



# Toolbox for Java<sup>TM</sup>: Advanced

*IBM @server iSeries 400 and AS/400e*

**Clifton Nock**

© Copyright IBM Corporation, 2000. All Rights Reserved.  
This publication may refer to products that are not currently  
available in your country. IBM makes no commitment to  
make available any products referred to herein.

**IBM @server. For the next generation of e-business.**

# *AS/400 Toolbox for Java: Advanced*

---

**This publication may refer to products that are not available in your country.**

**IBM makes no commitment to make available any products referred to herein.**

**The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:**

AIX	AS/400	DB2
IBM	OS/400	

**Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.**

**Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.**

***Other company, product or service names may be the trademarks or service marks of others.***

# *Table of contents*

---

## ***Table of Contents***

**Introduction**

**The AS400 object**

**Connection pooling**

**Proxy support**

**Component list**

**JDBC**

**Using the Toolbox on the AS/400**

**Record-level database access**

**Command call and program call**

**Program Call Markup Language (PCML)**

**HTML and servlet classes**

**JTOpen (Open source)**

**Resource framework**

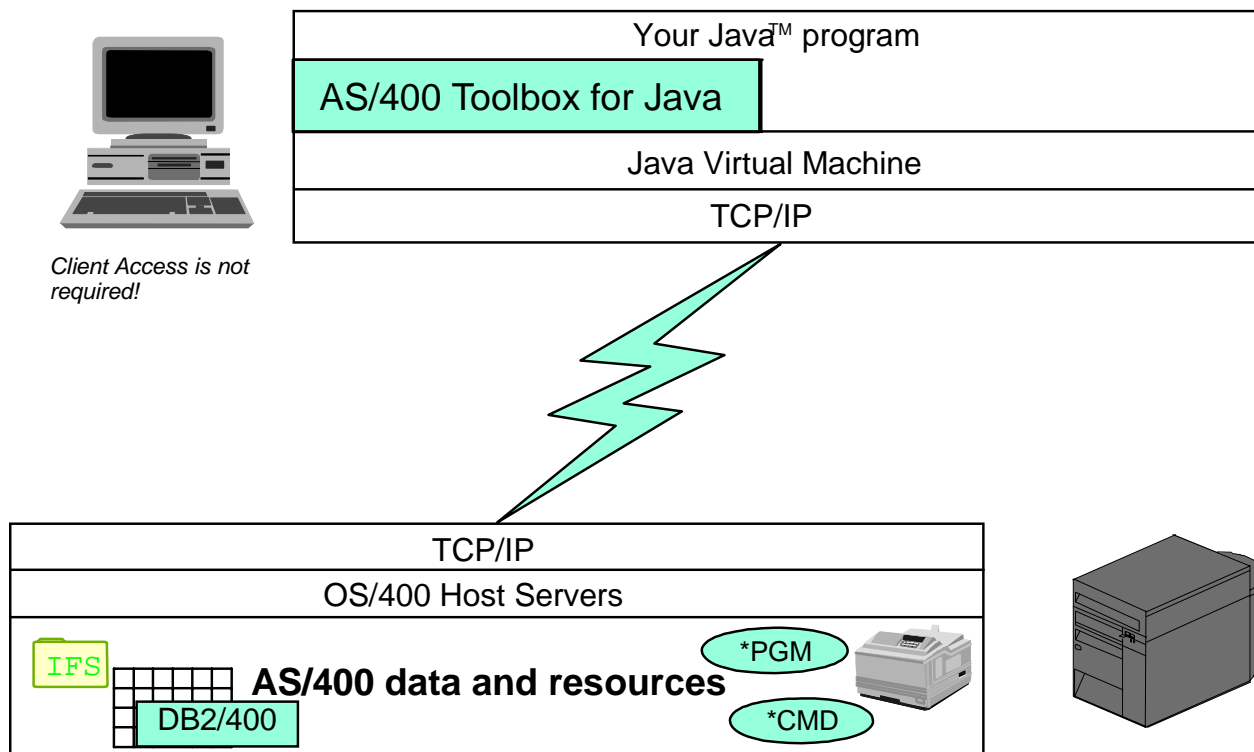
**JarMaker**

**References**

**AS/400e**

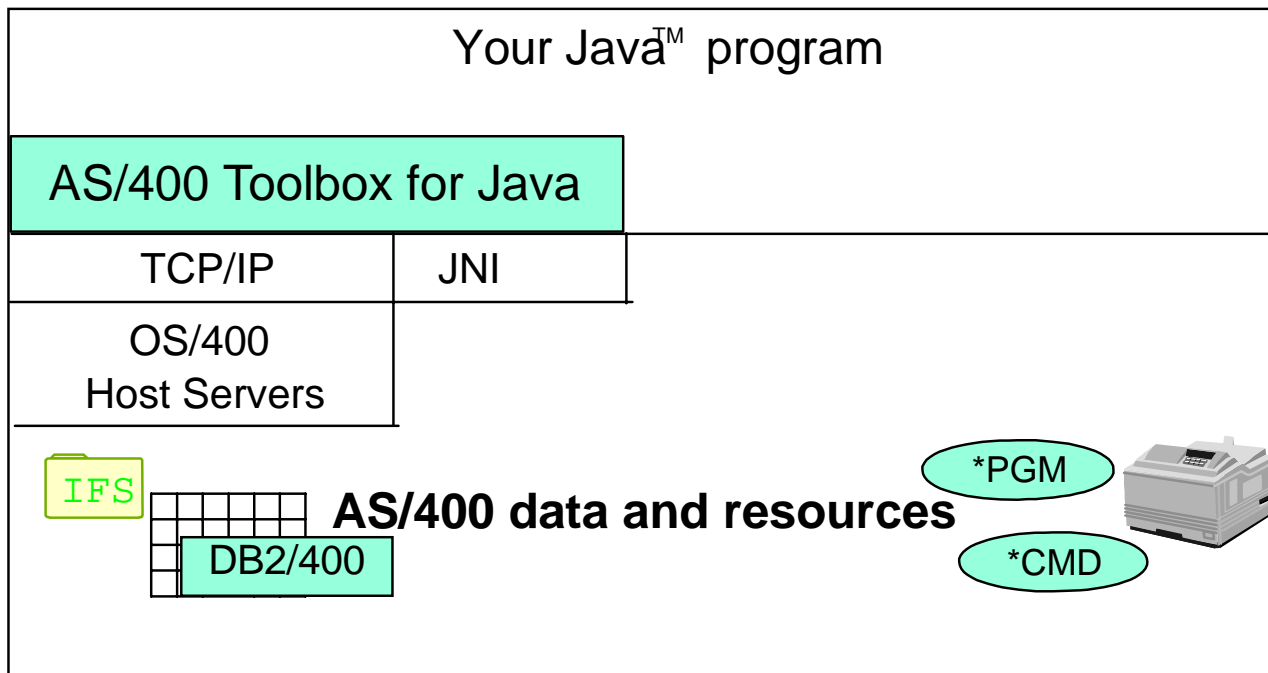
# AS/400 Toolbox for Java

- The Toolbox is a set of Java classes and utilities which provide access to AS/400 data and resources
