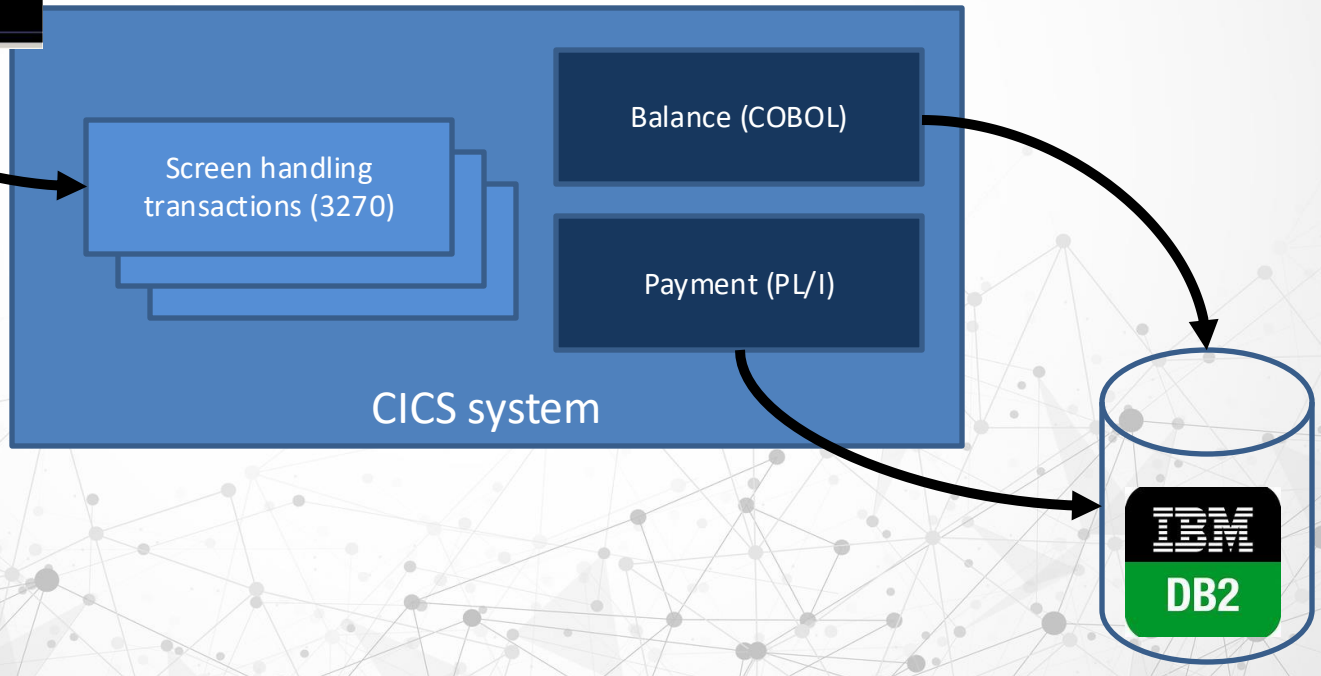
- Portable event-driven runtime for building microservices
- Open source, driven by Microsoft
- Language agnostic
- Platform agnostic
  - Cloud
  - Edge devices

https://dapr.io



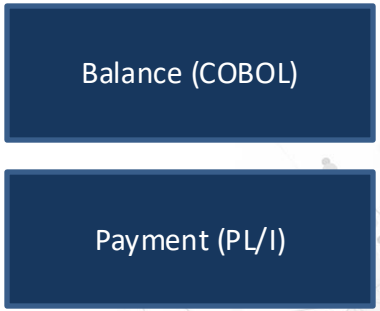
Cloud Native Demo architecture: the starting point



