IBM i Toolbox for Java (JTOpen)

Main package **com.ibm.as400.access**

Contains classes for communication to IBM i

Classes use TCP sockets in "host servers"

Host servers

STRHOSTSVR SERVER(*ALL) Start Host Server

WRKSRVTBLE Work with Service Table Entry

Service	**Description**	**Port**
as-central	Central	8470
as-database	Database	8471
as-file	File server	8473
as-rmtcmd	Command/program call	8475
. . .		

Common classes

AS400 – authentication info and connections
to IBM i host servers

AS400xxx – communication and helper classes

Classes use path strings, e.g.

/QSYS.LIB/MYLIB.LIB/MYFILE.FILE/MYMBR.MBR

Data area

DataArea – abstract class

CharacterDataArea,

DecimalDataArea,

LogicalDataArea,

LocalDataArea (*LDA)

Data queue

Base**DataQueue** – abstract class

DataQueue – FIFO/LIFO

KeyedDataQueue

FTP and SaveFile

AS400**FTP** – extends FTP class, does extra steps

(copy save files, check authority)

SaveFile – create, clear, save, …

FTP get

```
AS400 remoteServer = new AS400(host, userName);
AS400FTP ftp = new AS400FTP(remoteServer);
try {
    // Binary data transfer
    ftp.setDataTransferType(AS400FTP.BINARY);

    // FTP get command
    ftp.get(saveFilePathString, pcPathString);
} catch (Exception exc) {
    exc.printStackTrace();
}
```

FTP put

```
try {
    // Not necessary if PC file ends with .savf
    ftp.setDataTransferType(AS400FTP.BINARY);

    // FTP put command
    ftp.put(pcPathString, saveFilePathString);
} catch (Exception exc) {
    exc.printStackTrace();
}
```

Save file

```
String libFrom = "VZTOOL2";
String libTo = "VZTOOL3";
String savfName = "SAVE_FILE";

// Create SaveFile Java object
SaveFile saveFile = new SaveFile(remoteServer, libTo, savfName);

// Create the IBM i save file object in the library libTo
if (!saveFile.exists()) {
    saveFile.create();
}

// Save the library to the save file (with default parameters)
saveFile.save(libFrom);
```

IFS

IFS classes – superset of **java.io** (not **java.nio**)

IFSFile – create and manipulate directories and files

IFSFileInputStream, IFSFileOutputStream

IFSFileReader, IFSFileWriter

IFS

```
IFSFileOutputStream(AS400 remoteServer, String pathString,
         int shareOption, boolean append, int ccsid)
```

pathString – e. g. "/home/vzupka/QRPGLESRC/TESTRPG.MBR"

shareOption – Indicates how users can access the file.

• SHARE_ALL	Share access with readers and writers
• SHARE_NONE	Share access with none
• SHARE_READERS	Share access with readers
• SHARE_WRITERS	Share access with writers

Output IFS stream file

```
ByteBuffer buf = ByteBuffer.allocate(1000000);
Path pcFilePath = Paths.get(pcFilePathString);
// Open output IFS file - SHARED for all users, REWRITE data, CCSID
IFSFileOutputStream ifsOutStream =  new IFSFileOutputStream(
                                remoteServer, ifsFilePathString,
                                IFSFileOutputStream.SHARE_ALL, false, CCSID);
FileChannel fileChannel = (FileChannel) Files.newByteChannel(pcFilePath);
// Copy bytes using buffer
int bytesRead = fileChannel.read(buf);
while (bytesRead > 0) {
    // Write buffer to output file
    ifsOutStream.write(buf.array(), 0, bytesRead);
    buf.rewind(); // Set start of buffer
    bytesRead = fileChannel.read(buf);
}
```

Command call

```
// Enable calling CL commands
CommandCall cmdCall = new CommandCall(remoteServer);

// Clear physical file member - build command text
// CLRPFM FILE(VZTOOL3/QRPGLESRC) MBR(TESTRPG)
String clrPfmCommand = "CLRPFM FILE(" + libraryName + "/"
                                    + fileName + ") MBR(" + memberName + “)";
// Run the command
cmdCall.run(clrPfmCommand);
```

Command call messages

```
// Get list of messages from the command
AS400Message[] messagelist = cmdCall.getMessageList();

// Create array to get information from messages
String[] strArr = new String[messagelist.length];

if (messagelist.length > 0) {
    // Get all messages in the array
    for (int idx = 0; idx < messagelist.length; idx++) {
        strArr[idx] = messagelist[idx].getID() + "  "
                    + messagelist[idx].getText();
    }
}
```

```
CPC3101  Member TESTPROG file QRPGLESRC in VZTOOL3 cleared.
```

Spooled files

SpooledFileList – get filtered list of spooled files

SpooledFile – get attributes

Interpret control sequences programmatically

06 – Required New Line (RNL)
0b – Vertical Tab (VT)
0c – Form Feed (FF)
15 – New Line (NL)
25 – Line Feed/Index (LF/INX)
. . .
34c0nn – Absolute Horizontal Presentation Position (AHPP) nn
34c8nn – Relative Horizontal Presentation Position (RHPP) nn
. . .

