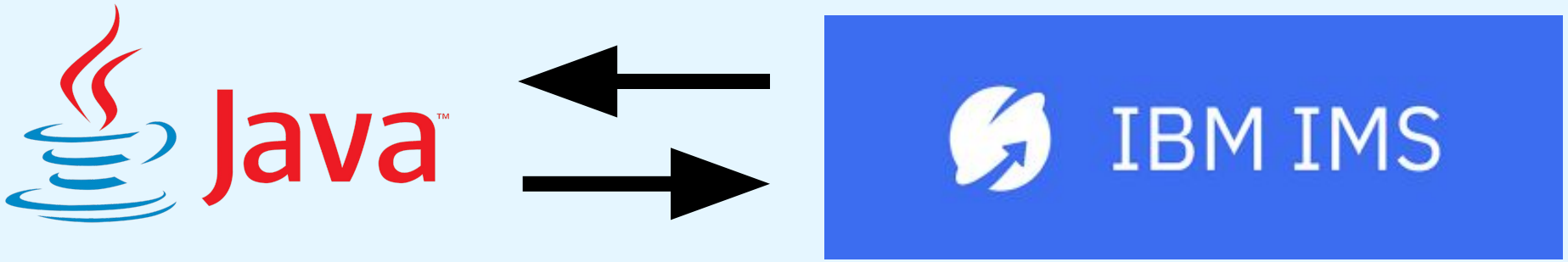
A Deep Dive into Java and IMS Interoperability



Andrew Urso

IMS Technical Specialist, Worldwide Systems Center

Contact Me – Andrew.urso@ibm.com

Today's Agenda

- Accessing IMS Using VS Code
- Updating Metadata with DDL
- IMS Universal Drivers Overview
- Three Flavors of IMS Java Interoperability
  - Accessing IMS resources with a distributed Java Program
  - Accessing IMS resources in a JBP region
  - Accessing IMS resources in a JMP region

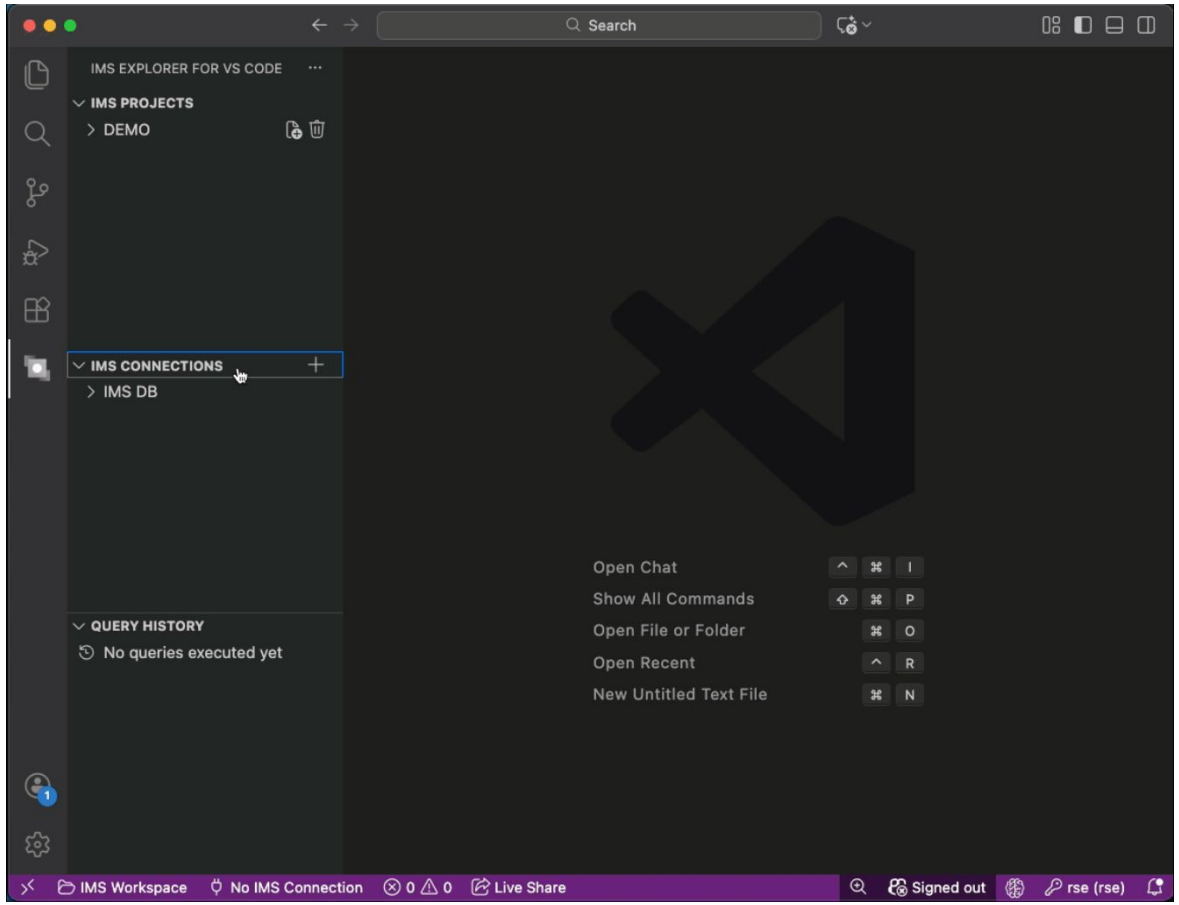
IMS Explorer for VsCode

IMS Explorer for VS Code supports the following actions:

- Create and edit projects, driver definitions, and database connections
- Connect to IMS Catalog and browse IMS databases
- Import and visualize PSBs and DBDs from IMS Catalog
- Access IMS data with SQL queries

Creating IMS Connection Profiles

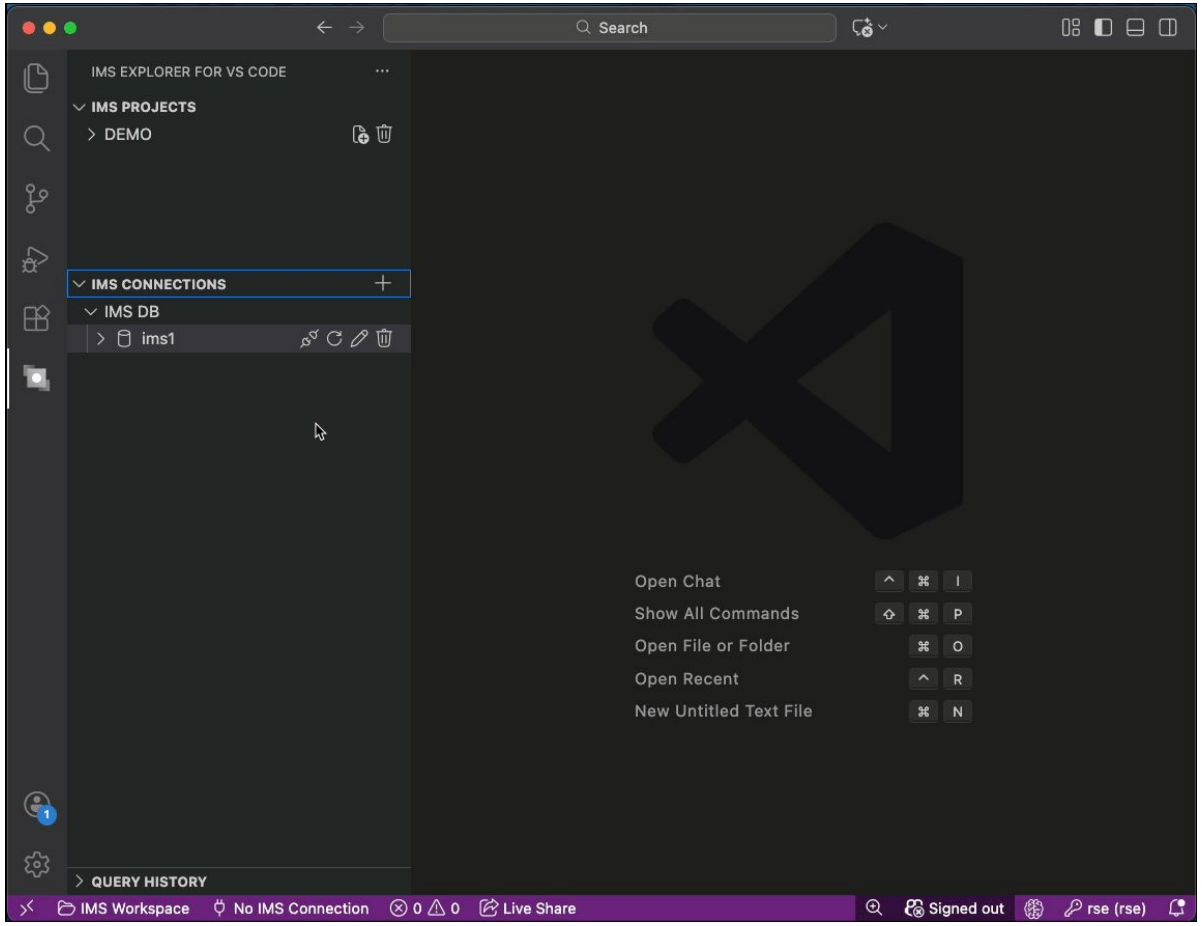
- IMS connection profiles are stored using a common *.json file
- IMS connections can be organized into folders and be expanded or collapsed