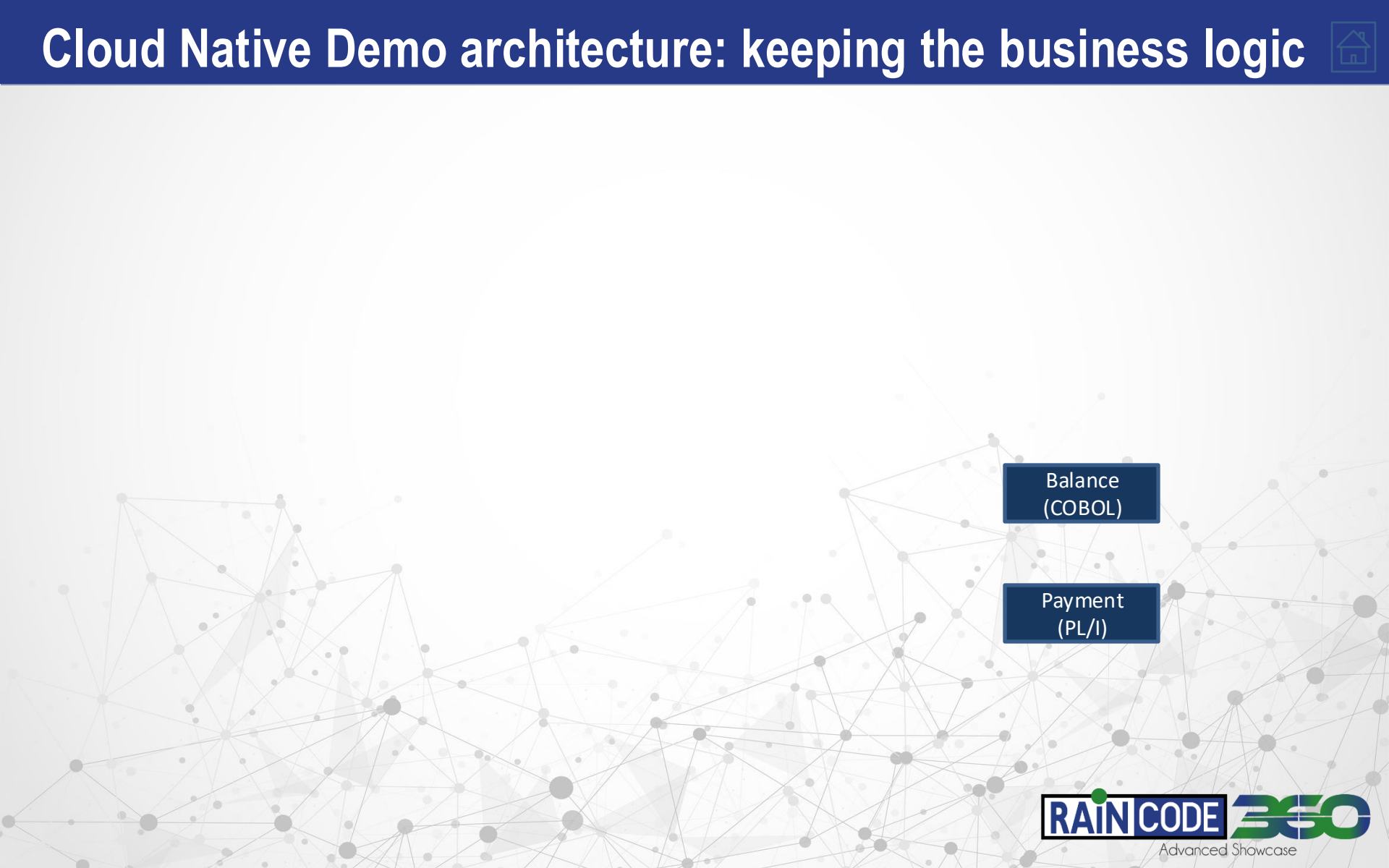


RAINCODE 360
Advanced Showcase

Cloud Native Demo architecture: keeping the business logic

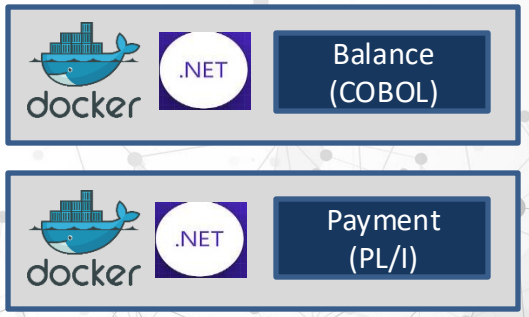


Cloud Native Demo architecture: keeping the business logic



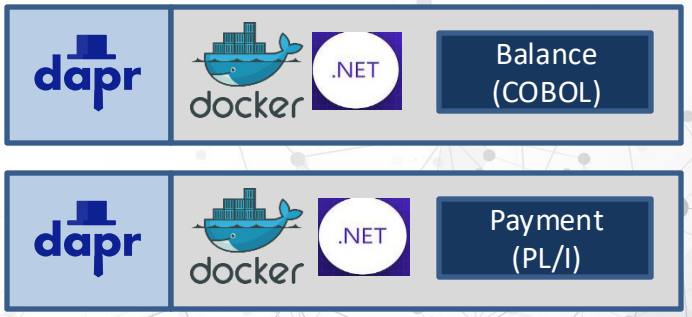
RAINCODE 360
Advanced Showcase

Cloud Native Demo architecture: encapsulation in containers

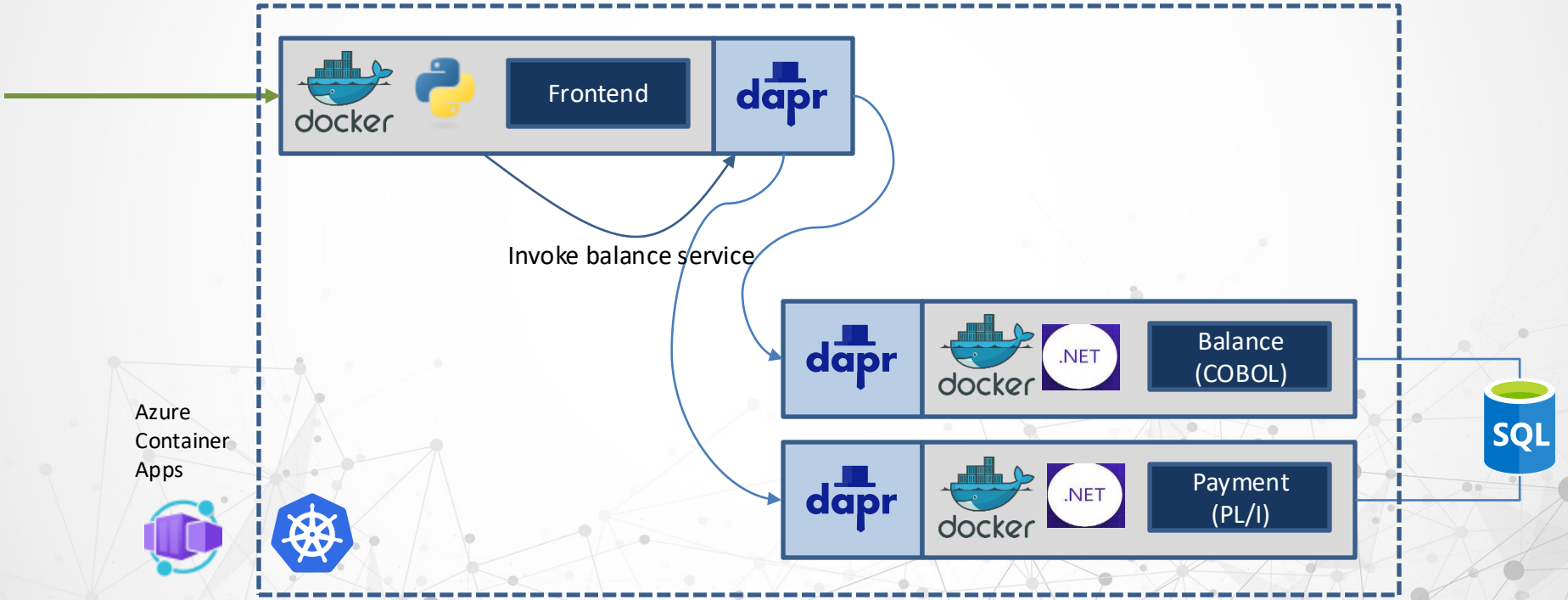


RAINCODE 360
Advanced Showcase

Cloud Native Demo architecture: injecting a Dapr sidecar



Cloud Native Demo architecture



Cloud Native Demo – Automated structure/JSON mapping

```
01 W-COMMAREA.
    …
    10 C007.
        20 C007-ACCOUNTNUMBER PIC X(20).
        20 C007-AMOUNT PIC ---.---.--9,99.
    …
```

JSON input request body to balance service:

```
{
    "C007": {
        "C007-ACCOUNTNUMBER": "BE28-..."
    }
}
```

RAINCODE 360 Advanced Showcase

Cloud Native Demo – Automated structure/JSON mapping

```
01 W-COMMAREA.
    …
    10 C007.
        20 C007-ACCOUNTNUMBER PIC X(20).
        20 C007-AMOUNT PIC ---.---.--9,99.
    …
```

JSON output response body from balance service:

```
{
    "C007": {
        "C007-ACCOUNTNUMBER": "BE28-...",
        "C007-AMOUNT": 123.45
    }
}
```

RAINCODE BEO Advanced Showcase

Cloud Native Demo Setup

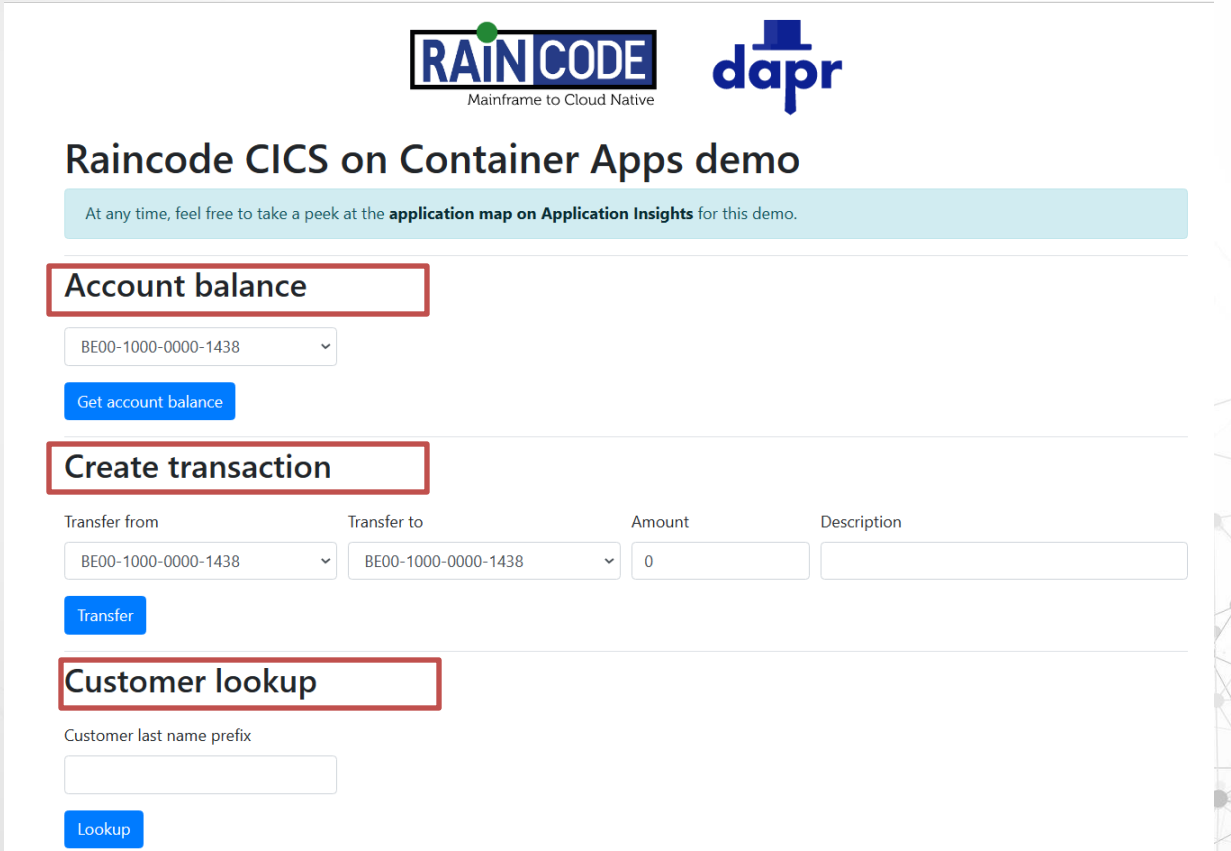
- Raincode 360 provides the Cloud Native demo out-of-the-box
  - Azure resources are created (Azure Container Apps Environment, etc.)
  - The Raincode Bank Application demo gets automatically deployed onto the Container Apps
  - Dapr is enabled on each of these Container Apps
- Deployed with the distributed tracing feature of Dapr enabled, allowing for application analysis through Azure Application Insights.
- The Cloud Native demo shares the same SQL database as the CICS demo.
  - Database changes made on one side will be reflected on the other.
- The demo wraps 4 programs from the “Qix - Emulator - Bank Application“ demo as Dapr microservices:
  - Account list (`COB010A.cob`)
  - Account balance (`COB011.cob`)
  - Payment (`PLI001.pli`)
  - Customer list (`COB004A.cob`)

RAINCODE 360 Advanced Showcase

Cloud Native Demo – How to demonstrate

- Click on the Cloud native demo 

desktop shortcut to access the demo frontend.