- Useful in client/server environments - any Java client!
  - Java client application
  - Java applet (in browser)
  - Java servlet - communicating with an AS/400 from another web server



# AS/400 Toolbox for Java

- Since it is pure Java, the Toolbox also runs on the AS/400 Java Virtual Machine
- Toolbox runs optimized on the AS/400 - makes direct API calls using JNI
  - Your code is the same - the Toolbox selects its own implementation based on whether it is running on an AS/400 or not
- Useful in server environments - any Java server
  - Server to a client application
  - Server application
  - Java servlet - running on an AS/400



AS/400e

# *Toolbox on an AS/400*

## **Considerations for running the Toolbox on the AS/400:**

### **API set to use:**

- Native JDBC driver vs Toolbox JDBC driver
- java.io.File vs IFSFile
  
- Portability vs complexity
  - JNI vs ProgramCall / CommandCall

### **CRTJVAPGM on Toolbox file**

- jt400.zip/jt400.jar or jt400Access.zip or jt400Native.jar

### **AS400 Object can use current job's user ID and password**

- When Java program and data on same AS/400
- When Java program on one AS/400 and data is on another AS/400

### **Many Toolbox components can stay in the current job using AS/400 API calls instead of a server job.**

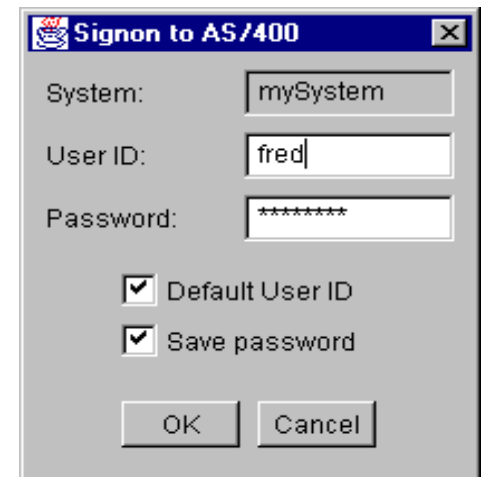
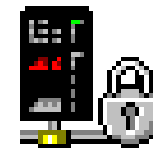
- Other functions still use server job
- CommandCall and ProgramCall do this conditionally
  - based on whether the command or program is threadsafe
  - see the setThreadSafe() method (*JTOpen 2.0 only*)