IMS Connection Tree and Table Views

The IMS Connections view can be used to:

- Add IMS catalog and database connections
- Expand and collapse database connection contents to view DBDs, PSBs, and PCBs
- Select an individual DBD, PSB, or PCB and right-click to view DDL segments, and source code



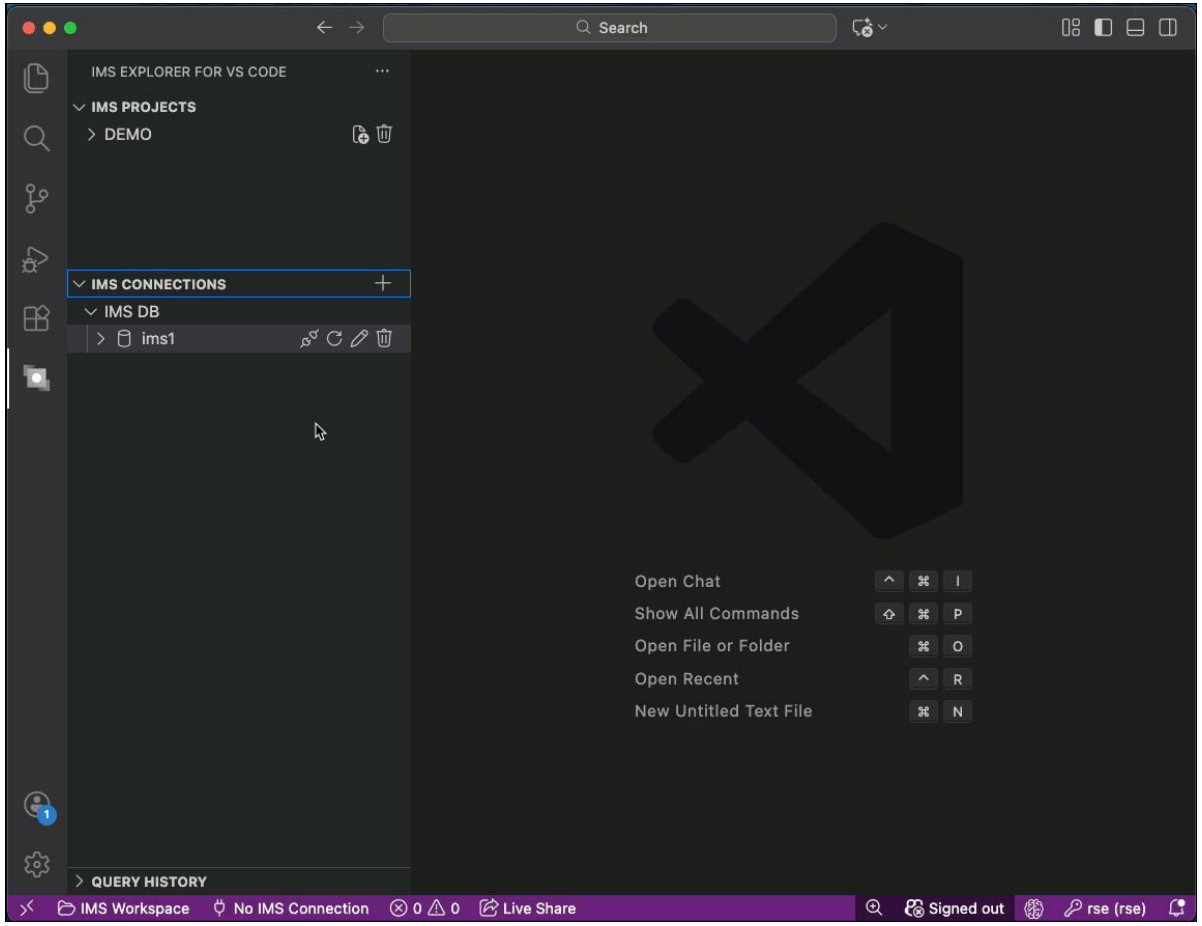
Creating IMS Projects

Creating an IMS Project allows for:

- Editing and executing SQL against a database
- Viewing, sorting, and filtering results
- Import and Visualize PSBs and DBDs

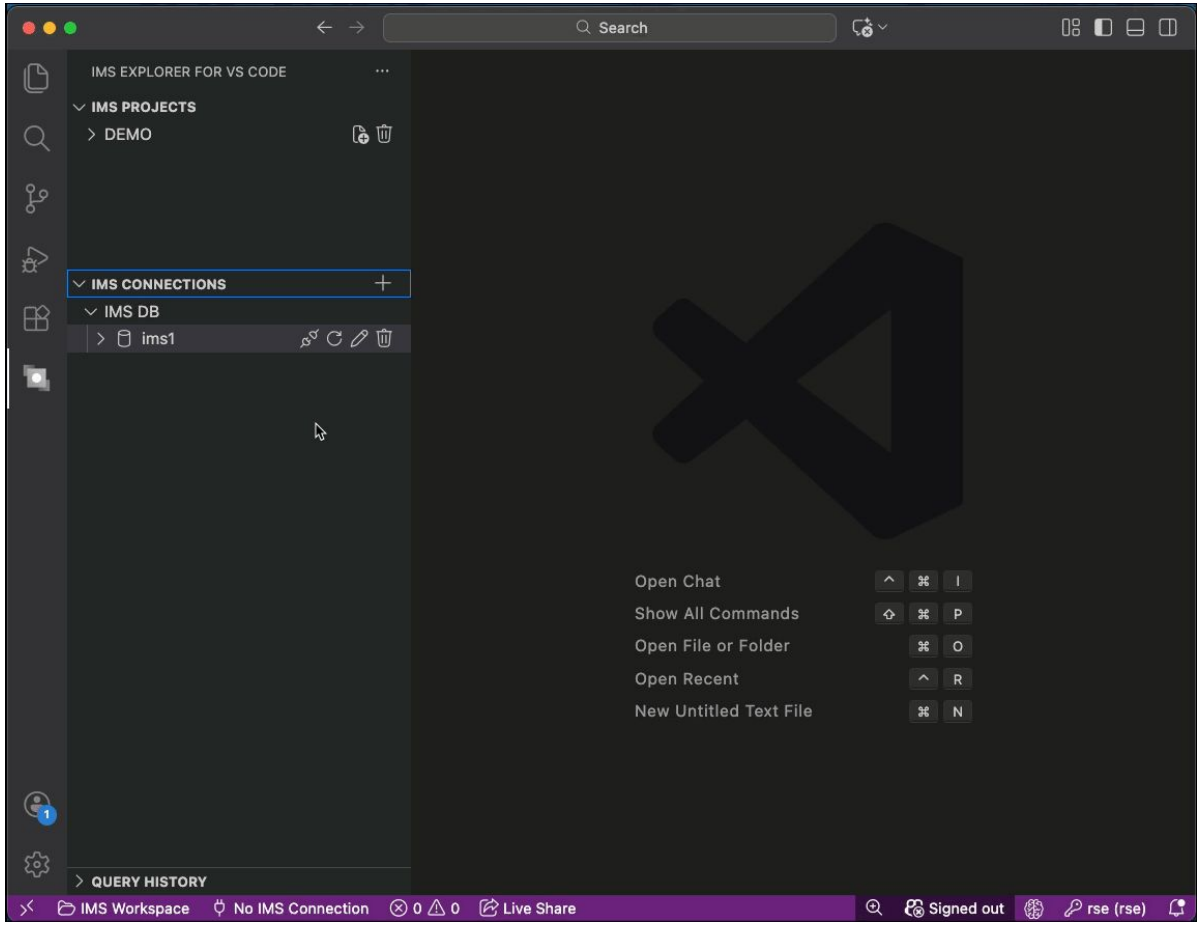
All result sets are stored in the Query History view.