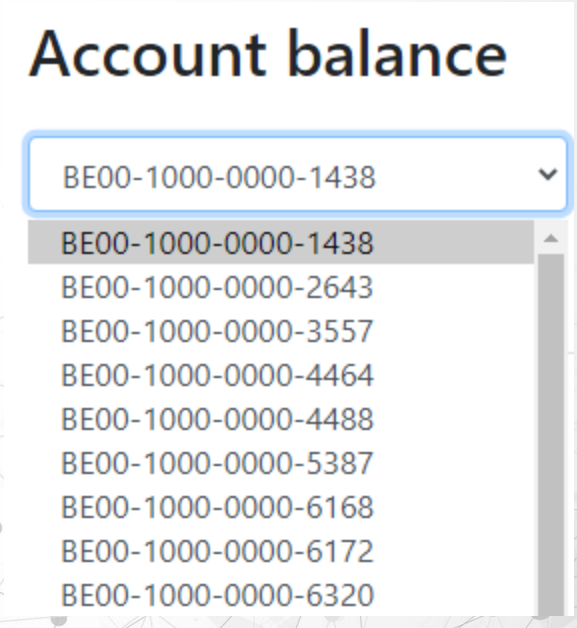
Interface is divided into three sections

RAINCODE B2C Advanced Showcase

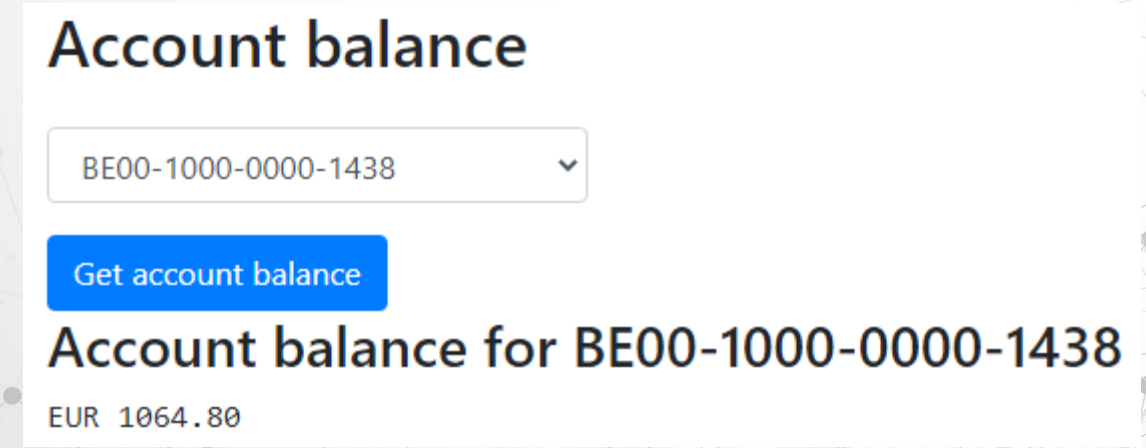
Cloud Native Demo – Web Frontend

Account Balance

- Select the account from drop-down menu of the account number box



- Click on 'Get account balance' for the balance inquiry

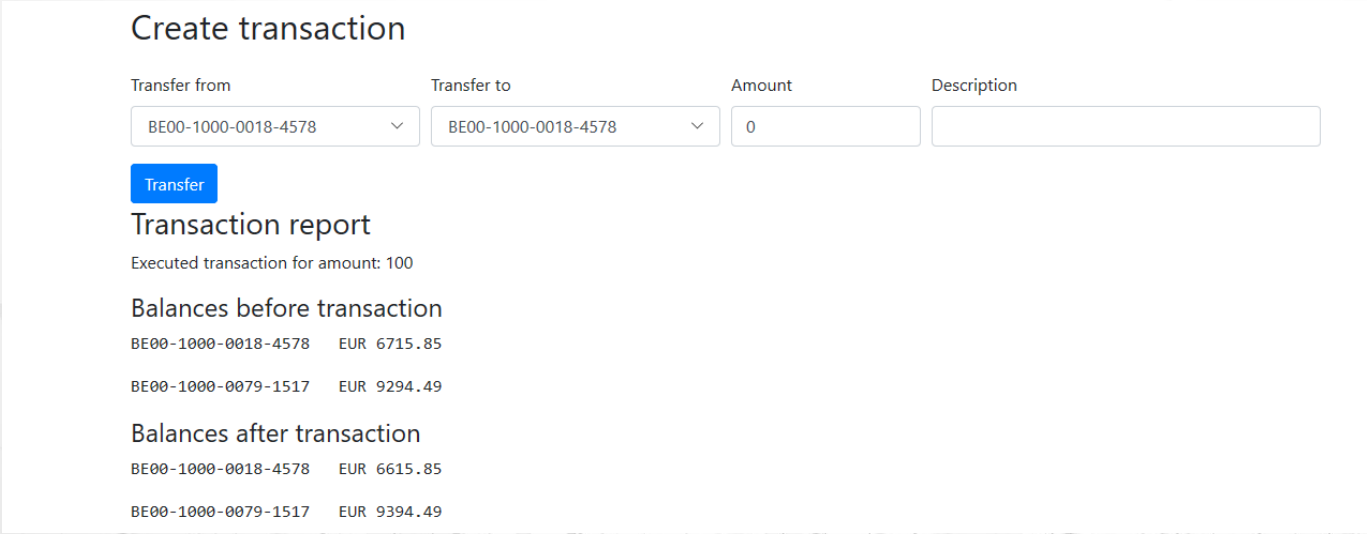


RAINCODE BEO Advanced Showcase

Cloud Native Demo – Web Frontend

Create Transaction

- Creates the payment
- Select **Transfer from** to **Transfer to**
- Add **Amount** and **Description**, and press the transfer button

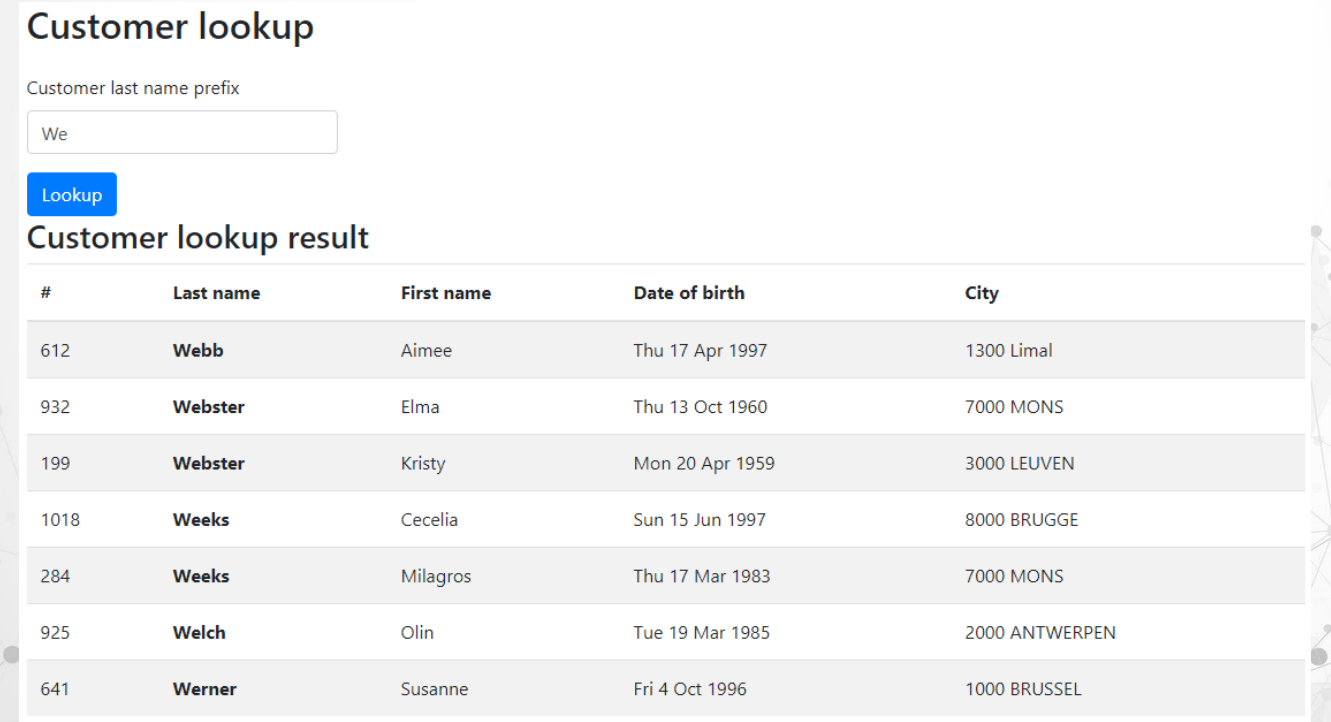


RAIN CODE B2C Advanced Showcase

Cloud Native Demo – Web Frontend

Customer lookup

- Looks for the customer in the database that matches the prefix entered.
- Click Lookup to populate the list on UI