# *The AS400 object*

**Represents a connection to the AS/400**

**Encapsulates security/identity**

- Password caching available
- Establish a default UserID
- Sign-on GUI if UserID/password not supplied by application
- Change password GUI when appropriate
- **Provides Secure Sockets Layer (SSL) communication**
  - Encryption and server authentication



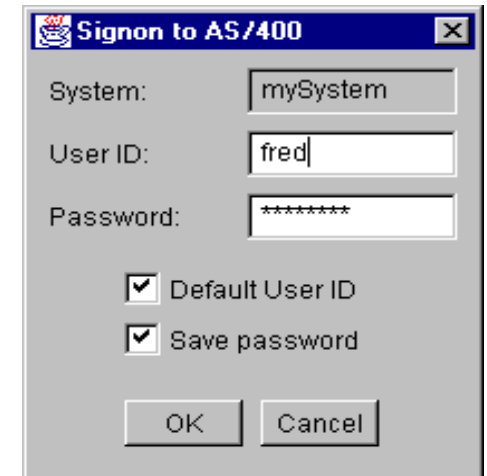
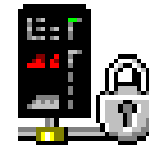
***Most Toolbox classes use the AS400 object***

# The AS400 object

## When running on the AS/400, Toolbox can pick up current job's user ID and password

- Use default constructor or \*CURRENT

```
new AS400();  
new AS400("localhost", "*CURRENT", "*CURRENT");
```



## Represents a connection to the AS/400

- Single vs multiple identities
- Single vs multiple connections
- Implicit vs explicit connection

```
AS400 sys  = new AS400();           // if on client, will prompt for system, uid, pwd  
AS400 sys2 = new AS400("mySystem"); // if on client, will prompt for uid, pwd  
AS400 sys3 = new AS400("mySystem", "uid1", "pwd1");  
AS400 sys4 = new AS400("mySystem", "uid2", "pwd2");
```

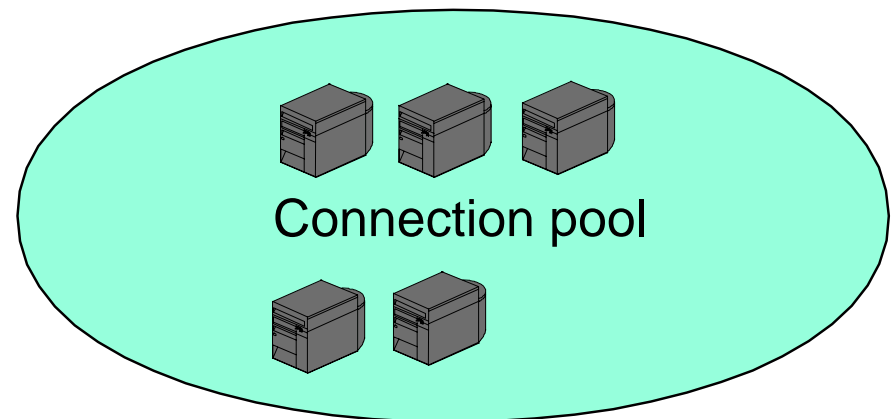
```
CommandCall cc = new CommandCall(sys); // cc and cc2 will share a connection  
CommandCall cc2 = new CommandCall(sys);  
CommandCall cc3 = new CommandCall(sys3); // cc3 and cc4 tasks will go against  
CommandCall cc4 = new CommandCall(sys4); // different profiles
```



# Connection pooling

Connection pooling can help performance *significantly*

- Connecting to the AS/400 is an expensive operation
- Pooling means reusing AS400 objects - i.e. keeping the connection open for later
- Saves frequent disconnects and reconnects
- Common scenario: servlets
  - Without pooling: Create a new AS400 object for each invocation of the servlet
  - With pooling: Grab a preconnected AS400 object from the pool for each invocation of the servlet, return it when done!
- Connections will be added as needed