Result sets can be downloaded into a CSV file.



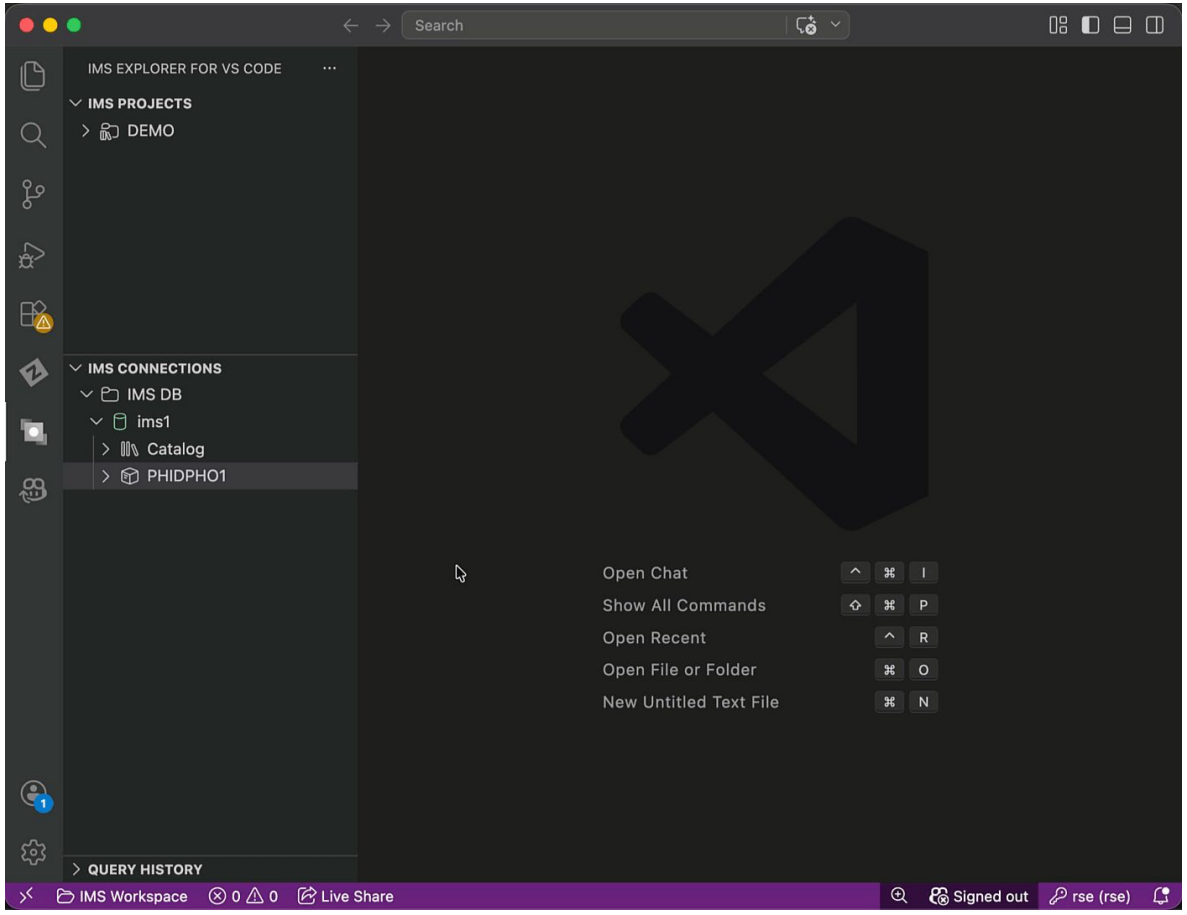
Import DBDs and PSBs from the IMS Catalog

- Filter and select which DBDs/PSBs to import into a project
- Optionally, you can choose to import any referenced DBDs



Visualize DBDs and PSBs

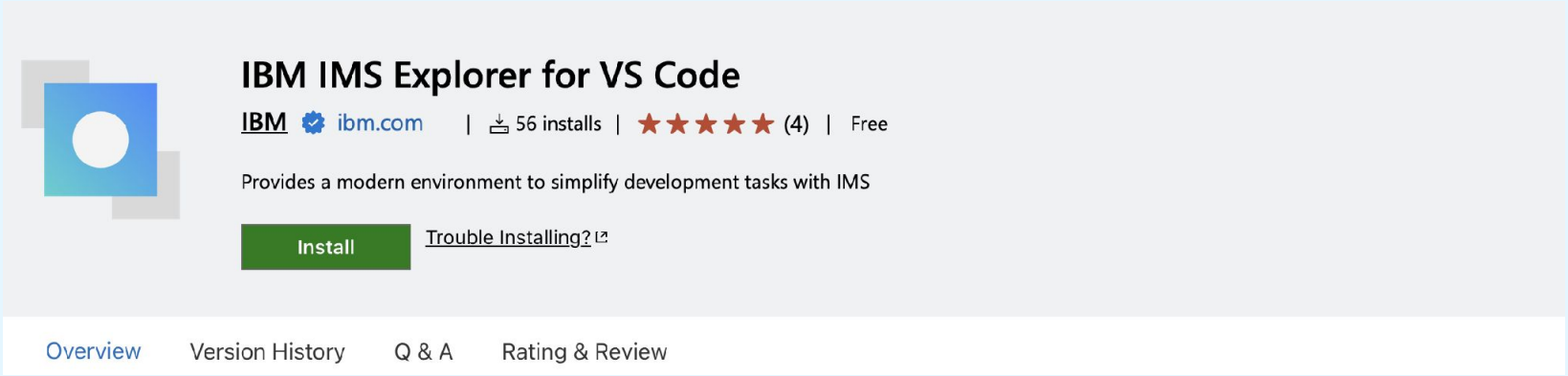
- View DBDs and PSBs in an interactive hierarchical visualization, allowing visual navigation of segments and view of segment fields by expanding their contents.
- Swap between source, DDL, tables, or visualization as needed using tabs or context menus.



Try it now!

IMS Explorer for VsCode is now available in the VsCode Extensions Marketplace!