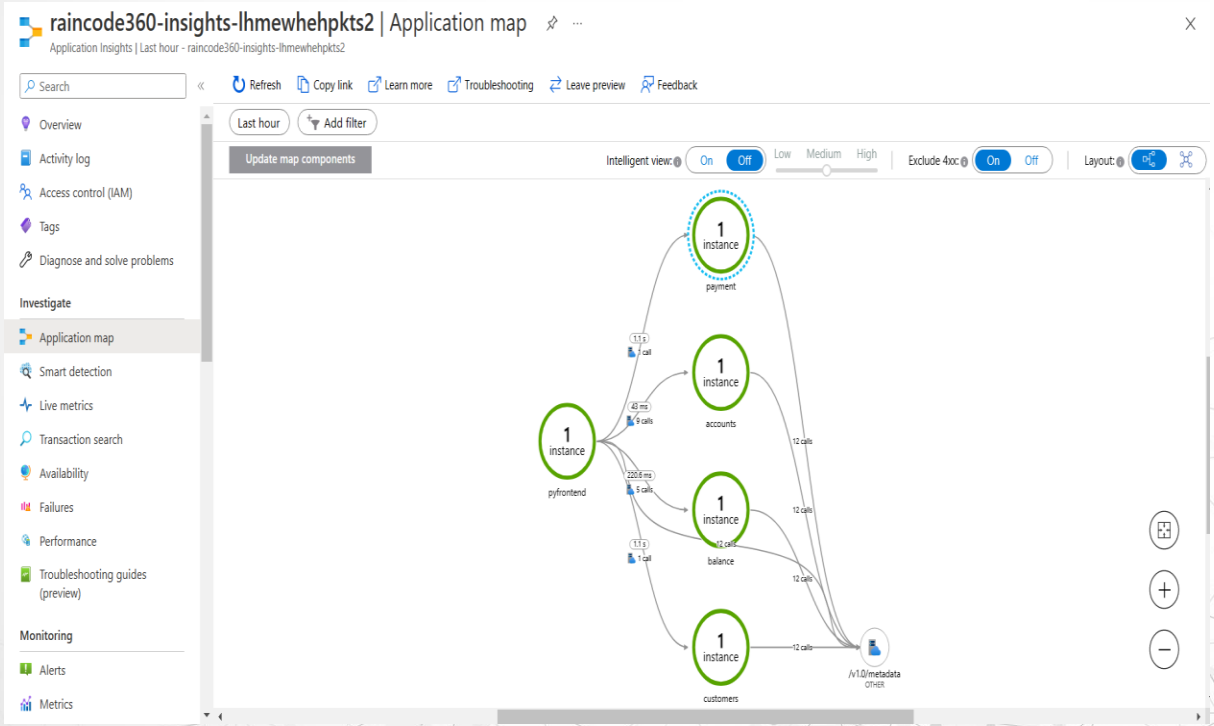
RAINCODE 360 Advanced Showcase

Cloud Native Demo – Dapr distributed tracing

- At any point, the Application Map can be reached through the link on the demo frontend.

Cloud Native Demo – Dapr distributed tracing

- Each node represents an application component
- Each edge represents a relationship (i.e., call) between identified components



Application Map identifies:

- Web frontend (pyfrontend)
- Legacy services:
  - Accounts (COBOL)
  - Balance (COBOL)
  - Customers (COBOL)
  - Payment (PL/I)
- User can see how many calls have been made, and their performance.

There will be a slight lag whilst information is populated in the Application Map after an action is made through the Cloud Native Demo frontend (usually less than 5 minutes).

RAINCODE 360 Advanced Showcase

Cloud Native Demo – Sources

- The sources for the Cloud Native demo are available on the Raincode 360 environment
- Contents:
  - The frontend written in Python 3 using the Django 3.2 framework
    - C:\Raincode360\Repositories\Demos\Azure\Dapr\pyfrontend
  - Our generated wrapper images for legacy programs & frontend from the Bank Application demo
    - https://hub.docker.com/search?q=raincode360
- Raincode tooling support for automated containerization of legacy applications is currently under development.

Cloud Native Demo – Rehost QIX-Emulator bank demo on AKS

RAINCODE 360
Advanced Showcase

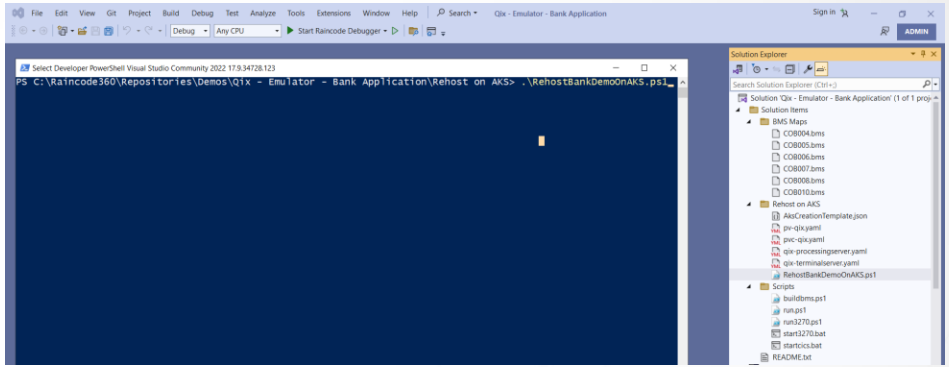
Cloud Native Demo – Rehost QIX-Emulator bank demo on AKS

- Rehost QIX emulator bank demo on AKS using containerization
- Prerequisites:
  - You must be a contributor to the Azure subscription where the Raincode 360 VM is hosted.
  - You must execute the PowerShell script (**RehostBankDemoOnAKS.ps1**) in the Administrator mode

Cloud Native Demo – Rehost QIX-Emulator bank demo on AKS

There are two ways to execute the demo:

1: Open the solution from the folder C:\Raincode360\Repositories\Demos\Qix - Emulator - Bank Application\ Rehost on AKS, and execute the PowerShell script (***RehostBankDemoOnAKS.ps1***) by navigating to the Tools -> Command line options -> Developer Powershell