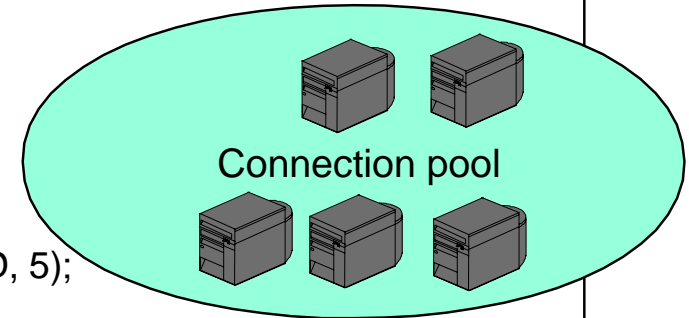
Available only  
in JTOpen 2.0

AS/400e

# Connection pooling

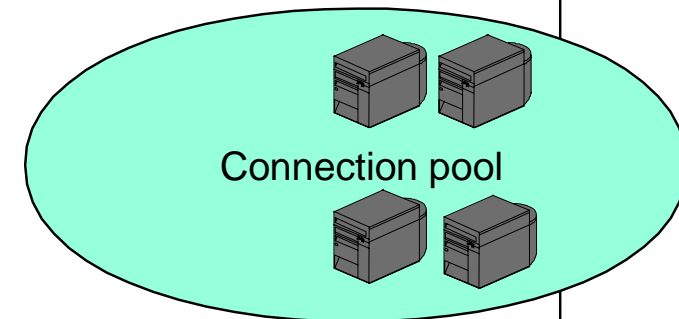
## Set up the connection pool

```
AS400ConnectionPool pool = new AS400ConnectionPool();  
pool.setMaxConnections(128);  
  
// Preconnect 5 connections to the AS400.COMMAND service.  
pool.fill("myAS400", "myUserID", "myPassword", AS400.COMMAND, 5);
```



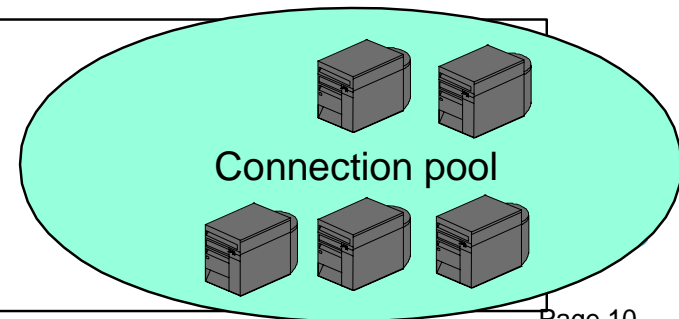
## Use a connection from the pool

```
AS400 connection = pool.getConnection("myAS400", "myUserID",  
                                     "myPassword", AS400.COMMAND);  
  
CommandCall cmd = new CommandCall(connection);  
cmd.run("CRTLIB FRED");
```



## Return it to the pool when done

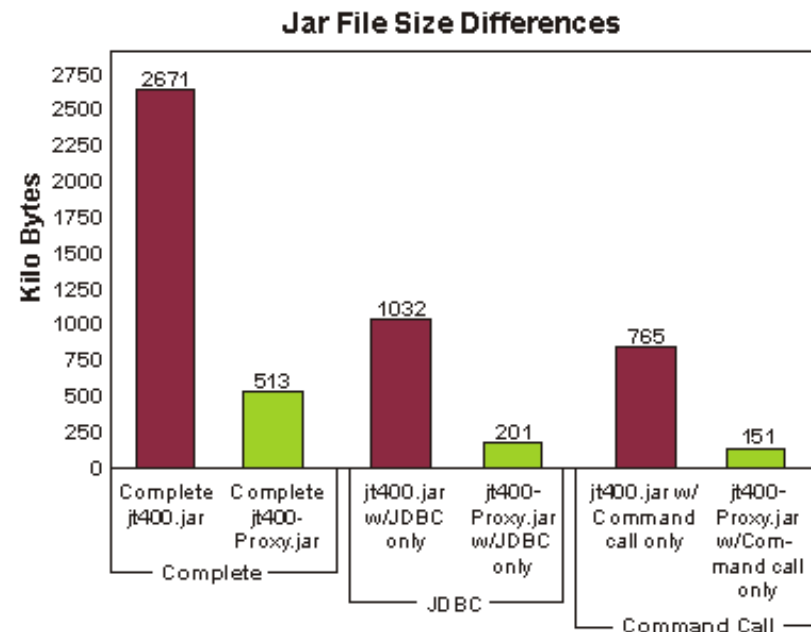
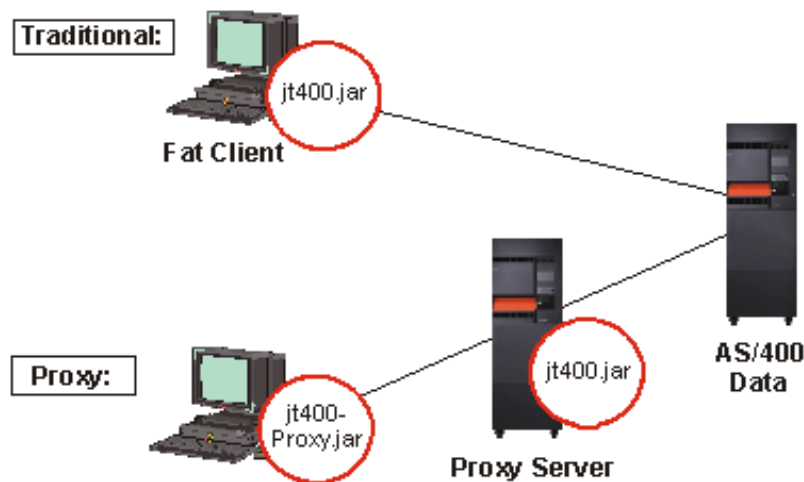
```
pool.returnConnectionToPool(connection);
```