You can find it here: ibm.biz/imsexplore



Using DDL to Create Field Metadata

Updating DBDs with DDL

DDL can be used to easily update DBD fields

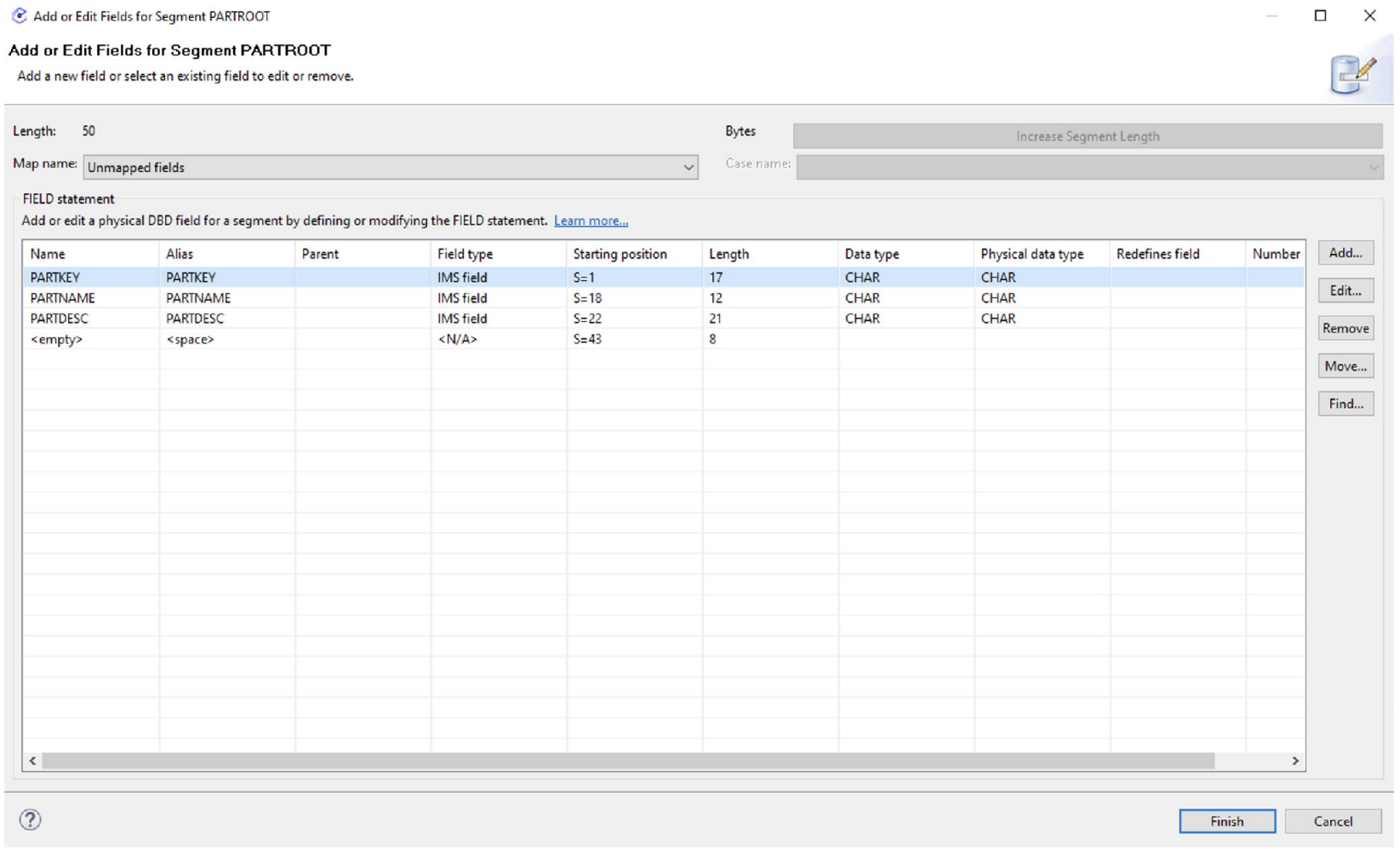
- The easy way
  - Explore for Development has tooling to easily create DDL statements

  Can be created by:

  - Manually specifying fields
  - Importing a COBOL copybook
- The ”hard” way
  - Write it yourself!

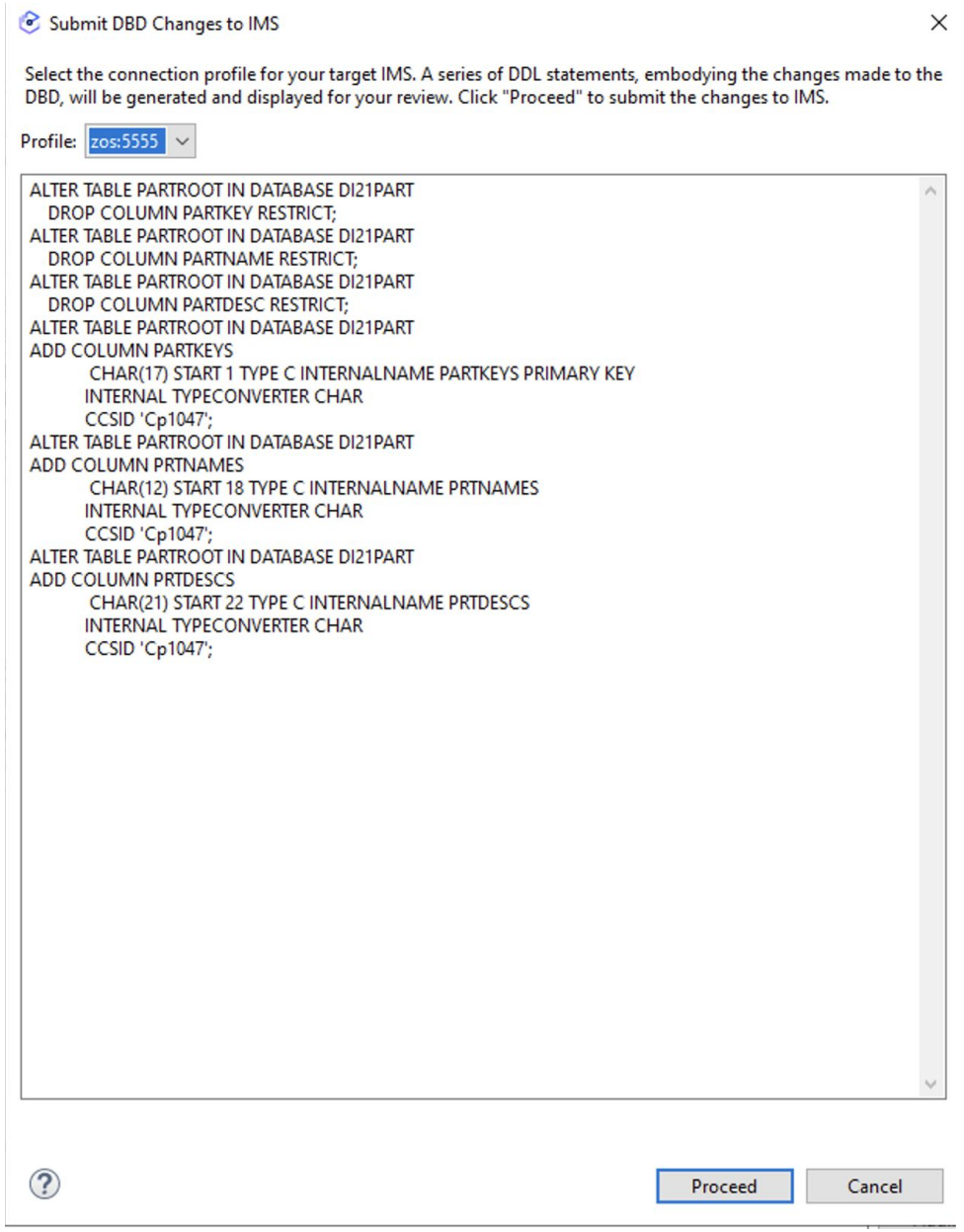
Updating DBDs with DDL

Fields can be easily added or edited



Updating DBDs with DDL

- Once changes are made the respective DDL is generated and can be submitted.
- After submitting the DDL, the DBD will need to be imported into the catalog.



IMS Universal Drivers – A Brief Overview

IMS Universal Drivers

The IMS Universal Drivers contain three main capabilities

- IMS Universal JDBC Driver
  - SQL-based standard interface for database access.
- IMS Universal DL/I Driver
  - Provides a stand-alone Java API for writing queries to IMS databases using semantics similar to traditional DL/I calls
- IMS Universal Database resource adapter
  - Allows for access to IMS resources and transaction management in a Java webserver

IMS Universal Drivers

What can these drivers do?