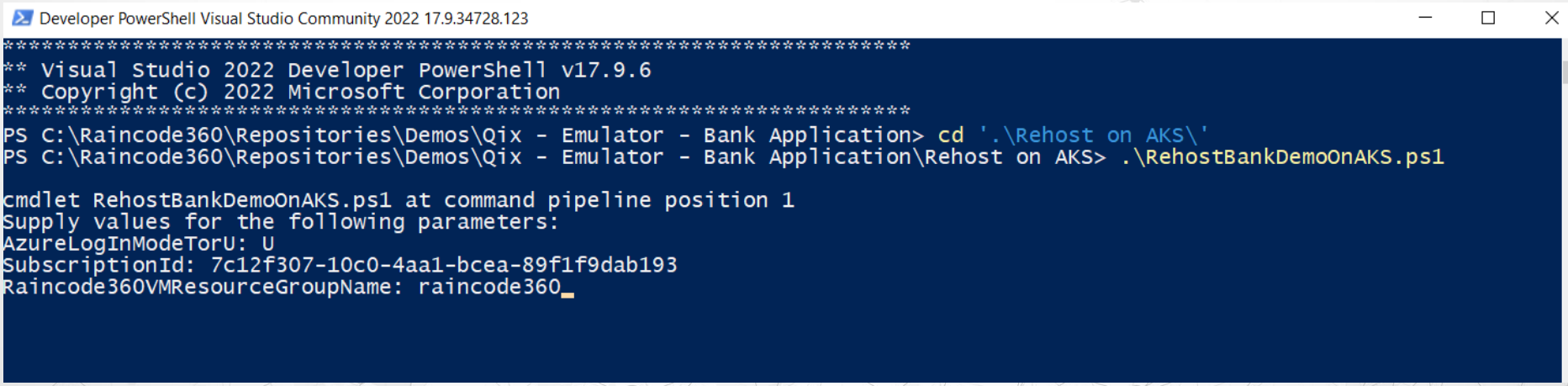
OR

2: Execute the demo through a shortcut placed on the desktop

RAINCODE 360
Advanced Showcase

Cloud Native Demo – Rehost QIX-Emulator bank demo on AKS

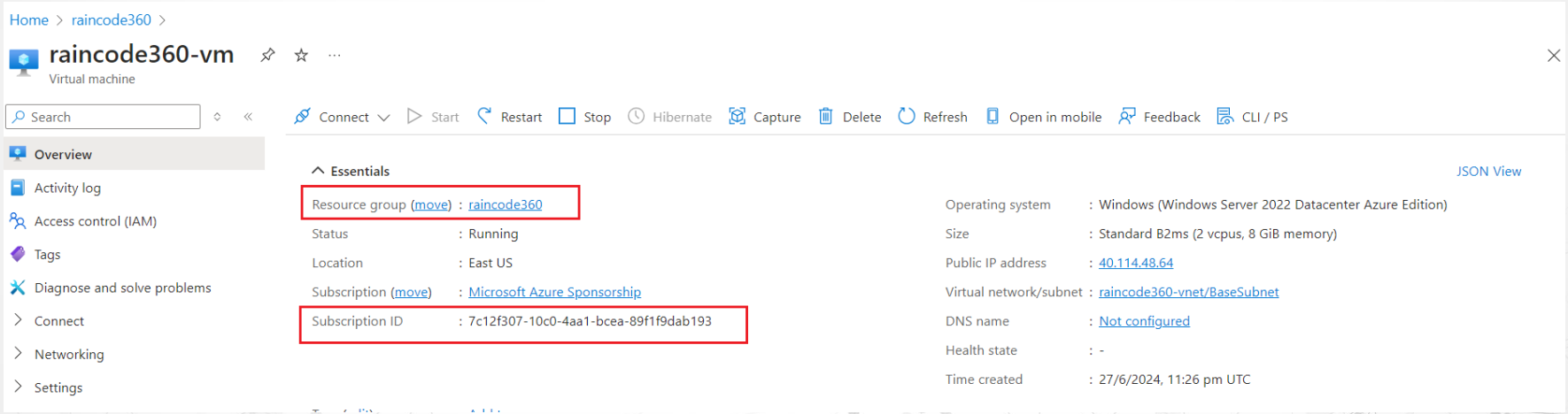
- Once you execute the script, it will ask you for the login method. There are two types of login methods:
  - User mode (U) – Login with the user credentials through the Azure portal. Azure portal opens in an explorer.
  - Tenet mode (T) - The tenant credentials required to Log in to Azure.
- Once you input the login method, it will ask for a SubscriptionId.
- Next, it will ask for the Raincode360-VM resource group's name for VNET peering.



RAINCODE 360 Advanced Showcase

Cloud Native Demo – Rehost QIX-Emulator bank demo on AKS

- SubscriptionId and the resource group name are copied from the Resource group of raincode360-vm on the Azure portal.



It is recommended that the AKS cluster and file storage be deployed within the same subscription ID.

RAINCODE 360
Advanced Showcase

Cloud Native Demo – Rehost QIX-Emulator bank demo on AKS

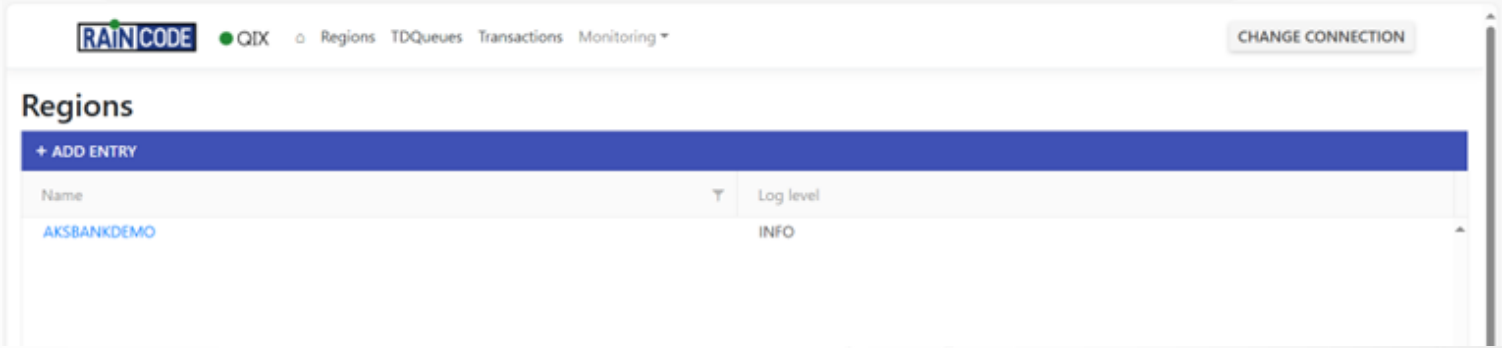
- When you press enter after providing the resource group name, it will set up a QIX region AKSBANKDEMO.

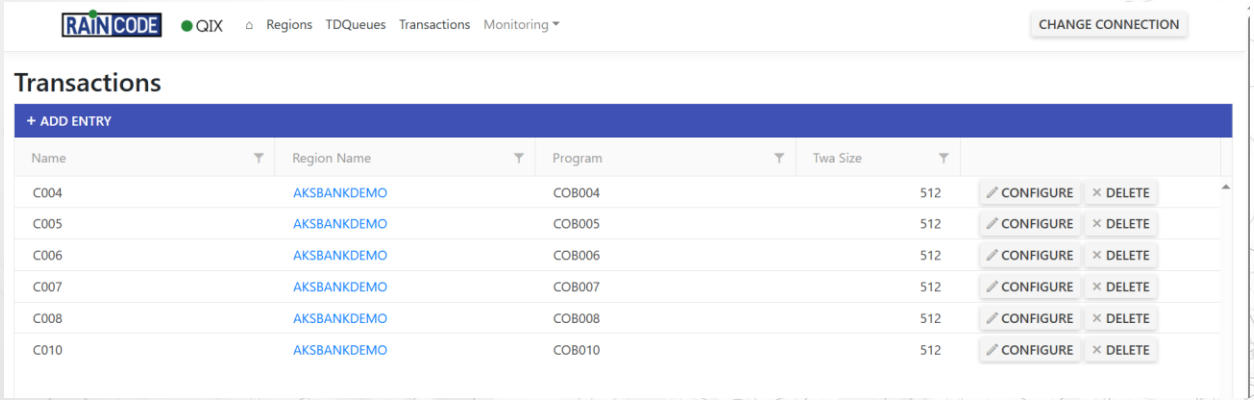
```
cmdlet RehostBankDemoOnAKS.ps1 at command pipeline position 1
Supply values for the following parameters:
AzureLogInModeTorU: U
SubscriptionId: 7c12f307-10c0-4aa1-bcea-89f1f9dab193
Raincode360VMResourceGroupName: raincode360
[2024-07-01 07:47:59] : Started the 'Rehosting of BankDemo Emulator Application to AKS' at: 07-01-2024 07:47
[2024-07-01 07:47:59] : Building solution C:\Raincode360\Repositories\Demos\Qix - Emulator - Bank Application\Qix - Emulator - Bank Application.sln in Release
 mode...
[2024-07-01 07:47:59] : Writing stdout to C:\Raincode360\Logs\msbuild.20240701T0747596597Z.out.log.
[2024-07-01 07:47:59] : Writing stderr to C:\Raincode360\Logs\msbuild.20240701T0747596597Z.err.log.
[2024-07-01 07:48:06] : Solution C:\Raincode360\Repositories\Demos\Qix - Emulator - Bank Application\Qix - Emulator - Bank Application.sln built successfully
in Release mode.
[2024-07-01 07:48:06] : Created Dlls directory and copied the program executables
[2024-07-01 07:48:06] : Created Maps directory and copied the Bms map Xmls
[2024-07-01 07:48:06] : Writing stdout to C:\Raincode360\Logs\regioncreator.20240701T0748068423Z.out.log.
[2024-07-01 07:48:06] : Writing stderr to C:\Raincode360\Logs\regioncreator.20240701T0748068423Z.err.log.
[2024-07-01 07:48:08] : Created 'AKSBANKDEMO' region.
```

RAINCODE 360 Advanced Showcase

Cloud Native Demo – Rehost QIX-Emulator bank demo on AKS

- You can see the created QIX region through the Raincode console.





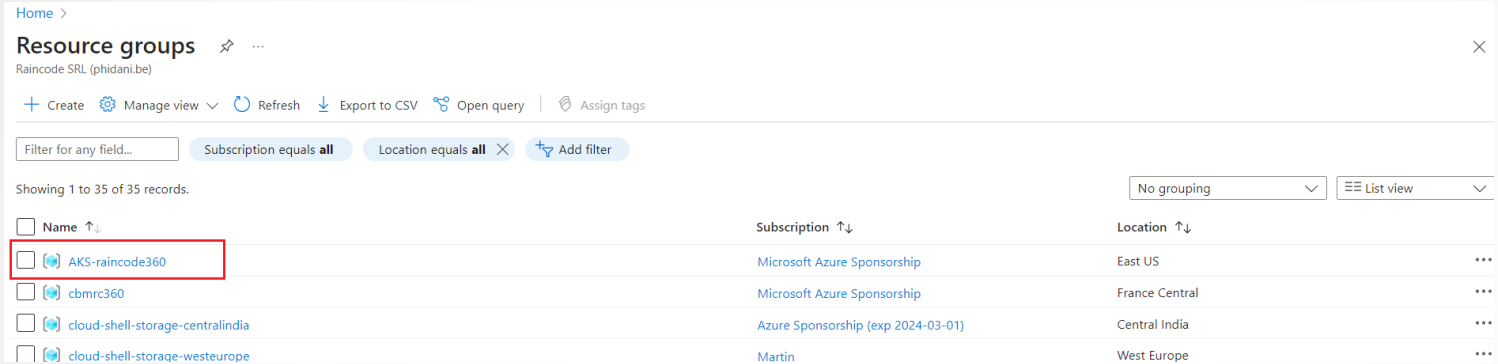
- You can check the transactions through QIX transactions tab in the Raincode console

For more details, refer to [Raincode Console]

RAINCODE 360 Advanced Showcase

Cloud Native Demo – Rehost QIX-Emulator bank demo on AKS

- The script will create a separate resource group for the AKS cluster and storage account.