# Proxy support

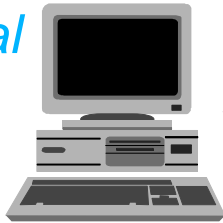
## Proxy support is a three-tier solution:

- Set of lightweight classes runs on client
- Full set of Toolbox classes runs on middle-tier
- Useful for:
  - Applets - the smaller the jar file, the faster the download
  - Firewalls - only need to open up 1 port



# Proxy support

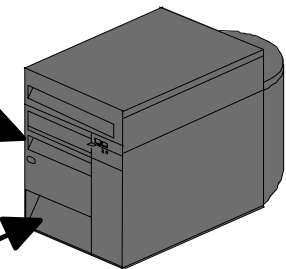
*Traditional client*



```
java -classpath jt400.jar MyApp
```

*No change to application -  
Same program can run in  
traditional mode or proxy mode  
just by changing a system  
property*

*Data server*



*Proxy client*



```
java -classpath jt400.jar  
com.ibm.as400.access.ProxyServer
```

*Proxy server*

*(this does not have to  
be an AS/400 - it can  
be any Java machine)*

```
java -classpath jt400Proxy.jar  
-Dcom.ibm.as400.access.AS400.proxyServer=pserver  
MyApp
```

**AS/400e**

# Component list (part 1)

<b>Component</b>	<b>Toolbox includes...</b>	<b>Native optimization</b>	<b>Proxy support</b>
AS400 object	●	●	●
AS400JPing/JPing	● <i>JTOpen 2.0</i>		
Authentication ( <i>com.ibm.as400.security.auth</i> package)	●	●	
Command call	●		●
Connection pool	● <i>JTOpen 2.0</i>		
Data area	●		●
Data conversion	●		●
Data description	●		●
Data queue	●	●	●
Digital cerificate	●	●	
Environment variable	● <i>JTOpen 2.0</i>		
File Transfer Protocol (FTP)	●		
Graphical Toolbox ( <i>com.ibm.as400.ui.*</i> package)	●		
GUI classes ( <i>com.ibm.as400.vaccess</i> package)	●		
HTML classes ( <i>com.ibm.as400.util.html</i> package)	●		

- **Components in *com.ibm.as400.access* package unless otherwise noted**

# Component list (part 2)

<b>Component</b>	<b>Toolbox includes...</b>	<b>Native optimization</b>	<b>Proxy support</b>
Integrated file system (IFS)	●	Use java.io.File	●
JarMaker	●		
Java application call	●		
Java program information	● <i>JTOpen 2.0</i>		
JDBC	●	Use native JDBC driver	●
Job and job log	●		
Message file	●		
Message queue	●		
NetServer	● <i>JTOpen 2.0</i>		
Permissions	●		
Print (e.g. spooled files, printers)	●		
Product license	● <i>JTOpen 2.0</i>		

- Components in *com.ibm.as400.access* package unless otherwise noted

# Component list (part 3)

<b>Component</b>	<b>Toolbox includes...</b>	<b>Native optimization</b>	<b>Proxy support</b>
Program call	●		●
Program Call Markup Language (PCML) ( <i>com.ibm.as400.data package</i> )	●		
Proxy server	●		●
Record-level database access	●	●	●
Resource framework ( <i>com.ibm.as400.resource package</i> )	● <i>JTOpen 2.0</i>		
Secure Sockets Layer (SSL)	●	●	● <i>JTOpen 2.0</i>
Service program call	●		●
Servlet classes ( <i>com.ibm.as400.util.servlet package</i> )	●		
Software resources	● <i>JTOpen 2.0</i>		
System pool	●		
System status	●		
System value	●		
Toolbox installer	●		
User and group	●		
User space	●		