- Powers software such as Explore for Development, Explore for VsCode
- Create distributed connections to IMS databases
- Create Java Programs that can execute SQL and DL/I calls against IMS databases
- Create and edit field and segment information for IMS databases
- Create Java programs that run in IMS dependent regions (JMPs/JBPs)
- Read and write to IMS message queues
- And so much more!

IMS Universal Drivers

The IMS Universal Drivers have two connection types

- Type-4 Connections
  - For distributed programs, communication is handled over TCP/IP
  - These connections can browse metadata, execute SQL statements, can power webserver applications
- Type-2 Connections
  - For native programs running in a JMP or JBP.
  - Execute SQL and DLI statements, read/write to IMS message queues, and transactional programs

IMS – Java Interoperability in Three Ways

Accessing IMS with Java

There are three main application types that can access IMS resources

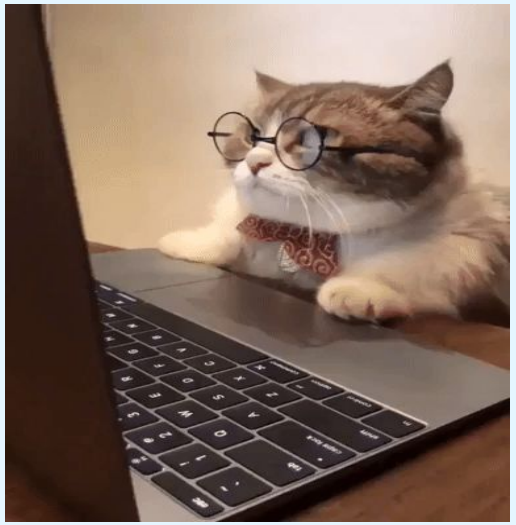
- Distributed program using Type-4 drivers
- Native batch program running in a JBP region
- Native message driven program running in a JMP region

Program #1

Get Parts Inventory
(Distributed, JDBC)

Meet our Mainframers!

Systems Programmer



Java Application Developer

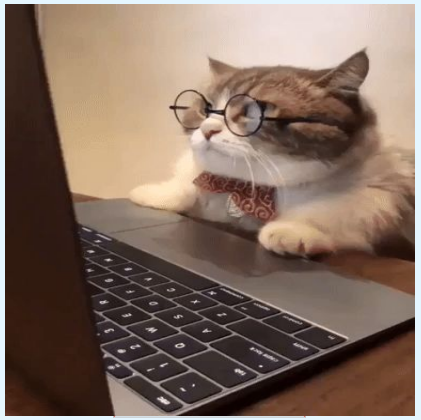
IMS JDBC

Systems Programmer

Requirements for this program to run:

- IMS Catalog and Managed ACBs (Recommended)
- Database metadata populated in the catalog (preferred)
- IMS Connect DRDA, ODBM

IMS JDBC



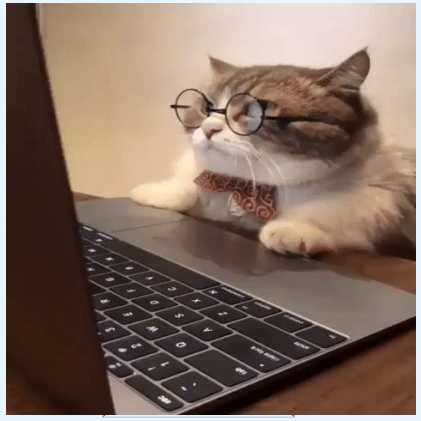
Java Application Developer

Creating your distributed Java program

- Java programs can use the IMS Universal Drivers to create a Type-4 (distributed) connection to an IMS database.
- The code snippet to the right shows what will need to be specified to create a connection.
- The scope of the connection will be to the PSB specified on the “setDatabaseName()” parameter.

```
IMSDataSource ds = new IMSDataSource();
ds.setHost(host: "sample.host.name");
ds.setPortNumber(portNumber: 5555);
ds.setDriverType(driverType: 4);
ds.setUser(user: "myUser");
ds.setPassword(password: "myPass");
ds.setDatabaseName(databaseName: "DI21PART");
connection = ds.getConnection();
```

IMS JDBC



Java Application Developer

IMS SQL

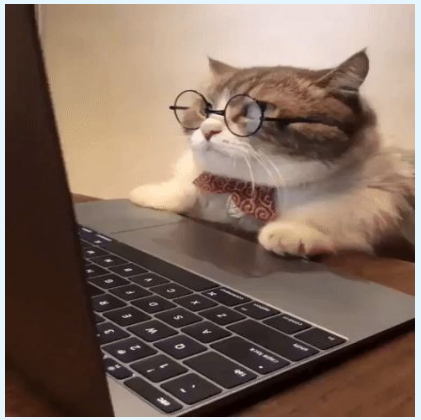
- IMS SQL allows anyone with database skills to easily work with IMS databases ... no DL/I skills required!
- Queries are easier to write and understand. Compare the query below to its DL/I equivalent.

```
st.executeQuery("SELECT * FROM DBPCB01.PARTROOT'");
```

```
DL/I translation for 'SELECT * FROM DBPCB01.PARTROOT' is:
GU   PARTROOT

(LOOP)
GN   PARTROOT
```

```
Displaying query results
PARTKEY: 02AN960C10
PARTNAME:          WAS
PARTDESC:       WASHER

PARTKEY: 02CK05CW181K
PARTNAME:          CAP
PARTDESC:       CAPACITOR
```

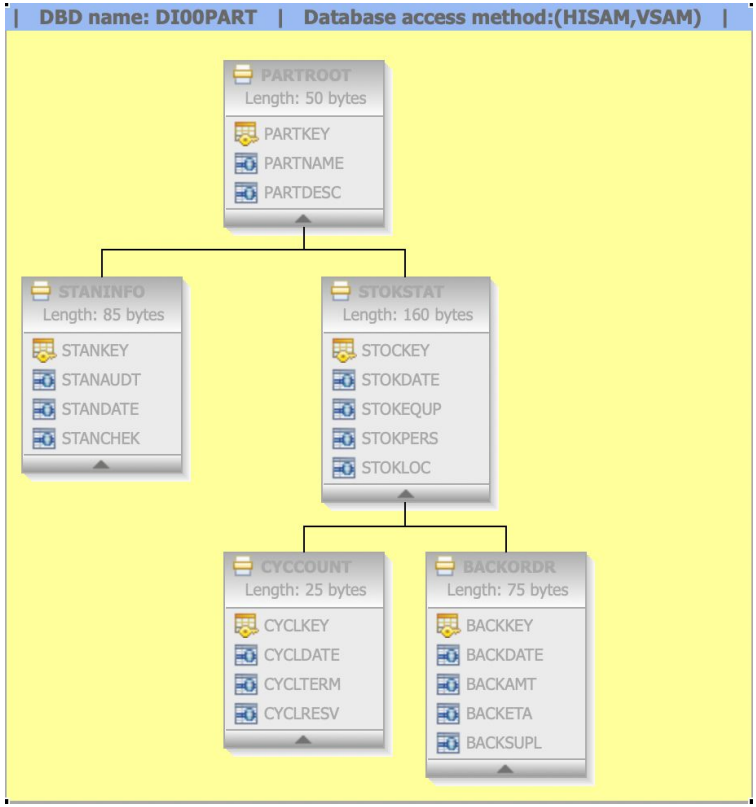
IMS JDBC



Java Application Developer

More IMS SQL!