- Under the new resource group, a storage account and a file share folder named **qixfileshare** are created

```
[2024-07-01 07:48:16] : Created the resource group: AKS-raincode360
[2024-07-01 07:48:16] : Creating Storage account: qixstorageaccountwqvst
[2024-07-01 07:48:36] : Storage account: qixstorageaccountwqvst created under resource group: AKS-raincode360 at location: eastus
[2024-07-01 07:48:36] : Created the Fileshare 'qixfileshare'
```

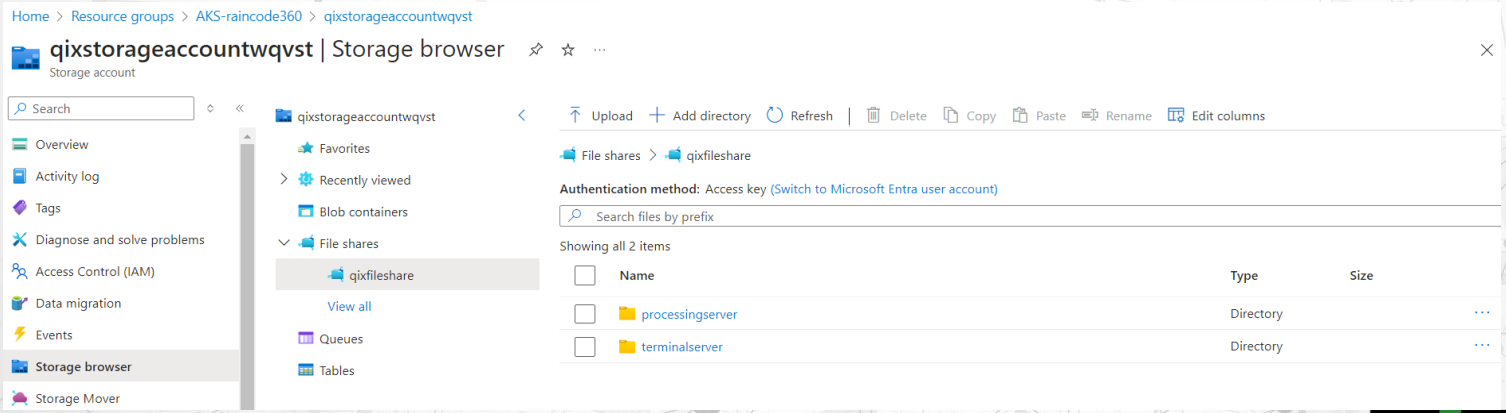
RAINCODE 360 Advanced Showcase

Cloud Native Demo – Rehost QIX-Emulator bank demo on AKS

- In qixfileshare, two directories, a processingserver and a terminalserver, are created, and license files are uploaded to both. BMS XMLs are uploaded to the terminalserver, and program executables are uploaded to the processingserver.

```
[2024-07-01 07:48:36] : Created the directory 'processingserver' under FileShare 'qixfileshare'
[2024-07-01 07:48:36] : Created the directory 'terminalserver' under FileShare 'qixfileshare'
[2024-07-01 07:48:36] : Uploaded license to both 'terminalserver' and 'processingserver' folders under fileshare 'qixfileshare'
[2024-07-01 07:48:38] : Uploaded BMS xmls to 'terminalserver' folder under fileshare 'qixfileshare'
[2024-07-01 07:48:42] : Uploaded program executables to 'processingserver' folder under fileshare 'qixfileshare'
[2024-07-01 07:48:42] : Installing Az module....
```

- In the AKS resource group, navigate to the file share folder. Within this folder, you will find the processingserver and terminalserver directories, which contain the License file, BMS XMLs, and DLLs.



RAINCODE 360 Advanced Showcase

Cloud Native Demo – Rehost QIX-Emulator bank demo on AKS

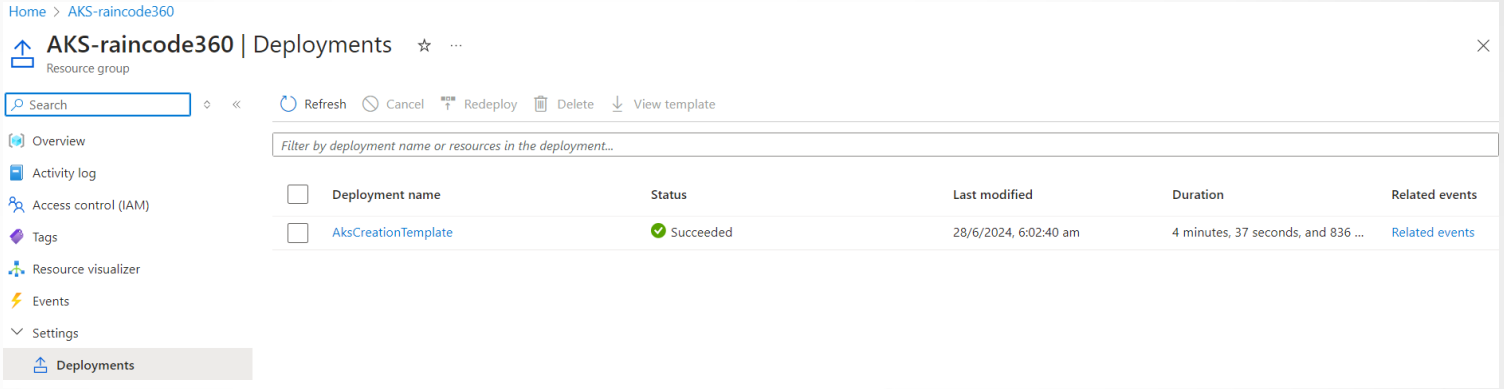
- Further, the script installs the Kubernetes cluster and creates peer networking between sqlserver and AKS virtual network.

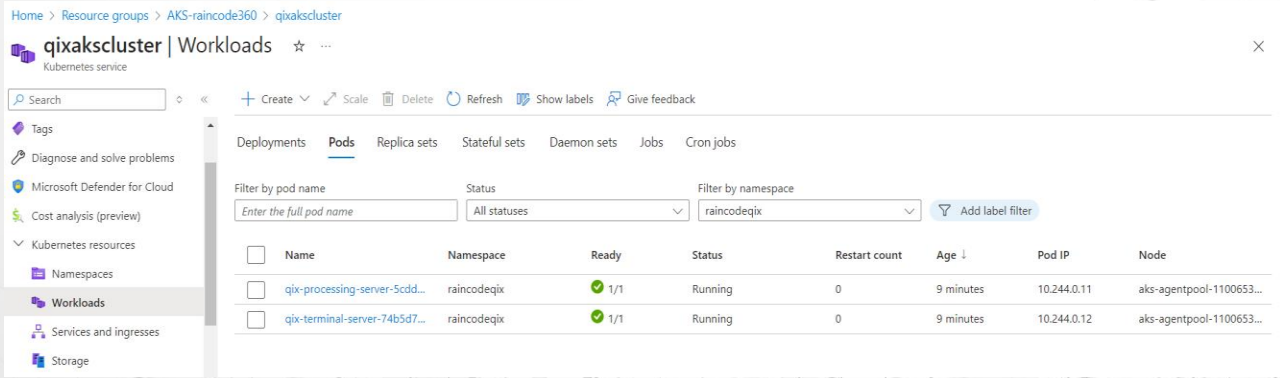
```
[2024-07-01 07:48:43] : Installing Kubectl ....
[2024-07-01 07:48:43] : Kubectl installation completed successfully.
[2024-07-01 07:48:43] : Creating kubernetes service using AKsTemplatefile 'C:\Raincode360\Repositories\Demos\Qix - Emulator - Bank Application\Rehost on AKS\A
ksCreationTemplate.json' under resouce group 'AKS-raincode360'.
[2024-07-01 07:53:33] : Kubernetes service created successfully.
[2024-07-01 07:53:33] : Creating network peer between SqlServer and AKS virtual networks
[2024-07-01 07:53:45] : Peer networking added between 'raincode360-vnet' & 'aks-vnet-35064155'
[2024-07-01 07:53:56] : Peer networking added between 'aks-vnet-35064155' & 'raincode360-vnet'
[2024-07-01 07:53:56] : Connecting to Kubernetes cluster to deploy resources...
[2024-07-01 07:53:57] : NameSpace created: 'raincodeqix'
[2024-07-01 07:53:57] : Secret created: 'qix-storage-secret'
[2024-07-01 07:53:57] : Secret created: 'qix-config-secret'
[2024-07-01 07:53:58] : Qix Persistant volume created
[2024-07-01 07:53:58] : Qix Persistant volume claim created
[2024-07-01 07:53:58] : Qix processing server deployment created
[2024-07-01 07:53:59] : Qix terminal server deployment & service created

NAME                                    READY   STATUS              RESTARTS   AGE   IP       NODE                                NOMINATED NODE   READINESS
GATES
qix-processing-server-5cdd5bc7c8-k8rrl   0/1     ContainerCreating   0          1s    <none>   aks-agentpool-11006533-vmss000001   <none>           <none>
qix-terminal-server-74b5d78f6b-7sscm     0/1     Pending             0          0s    <none>   <none>                              <none>           <none>
NAME                 TYPE           CLUSTER-IP     EXTERNAL-IP   PORT(S)          AGE   SELECTOR
qix-terminal-server   LoadBalancer   10.0.140.141   <pending>     993:30485/TCP   0s    app=qix-terminal-server
[2024-07-01 07:53:59] : Waiting 45 seconds for containers & service to be live...

NAME                                    READY   STATUS    RESTARTS   AGE   IP            NODE                                NOMINATED NODE   READINESS GATES

qix-processing-server-5cdd5bc7c8-k8rrl   1/1     Running   0          47s   10.244.0.11   aks-agentpool-11006533-vmss000001   <none>           <none>
qix-terminal-server-74b5d78f6b-7sscm     1/1     Running   0          46s   10.244.0.12   aks-agentpool-11006533-vmss000001   <none>           <none>
NAME                 TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)          AGE   SELECTOR
qix-terminal-server   LoadBalancer   10.0.140.141   4.156.121.63   993:30485/TCP   46s   app=qix-terminal-server
[2024-07-01 07:54:45] : About to wait 120 seconds for you to inspect and ready for disconnecting....
```