- Components in *com.ibm.as400.access* package unless otherwise noted

## ***The Java standard for database access***

**Write Java programs in terms of standard JDBC interfaces, then plug in *any* JDBC driver - to work with *any* database!**

- Java gives you platform independence, JDBC gives you database independence

**java.sql package in Java Developers Kit**

**SQL is used extensively**

- Based on X/Open SQL Call Level Interface

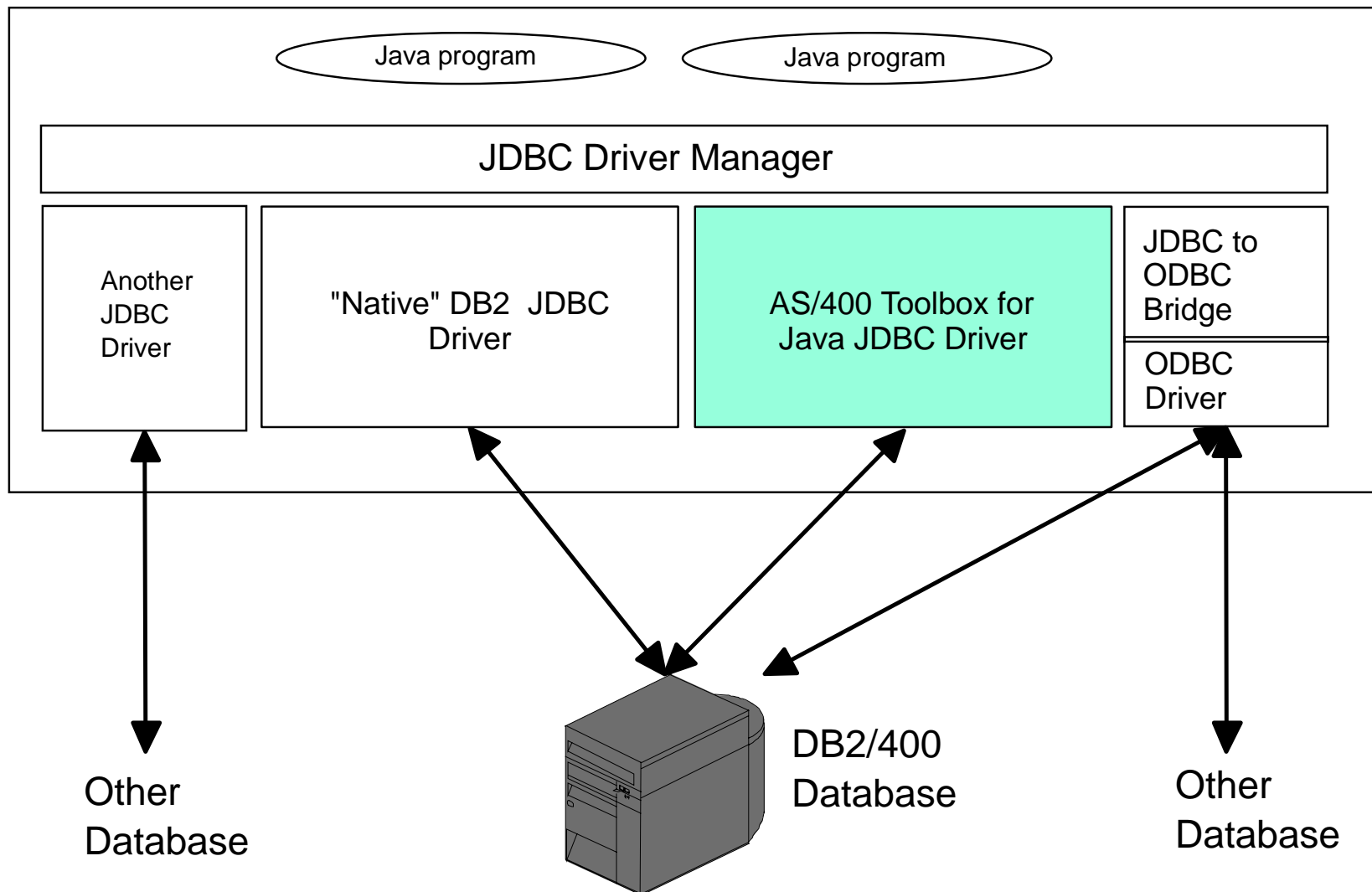
**Also supports:**

- Database definitions, manipulations, and queries
- Stored procedures
- Catalog methods
- Transactions (commit, rollback, isolation levels)