- IMS SQL allows for Hierarchical databases to be queried like relational database
- The example query below shows how a hierarchical database can be “traversed” using relational logic

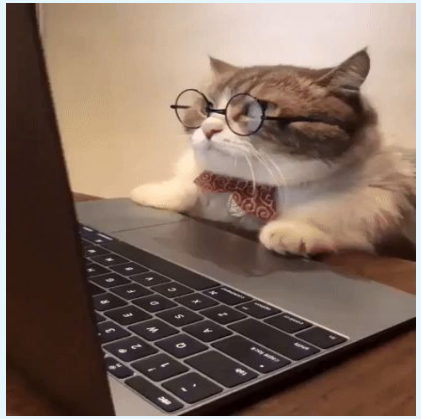



```
SELECT * FROM DBPCB01.BACKORDR WHERE PARTROOT_PARTKEY = '027618032P101'
and STOKSTAT_STOCKEY = '0025900326';
```

- IMS SQL also supports INSERT, UPDATE, and DELETE queries

```
"INSERT INTO DBPCB01.PARTROOT (PARTKEY, PARTNAME, PARTDESC) VALUES ('027618032P202', '', ' CAT 5 ETHERNET CABLE')";
"UPDATE DBPCB01.PARTROOT SET PARTDESC=' CAT 6 ETHERNET CABLE' WHERE PARTKEY='027618032P202'";
"DELETE FROM DBPCB01.PARTROOT WHERE PARTKEY='027618032P202'";
```

IMS JDBC



Java Application Developer

Program Complete!

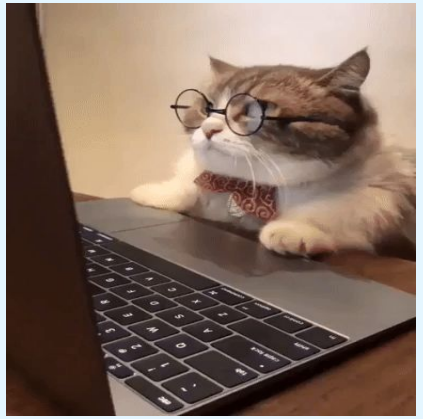
- The attached example is a simple program that creates a Type-4 (Distributed) connection to an IMS database and passes a simple SQL SELECT query against it.
- This is just a small glimpse into the wide variety of use cases that Java's IMS drivers are capable of

```java
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;
import com.ibm.ims.jdbc.IMSDataSource;

public class JBPLAB {
    Run | Debug
    public static void main(String[] args) {
        executeAndDisplaySqlQuery();
    }

    private static Connection (int driverType) throws Exception{
        Connection connection = null;
        if (driverType == 4) {
            IMSDataSource ds = new IMSDataSource();
            ds.setHost("sample.host.name");
            ds.setPortNumber(5555);
            ds.setDriverType(driverType);
            ds.setUser("myUser");
            ds.setPassword("myPass");
            ds.setDatabaseName("DI00PART");
            connection = ds.getConnection();

        } else if (driverType == 2) {
            // Establish a Type 2 Connection Here.
        }
        return connection;
    }

    private static void executeAndDisplaySqlQuery() throws Exception {
        Connection connection = createAnImsConnection(4);

        String sql = "SELECT * FROM PCB01.PARTROOT";

        Statement st = connection.createStatement();
        ResultSet rs = st.executeQuery(sql);

        ResultSetMetaData rsmd = rs.getMetaData();
        int colCount = rsmd.getColumnCount();

        System.out.println(x: "\nDisplaying query results");
        while (rs.next()) {
            for (int i = 1; i <= colCount; i++) {
                System.out.println(rsmd.getColumnName(i) + ": " + rs.getString(i));
            }
            System.out.println();
        }
    }
}
```

Program #2

Get Parts Inventory
(Native, JBP, JDBC)

IMS JBP



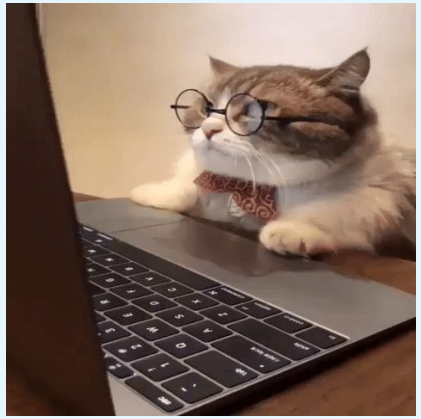
Java Application Developer

Creating JBPPARTS.jar

With a few minor changes, we can transform our program into a native application that runs in a JBP region

- The connection type will be changed to a Type-2 connection (Native) instead of a Type-4 connection.
- Before any ”work” can be done against the database, a Transaction object must be created.
- The ”createApplication()” function starts the workload on the JBP. Passing it to a Transaction object allows for greater control of the transaction through the Java program. This allows for manual abending, access to the Message Queue, and much more!

IMS JBP

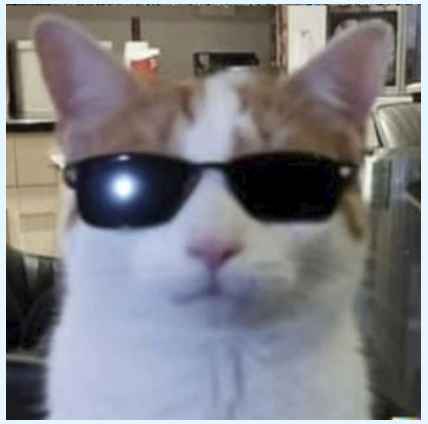


Java Application Developer

Program Complete!

```java
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.Statement;

public class JBPLAB {
    Run | Debug
    public static void main(String[] args) {
        executeNativeApplication();
    }

    private static Connection createAnImsConnection(int driverType) throws Exception{
        Connection connection = null;

        IMSDataSource ds = new IMSDataSource();
        ds.setDriverType(2);
        ds.setUser("myUser");
        ds.setPassword("myPassword");
        ds.setDatabaseName("DI00PART");
        connection = ds.getConnection();

        return connection;
    }

    private static void executeNativeApplication() throws Exception {
        // Write a native IMS JBP application
        Connection connection = createAnImsConnection(driverType: 2);

        // Start the unit of work
        Application app = ApplicationFactory.createApplication();
        Transaction transaction = app.getTransaction();

        // Do some work by displaying your inserted record
        String sql = "SELECT * FROM DBPCB01.PARTROOT";
        Statement st = connection.createStatement();
        ResultSet rs = st.executeQuery(sql);
        ResultSetMetaData rsmd = rs.getMetaData();
        int colCount = rsmd.getColumnCount();

        System.out.println(x: "\nDisplaying query results");
        while (rs.next()) {
            for (int i = 1; i <= colCount; i++) {
                System.out.println(rsmd.getColumnName(i) + ": " + rs.getString(i));
            }
            System.out.println();
        }
        // Commit your unit of work
        connection.close();
        app.end();
    }
```

IMS JBP



Systems Programmer

JBP Environment Requirements

- PSB and Program defined for the JBP region to use
  - ProgramType = Java
  - RegionType = JBP
- Program *.jar and relevant libraries in USS
- Configure Java Environment in Proclib
- Create a job to start JBP region

IMS JBP

Systems Programmer

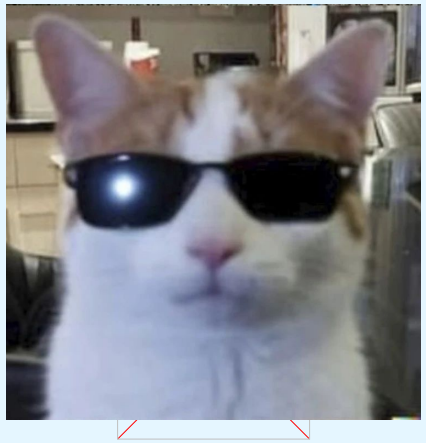
Configuring your Java Environment

DFSJVMEV:

This Proclib member specifies the JDK version and location of libT2DLI.so

```
****** ********************************* Top of Data ********************************
000001 ************************************************************************
000002 * Specify the location of Java native code (libT2DLI.so and
000003 * libT2DLI_64.so) and Java Virtual Machine (JVM) installation.
000004 *
000005 * Note: If you need to add another path, the previous line should have
000006 * ':' to separate the paths and '>' as the continuation character.
000007 ************************************************************************
000008 LIBPATH=>
000009 /usr/lpp/java/J17.0_64/bin:>
000010 /usr/lpp/java/J17.0_64/lib/j9vm:>
000011 /usr/lpp/java/J17.0_64/lib:>
000012 /shared/ims-work/lib
000013 *
****** ******************************** Bottom of Data ********************************
```