Spooled files

```
SpooledFileList splfList = new SpooledFileList(remoteServer);
Enumeration<SpooledFile> spooledFiles = splfList.getObjects();

SpooledFile splf = (SpooledFile) spooledFiles.nextElement();

PrintParameterList printParameterList = new PrintParameterList();
printParameterList.setParameter(SpooledFile.ATTR_SPOOLFILE, namePar);
. . .
InputStream inputStream = splf.getInputStream(printParameterList);
int bytesRead = inputStream.read(inputBuffer);
while (bytesRead != -1) {
    for (int idx = 0; idx < bytesRead; idx++)
        byte byteRead = inputBuffer[idx];
        . . . // Interpret the byteRead
    }
}
```

Data types

Toolbox	**Java**	**RPG**
AS400**Text**	String	CHAR
AS400**Bin2**	Short	INT(5)
AS400**Bin4**	Integer	INT(10)
AS400**ByteArray**	byte[n]	CHAR(1) DIM(n)
AS400**Float4**	Float	FLOAT(4)
AS400**Float8**	Double	FLOAT(8)
AS400**PackedDecimal**	BigDecimal	PACKED(n [: d])
AS400**UnsignedBin2**	Integer	UNS(5)
AS400**UnsignedBin4**	Long	UNS(10)
AS400**ZonedDecimal**	BigDecimal	ZONED(n [: d])

String ↔ AS400Text

```
byte[] bytes;

// Create text converter with length in bytes and CCSID
AS400Text as400text = new AS400Text(8, 870);

// Convert string to byte array
bytes = as400text.toBytes("1234ABCŘ");

// Result in hex: f1 f2 f3 f4 c1 c2 c3 ae

// Convert byte array back to string
String string = (String) as400text.toObject(bytes);
```

Integer ↔ AS400Bin4

```
// Create converter with length in bytes
AS400Bin4 as400Bin4 = new AS400Bin4();

// Convert string to byte array
bytes = as400Bin4.toBytes(new Integer("999999999"));

// Result in hex: 3b 9a c9 ff

// Convert byte array back to Integer
Integer integerNumber = (Integer) as400Bin4.toObject(bytes);
```

Record access files

SequentialFile – record access

KeyedFile – record access

AS400RecordDescription – retrieve record formats

RecordFormat – set record format to a file

Record – read/write

FieldDescription

Sequential input file

```
// Create an AS400FileRecordDescription object
AS400FileRecordDescription inRecDesc =
                             new AS400FileRecordDescription(remoteServer, as400PathString);
// Get list of record formats of the database file
RecordFormat[] formats = inRecDesc.retrieveRecordFormat();

// Create a SequentialFile object that represents the file
SequentialFile as400seqFile = new SequentialFile(remoteServer, as400PathString);

// Set the record format (the only one)
as400seqFile.setRecordFormat(formats[0]);

// Open the source physical file member as a sequential file
as400seqFile.open(AS400File.READ_ONLY, 0,
                              AS400File.COMMIT_LOCK_LEVEL_NONE);
. . .
```

```
// Read the first source member record
Record inRecord = as400seqFile.readNext();
while (inRecord != null) {
    StringBuilder textLine = new StringBuilder();

    byte[] bytes = inRecord.getFieldAsBytes("SRCDTA");

    // Convert data using from AS400Text to String
    AS400Text textConverter = new AS400Text( bytes.length, CCSID, remoteServer);
    String str = (String) textConverter.toObject(bytes);

    // Translate the String to PC charset
    String translatedData = new String(str.getBytes(pcCharset), pcCharset);
    textLine.append(translatedData).append("\n");

    pcOutFile.write(textLine.toString()); // Write line to the PC file
    inRecord = as400seqFile.readNext(); // Read next source member record
}
as400seqFile.close();
pcOutFile.close();
```

Get fields from record

```
DecimalFormat df1 = new DecimalFormat("0000.00");
DecimalFormat df2 = new DecimalFormat("000000");

// Sequential number field edited (with the dot)
String seq = df1.format((Number) inRecord.getField("SRCSEQ"));
// 4 digits + 2 decimals (remove the dot)
String seq2 = seq.substring(0, 4) + seq.substring(5);

// Date field
String srcDat = df2.format((Number) inRecord.getField("SRCDAT"));

// Data line field
String srcData = (String) inRecord.getField("SRCDTA");
```

Program call

```
// Create ProgramCall object
ProgramCall program = new ProgramCall(remoteServer);

// Initialize the name of the program to run
String programName = "/QSYS.LIB/VZTOOL.LIB/TESTPROG.PGM";

// Set up 3 parameters
ProgramParameter[] parameterList = new ProgramParameter[3];
```