# JDBC

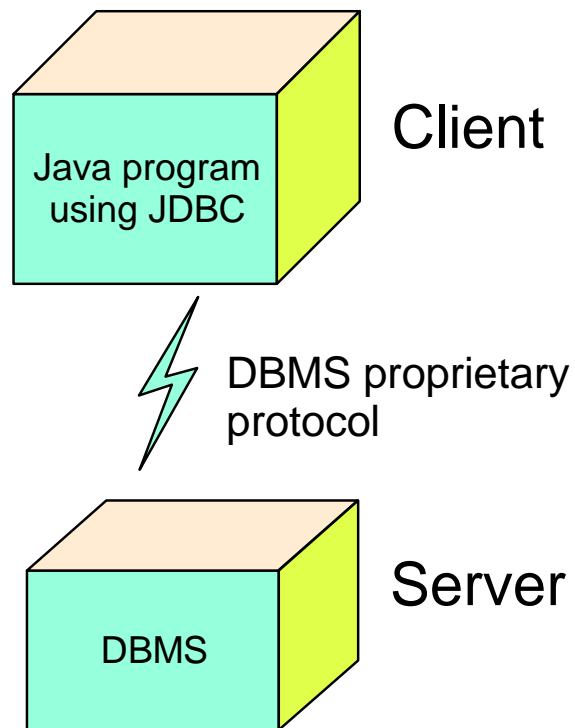
## *The Java standard for SQL database access*