IMS JBP



Systems Programmer

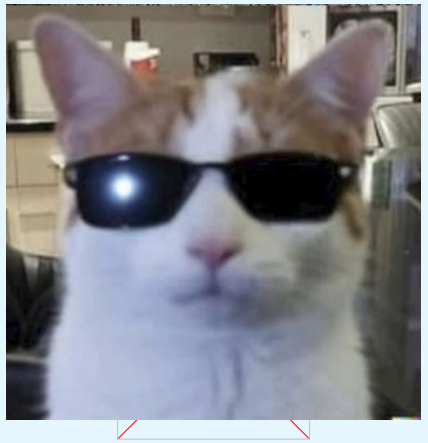
Configuring your Java Environment

DFSJVMMS:

This Proclib member sets up the Java Classpath and determines JVM Options

```
000001 ********************************************************************
000002 * Specify the profile that has environment settings and JVM options.
000003 * The following two JVM options are required.
000004 ********************************************************************
000005 -Djava.class.path=>
000006 /shared/ims-work/imsudb.jar:>
000007 /usr/lpp/ims/ims15.1/imsjava/imsutm.jar:>
000008 /u/dds3716/ims_imsd/lib/commons-io-2.4.jar:>
000009 /u/dds3716/ims_imsd/lib/commons-codec-1.15.jar:>
000010 /u/dds3716/ims_imsd/lib/commons-lang3-3.3.2.jar:>
000011 /u/dds3716/ims_imsd/lib/gson-2.5.jar:>
000012 /u/dds3716/ims_imsd/lib/ImsESConnectAPI.jar:>
000013 /shared/ims-work/URSOPRTS.jar:>
000014 /shared/ims/apps/jmpparts.jar:>
000015 /u/dds3716/ims_imsd/GetHotels.jar
000016 *
000017 ********************************************************************
000018 * JVM options for heap size tuning.
000019 ********************************************************************
000020 -Dibm.jvm.events.output=stdout
000021 -Xgcpolicy:gencon
```

IMS JBP



Systems Programmer

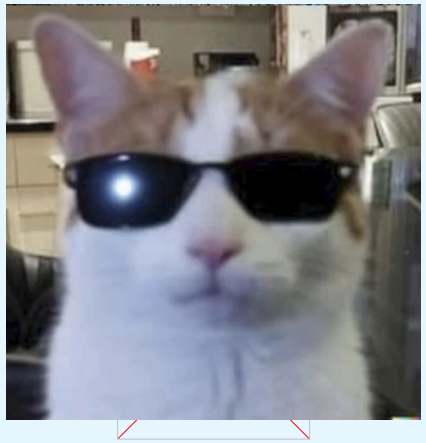
Configuring your Java Environment

DFSJVMAP:

This Proclib member maps your PSB to your main Java Class

```
**********************************************************************
* Test IMS JBP Program
**********************************************************************
JBPPARTS=com/ibm/ims/jbp/PartsInventory
*
```

IMS JBP



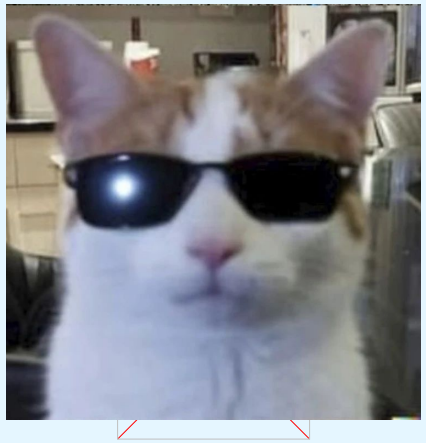
Systems Programmer

Start your JBP Region

- JBP Regions can be started by running a DFSJBP job
- This job determines various parameters and settings for your JBP region

```
//JBPJOB   JOB 1,IMS,MSGLEVEL=1,PRTY=11,CLASS=K,MSGCLASS=A,REGION=56K
//       EXEC DFSJBP,
//            IMSID=
//       PROC MBR=TEMPNAME,PSB=,JVMOPMAS=,OUT=,
//            OPT=N,SPIE=0,TEST=0,DIRCA=000,
//            STIMER=,CKPTID=,PARDLI=,
//            CPUTIME=,NBA=,OBA=,IMSID=,AGN=,
//            PREINIT=,RGN=56K,SOUT=A,
//            SYS2=,ALTID=,APARM=,ENVIRON=,LOCKMAX=,
//            PRLD=,SSM=,JVM=3164
//*
//JBPRGN EXEC PGM=DFSRRC00,REGION=&RGN,
//            PARM=(JBP,&MBR,&PSB,&JVMOPMAS,&OUT,
//            &OPT&SPIE&TEST&DIRCA,
//            &STIMER,&CKPTID,&PARDLI,&CPUTIME,
//            &NBA,&OBA,&IMSID,&AGN,
//            &PREINIT,&ALTID,
//            '&APARM',&ENVIRON,&LOCKMAX,
//            &PRLD,&SSM,&JVM)
//STEPLIB  DD DSN=IMS.&SYS2.SDFSJLIB,DISP=SHR
//         DD DSN=IMS.&SYS2.SDFSRESL,DISP=SHR
//         DD DSN=IMS.&SYS2.PGMLIB,DISP=SHR
//         DD DSN=CEE.SCEERUN,DISP=SHR
//         DD DSN=SYS1.CSSLIB,DISP=SHR
//PROCLIB  DD DSN=IMS.&SYS2.PROCLIB,DISP=SHR
//SYSUDUMP DD SYSOUT=&SOUT,
//         DCB=(LRECL=121,RECFM=VBA,BLKSIZE=3129),
//         SPACE=(125,(2500,100),RLSE,,ROUND)
```

IMS JBP



Systems Programmer

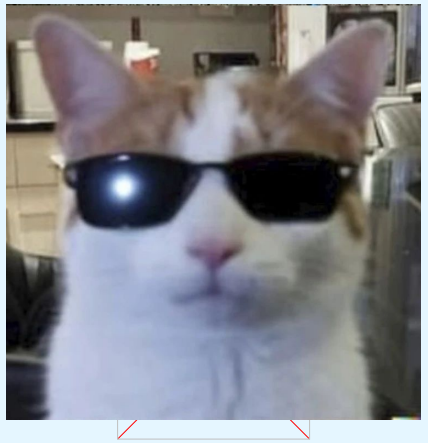
JBP Region Completed!

```
  Display  Filter  View  Print  Options  Search  H
 ------------------------------------------------
 SDSF OUTPUT DISPLAY DFSJBP1  JOB00083  DSID   106
 COMMAND INPUT ===> _
********************************* TOP OF DATA *****
Displaying query results
PARTKEY: 02AN960C10
PARTNAME:           WAS
PARTDESC:       WASHER
PARTKEY: 02CK05CW181K
PARTNAME:           CAP
PARTDESC:       CAPACITOR
PARTKEY: 02CSR13G104KL
PARTNAME:           KR1
PARTDESC:       KR1J50KS
PARTKEY: 02JAN1N976B
PARTNAME:           DIO
PARTDESC:       DIODE CODE-A
PARTKEY: 02MS16995-28
PARTNAME:           SCR
A      A
```

Program #3

Parts Inventory Management System (Native, JMP, JDBC, DLI, Message Driven)

IMS JMP



Systems Programmer

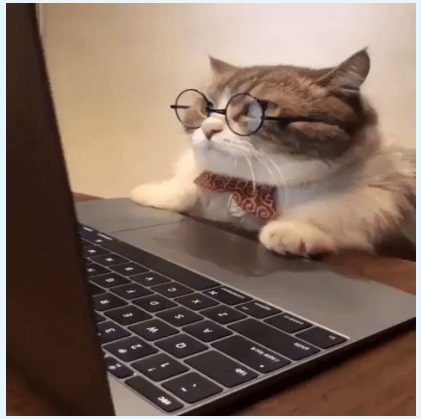
JMP Environment Requirements (*Similar* to JBP)

PSB, Program, and Transaction defined for the JMP region to use

- ProgramType = Java
- RegionType = JMP

- Program *.jar and relevant libraries in USS
- Configure Java Environment in Proclib
- Create a job to start JMP region

IMS JMP



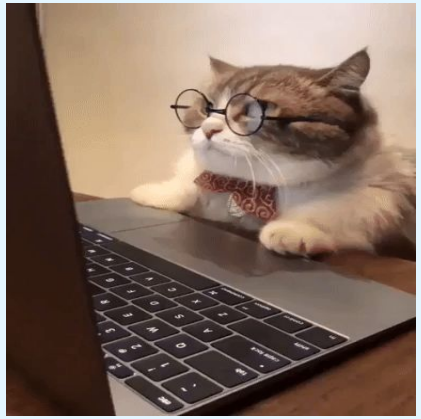
Java Application Developer

Creating a Message Driven Java Program

IMS Java Drivers provide support for reading from and placing output messages on IMS Message Queues

- Built in Message Queue objects can read messages from the queue when a JMP's Transaction code is called.
- Just like in the last example, JMP Programs can use the information passed into them to pass Java SQL calls against IMS Databases.
- Message Queues can handle and pass back the output messages created by the Java program.