RAINCODE 360 Advanced Showcase

Cloud Native Demo – Rehost QIX-Emulator bank demo on AKS

- Under the deployments tab, you can find the AKSCreation template.



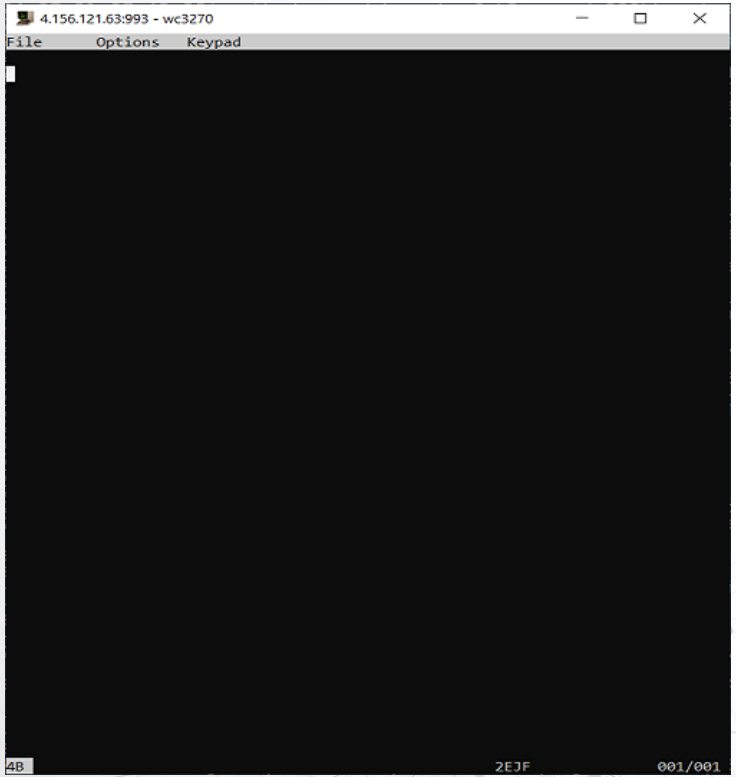


- If you choose the Pods tab under the Workloads, you can see the running containers.

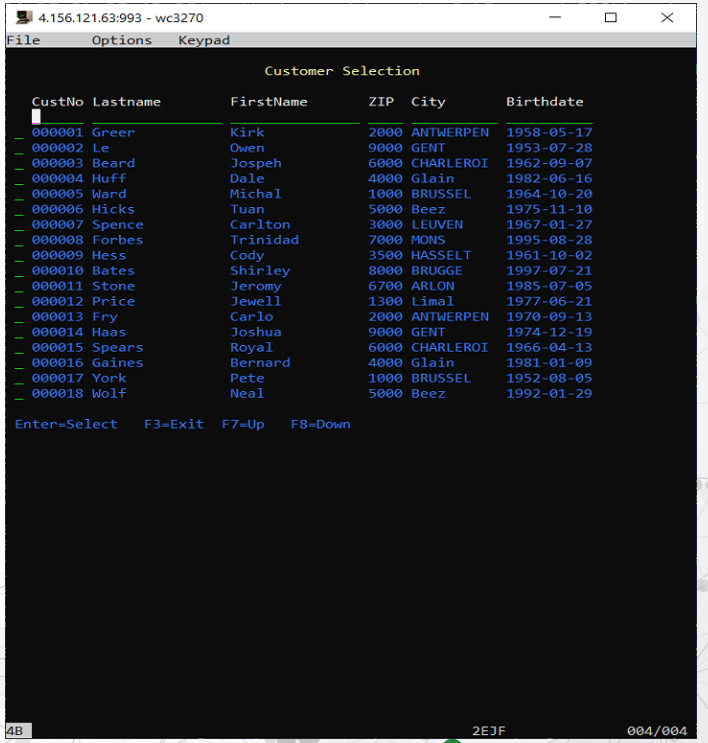
RAINCODE 360 Advanced Showcase

Cloud Native Demo – Rehost QIX-Emulator bank demo on AKS

Once the script is successfully executed, it will connect you to the 3270 terminal



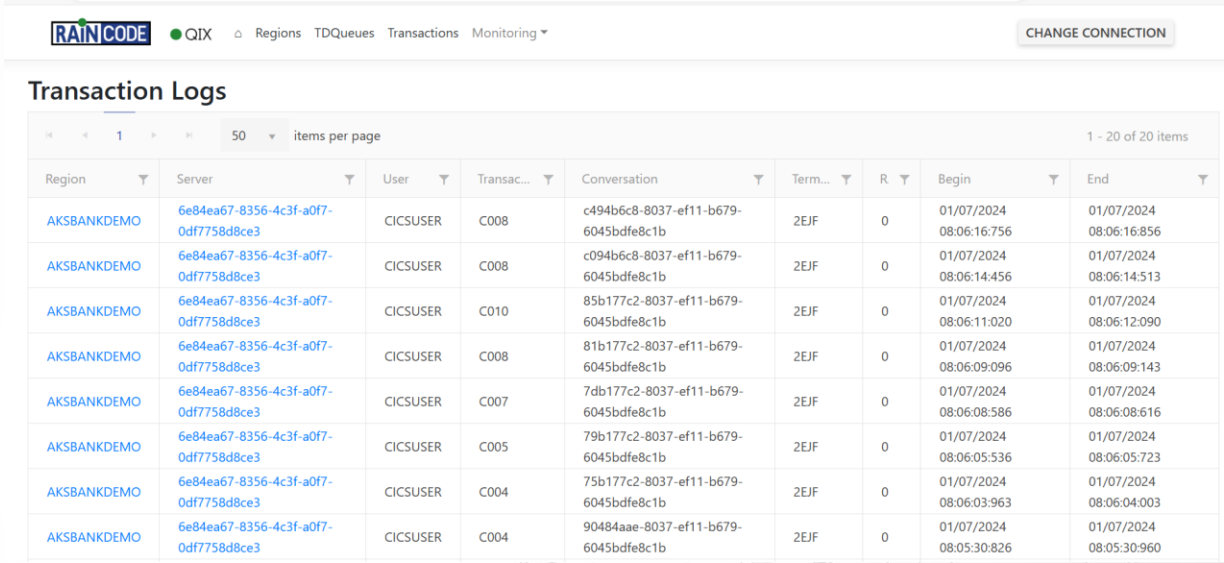
Type C004 the first transaction id of QIX emulator bank application and press enter

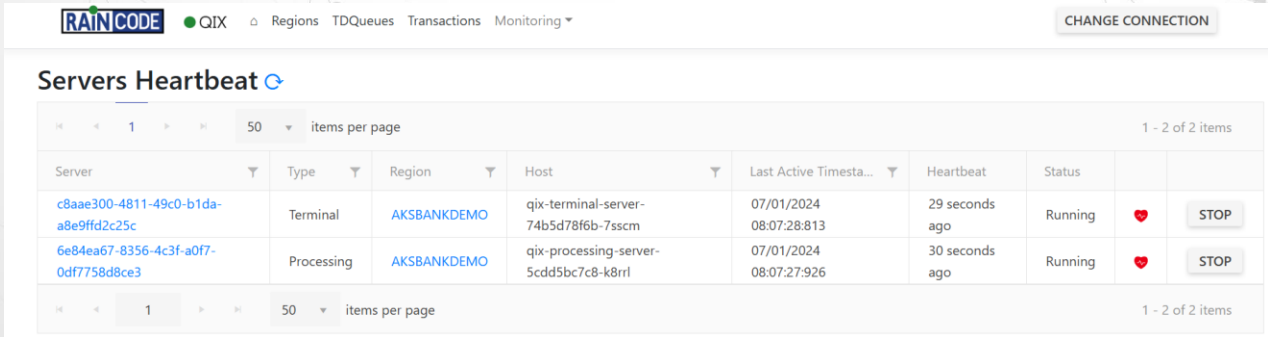


RAINCODE 360 Advanced Showcase

Cloud Native Demo – Rehost QIX-Emulator bank demo on AKS

You can check the transaction logs by navigating to the QIX Monitoring tab→ Transaction Logs in the Raincode console