Program call

```
// First parameter (input) sends a name text.
AS400Text nametext = new AS400Text(8);
parameterList[0] = new ProgramParameter(nametext.toBytes("My name”));

// Second parameter (output) gets the answer, up to 50 bytes long.
parameterList[1] = new ProgramParameter(50);

// Third parameter (in-out) sends a quantity and gets a value up to 30 bytes long.
short shortNbr = 300;
// Convert short number into byte array
AS400Bin2 bin2Converter = new AS400Bin2();
byte[] quantity = bin2Converter.toBytes(shortNbr);

// Define the third parameter as a structure with two subfields with overlay
parameterList[2] = new ProgramParameter(quantity, 30);
```

Program call

```
// Set the program name and parameter list
program.setProgram(programName, parameterList);

// Run the program
if (program.run() != true) {
   // Show the messages
    AS400Message[] messagelist = program.getMessageList();
    . . .
}
// Else no error, get output data
else {
    AS400Text text = new AS400Text(50);
    String param2 = (String) text.toObject(parameterList[1].getOutputData());
    text = new AS400Text(30);
    String param3 = (String) text.toObject(parameterList[2].getOutputData());
}
```

RPG program

```
**free
// main program called from Java or from RPG with 3 parameters
dcl-ds parameter3  template;   // data structure template for parameter 3
    outputParam char(30);   // returned value
    quantity  int(5) overlay(outputParam);  // short integer overlays characters
end-ds;
dcl-pi *N;   // program interface
    nametext char(8);   // input parameter
    answer char(50);    // output parameter
    outPar likeds(parameter3);   // in-out parameter
end-pi;
dcl-s padding char(100) inz(*all'-');

answer = 'ECHO' + %char(outPar.quantity); // to output parameter
outPar.outputParam = 'ECHO' + nametext;  // to output parameter

return;
```

Result of program call

```
ECHO 300------------------------------------------ 50 characters
ECHO My name ----------------- 30 characters
```

SQL – Connect database

```
Class.forName("com.ibm.as400.access.AS400JDBCDriver");

// Connection properties
Properties conprop = new Properties();
conprop.put("user", userName);
conprop.put("naming", "sql");
conprop.put("decimal separator", ".");
conprop.put("sort language", "CSY");
conprop.put("sort", "language");
conprop.put("libraries", library); // default schema

// Set login timeout in seconds
DriverManager.setLoginTimeout(5);

// DriverManager gets connection object for JDBC
Connection conn =
        DriverManager.getConnection("jdbc:as400://" + host, conprop);
conn.setAutoCommit(true);
. . .
```

SQL SELECT and result set

```
stmtText = "SELECT  *  FROM  PFJE_JC" + typeCode
                                            + " ORDER BY  SEQNBR";
try {
    Statement stmt = conn.createStatement();
    ResultSet rs = stmt.executeQuery(stmtText);

    while (rs.next()) {
        Integer seqnbrInt = rs.getInt(1);
        String type = rs.getString(2);
        . . .
    }
} catch (SQLException sqle) {
    msgValue = sqle.toString() + "\n";
    System.out.println("SQL or I/O Error: "+ msgValue);
}
```

Data types

Toolbox	Java	COBOL
AS400Text	String	PIC(X)
AS400Bin2	Short	COMP-4/BINARY PICTURE S9(5) with NOSTDTRUNC
AS400Bin4	Integer	COMP-4/BINARY PICTURE S9(10) with NOSTDTRUNC
AS400ByteArray	byte[n]	PIC X OCCURS(n)
AS400Float4	Float	COMP-1
AS400Float8	Double	COMP-2
AS400PackedDecimal	BigDecimal	COMP-3/PACKED-DECIMAL PIC S9(n-d)[Vd]
AS400UnsignedBin2	Integer	PIC 9(4) COMP-5
AS400UnsignedBin4	Long	PIC 9(9) COMP-5
AS400ZonedDecimal	BigDecimal	PIC S9(n) USAGE DISPLAY

BigDecimal ↔ AS400PackedDecimal

```
// Create converter with length in bytes
AS400PackedDecimal as400PackedDecimal =
                                        new AS400PackedDecimal(16, 2);
// Convert BigDecimal to byte array
bytes = as400PackedDecimal.toBytes(
                                        new BigDecimal("99999999999999.99"));

// Result in hex: 09 99 99 99 99 99 99 99 9f

// Convert byte array back to BigDecimal
BigDecimal bigDecimal
                          = (BigDecimal) as400PackedDecimal.toObject(bytes);
```

Connecting to file server

```
AS400 remoteServer = new AS400(host, userName);
try {
    // Connect FILE service of the IBM i server in advance
    remoteServer.connectService(AS400.FILE);
} catch (Exception exc) {
    exc.printStackTrace();
}
```