IMS JMP



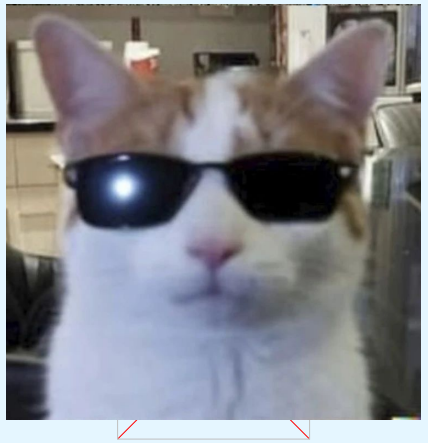
Java Application Developer

Program Complete!

- Attached is a screenshot of the program's main method
- The code shown reads from the Message Queue to build a SQL statement to execute against our database

```
public static void main(String[] args) {
try {
    inputMessage = app.getIOMessage(url: "class://com.ibm.ims.jmp.message.PartInfoInput");
    outputMessage = app.getIOMessage(url: "class://com.ibm.ims.jmp.message.PartInfoOutput");
    if (!foundError) {
        while (messageQueue.getUnique(inputMessage)) {
            action = inputMessage.getString(fieldName: "IN_COMMAND").trim();
            partKey = inputMessage.getString(fieldName: "PARTKEY").trim();

            if (action.equalsIgnoreCase(anotherString: "ADD") || action.equalsIgnoreCase(anotherString: "UPD")) {
                partName = inputMessage.getString(fieldName: "PARTNAME").trim();
                partDesc = inputMessage.getString(fieldName: "PARTDESC").trim();
            }

            PartInformation part = new PartInformation();
            part.setPartKey(partKey);
            part.setPartName(partName);
            part.setPartDesc(partDesc);
            PartInventoryService partService = new PartInventoryService(imsConnection);

            if (partService != null) {
                String returnMessage = "";
                if (action.equalsIgnoreCase(anotherString: "ADD")) {
                    returnMessage = partService.addPart(part);
                }
                else if (action.equalsIgnoreCase(anotherString: "DEL")) {
                    returnMessage = partService.deletePart(part.getPartKey());
                }
                else if (action.equalsIgnoreCase(anotherString: "UPD")) {
                    returnMessage = partService.updatePart(part);
                }
                else if (action.equalsIgnoreCase(anotherString: "GET")) {
                    returnMessage = partService.getContact(part.getPartKey());
                }
                else {
                    returnMessage = "ERROR: INVALID COMMAND WAS SPECIFIED";

                }
                    outputMessage.setString(fieldName: "RETURN_MSG", returnMessage);
                    messageQueue.insert(outputMessage, MessageQueue.DEFAULT_DESTINATION);
                    tran.commit();

            }
```

IMS JMP



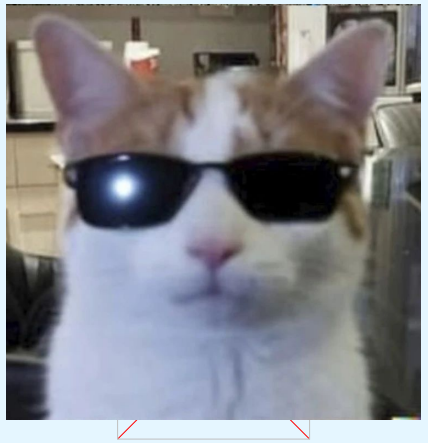
Systems Programmer

Deploy the JMP!

- This time you will need to create a DFSJMP job and place it in your IMS Region's Jobs dataset
- Instead of submitting this job you will need to start the region by passing a command to start the region to IMS
  - ‘/r xx , /START REGION DFSJMP’
- To interact with your program, sign on to IMS and invoke your transaction

```
RUNPARTS ADD ANC09236873  CAT 5 ETHERNET
```

IMS JMP



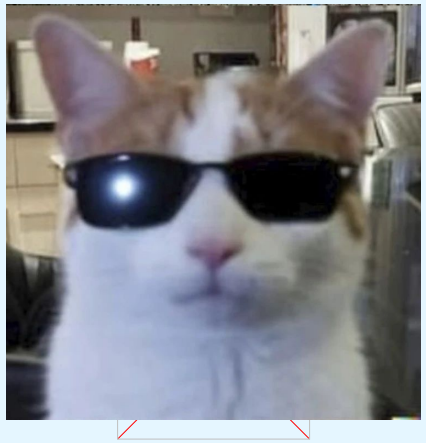
Systems Programmer

Deploy the JMP!

- This time you will need to create a DFSJMP job and place it in your IMS Region's Jobs dataset
- Instead of submitting this job you will need to start the region by passing a command to start the region to IMS
  - ‘/r xx , /START REGION DFSJMP’
- To interact with your program, sign on to IMS and invoke your transaction

```
RUNPARTS ADD ANC09236873  CAT 5 ETHERNET
```

IMS JMP



Systems Programmer

Program Complete!

```
SDSF OUTPUT DISPLAY DFSJMP1  JOB00090  DSID   103 LINE 0       COLUMNS 02- 81
COMMAND INPUT ===> _                                        SCROLL ===> CSR
******************************** TOP OF DATA ********************************
2026-05-13 14:17:56.786 Parts Inventory (Version 3.0 CATLG) application called,
2026-05-13 14:18:14.931 SQL: INSERT INTO DBPCB01.PARTROOT (PARTKEY, PARTNAME, P
ETHERNET')
2026-05-13 14:18:15.303 INFO: PART WITH PARTKEY 027618032P404 WAS SUCCESSFULLY
2026-05-13 14:18:24.74 Parts Inventory (Version 3.0 CATLG) application called,
2026-05-13 14:18:25.284 SQL: SELECT * FROM DBPCB01.PARTROOT WHERE PARTKEY = '02
2026-05-13 14:18:25.37 PARTKEY: 027618032P404    , PARTNAME: CAT 5 ETHERN, PART
2026-05-13 14:19:07.781 Parts Inventory (Version 3.0 CATLG) application called,
2026-05-13 14:19:07.984 SQL: UPDATE DBPCB01.PARTROOT SET PARTNAME = 'CAT6  ETHE
2P404'
2026-05-13 14:19:07.993 INFO: PART WITH PARTKEY 027618032P404 WAS SUCCESSFULLY
2026-05-13 14:19:15.432 Parts Inventory (Version 3.0 CATLG) application called,
```

Summary

The IMS Universal Drivers are incredibly capable and can power a wide variety of utilities, programs, and automations

Type 4 Programs

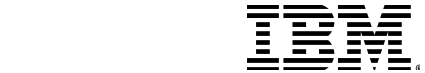
- Type-4 programs allow for distributed connections to IMS databases and for writing and executing IMS SQL queries

Type 2 Programs

- Type-2 programs run native to IMS and have even more capabilities
- Read/Write to IMS message queues, DLI calls, SQL calls, Transactional programs, Webserver applications, and more!

Questions?
Comments?

Thank you!




IBM