You can check the server heartbeat by navigating to the QIX monitoring tab→ Servers Heartbeat in the Raincode console

For more details, refer to [Raincode Console]

RAINCODE 360 Advanced Showcase

Cloud Native Demo – Rehost QIX-Emulator bank demo on AKS

- After completing your job, select Y to disconnect from the Azure session and Y to clean up the created AKS resources

```
[2024-07-01 07:54:45] : About to wait 120 seconds for you to inspect and ready for disconnecting....
Disconnect from the Azure Session(Y/N)?: y
Clean-up the created AKS resources(Y/N)?: y
[2024-07-01 08:09:16] : Removing Vnet peering from network - raincode360-vnet, Resource group - raincode360
[2024-07-01 08:09:27] : Cleaning up the resource group AKS-raincode360
[2024-07-01 08:15:44] : Logging out of Azure...
[2024-07-01 08:15:44] : Rehosting completed at:07-01-2024 08:15

Waiting for  0 seconds, press a key to continue ...
```

- If you select N, you must manually clean the created AKS resources

RAINCODE 360 Advanced Showcase

Expert mode

RAINCODE 360
Advanced Showcase

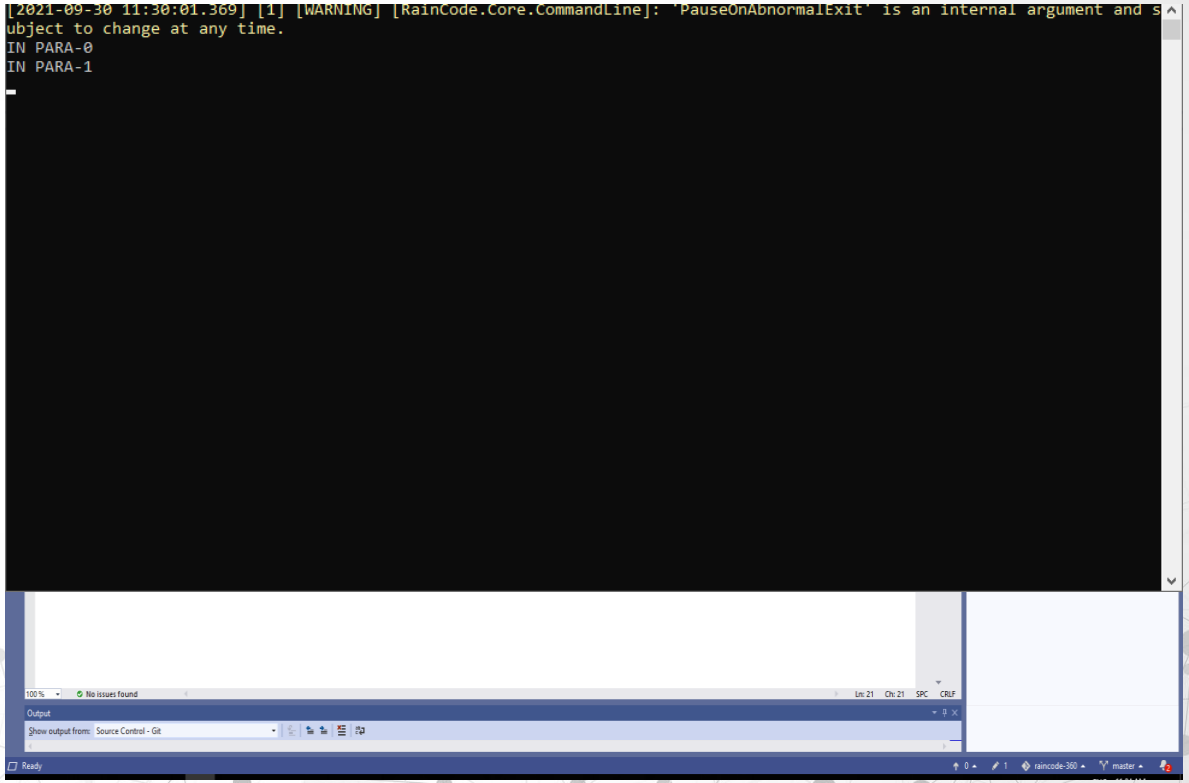
User-defined function

- The purpose of this demo is to demonstrate the usage of the Raincode scripting language to create a user-defined function to implement a coding guideline

- The user-defined function adds extra validation to the compiler so that it throws an error when the “GO TO” statement is encountered in the COBOL module

User defined function: Normal Flow

- Step 1: Open the solution from the folder Compiler – User Defined Coding Guideline
- Step 2: Select Build -> Rebuild Solution
- Step 3: In GOTOCBL.cob, set a breakpoint at GOBACK statement in PARA-1
- Step 4: Start the Raincode Debugger



Expected output: The execution will print “IN PARA-0” and “IN PARA-1” in consecutive lines.

- Step 5: Click continue or close the application window to exit the debugger

RAINCODE 360 Advanced Showcase

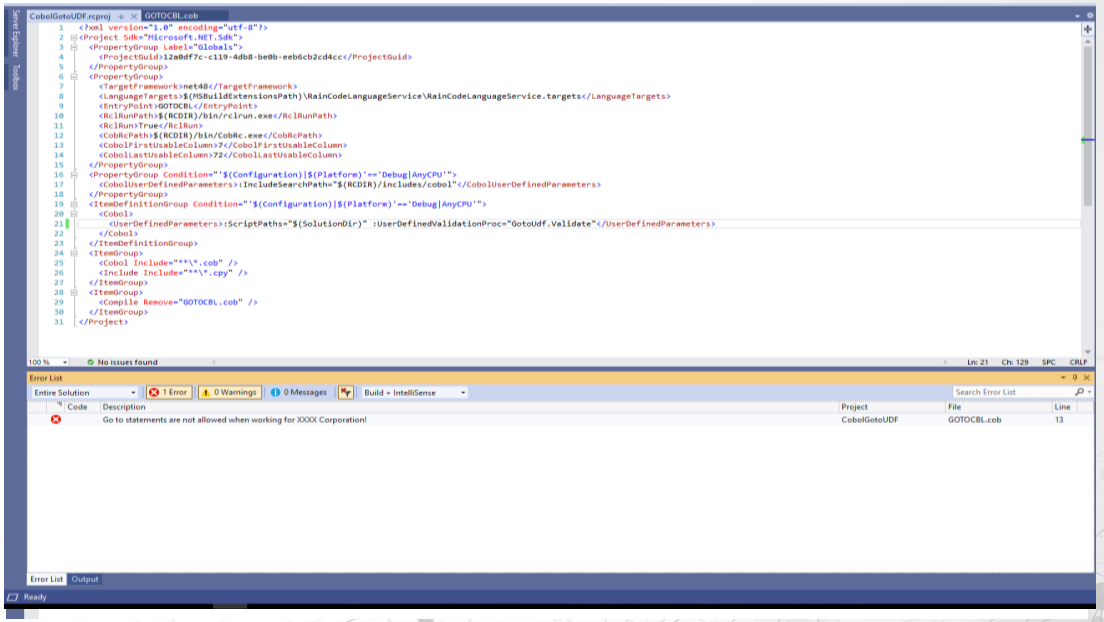
User defined function: Error Flow

- Step 1: Open the solution from the folder Compiler – User Defined Coding Guideline and click on CobolGotoUDF project from the solution explorer

- Step 2: From the displayed XML lines, uncomment " <UserDefinedParameters>:ScriptPaths="$(SolutionDir)" :UserDefinedValidationProc="GotoUdf.Validate"</UserDefinedParameters> "

- Step 3: Select Build -> Rebuild Solution

- Step 4: Start the Raincode Debugger



The compilation will fail, and an error message “Goto statements are not allowed when working for XXXX corporation!” will appear in the output tab

- Step 5: Click continue or close the application window to exit the debugger

RAINCODE BEO Advanced Showcase

RAINCODE HEADQUARTERS

Rue de la Caserne 45
1000 Brussels, Belgium

+32 2 522 06 63

info@raincode.com

RAINCODE INDIA

#1144, Guru Nilayam, 3rd Floor, Phase 2,
HSR Layout, Bangalore – 560 102, India

+91 99450 47258

info@raincode.com

www.raincode.com