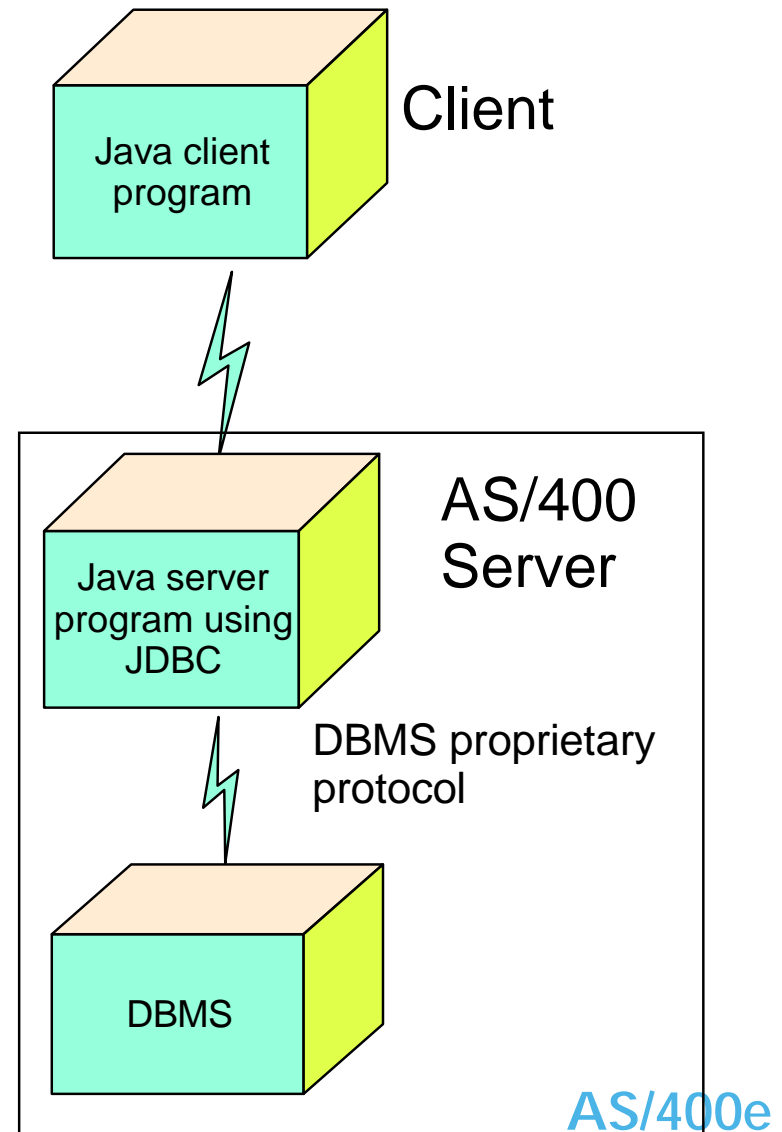
AS/400e

## Topologies

### Two-tier



### Three-tier



## Registering a JDBC driver

**A JDBC driver must be registered with the DriverManager**

**Most JDBC drivers will register themselves when they are loaded**

- `Class.forName ("JDBC.driver.class.name");`

**You can register JDBC drivers explicitly**

- `DriverManager.registerDriver (new JDBC.driver.class.name ());`

**The DriverManager can now dispatch requests to the registered JDBC driver**

```
// Register using a Java property
java -Djdbc.drivers=com.ibm.as400.access.AS400JDBCdriver myProgram

// Register by writing Java code
java.sql.DriverManager.registerDriver (new com.ibm.as400.access.AS400JDBCdriver());
java.sql.DriverManager.registerDriver (new com.ibm.db2.jdbc.app.DB2Driver());
```

## Connecting to a database

**Use the DriverManager to connect to a database**

- `Connection connection = DriverManager.getConnection("jdbc:your-databases-URL");`

**Userid and password are optional**

**The DriverManager will dispatch the connection request to the *appropriate* JDBC driver**

**Some drivers recognize additional connection properties**

```
Properties connProps = new Properties();  
connProps.put("cursor hold", "0");  
connProps.put("date format", "iso");
```

```
Connection c = DriverManager.getConnection("jdbc:as400://mySystem", connProps);
```

# *AS/400 JDBC driver choices*

## ***AS/400 Toolbox for Java JDBC driver***

***com.ibm.as400.access.AS400JDBCDriver***

- **100% Pure Java**
- **Communicates with the database using TCP/IP sockets**
- **Provides extended dynamic performance optimizations**
- **Great for:**
  - ◆ **Client/server applications**
  - ◆ **Applets**
  - ◆ **Servlets where the Web server and data are not on the same AS/400**



**AS/400e**

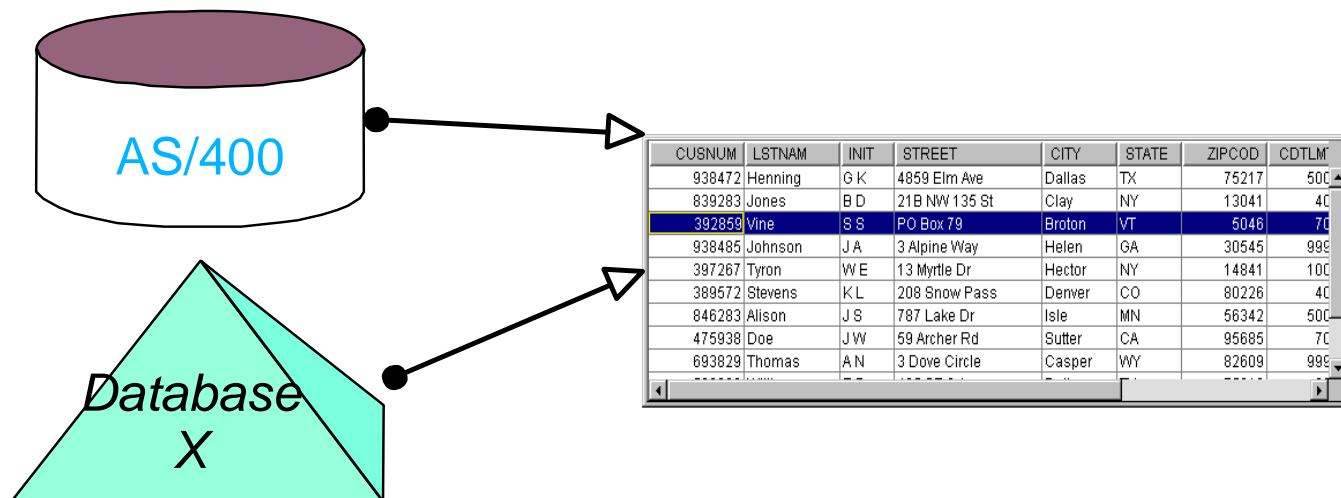
## **"Native" DB2 JDBC driver**

*com.ibm.db2.jdbc.app.DB2Driver*

- Communicates with the database using direct CLI calls
- Great for:
  - ◆ Server applications
  - ◆ Servlets where the Web server and data are on the same AS/400

## Code your program to be configurable

- Don't hardcode a JDBC driver
  - Allow your end users to plug in other JDBC drivers
  - Now your program works with *any* database!
- Differences between JDBC drivers:
  - Driver class name (needed for registering with the DriverManager)
  - URL syntax (needed for connecting)
  - Properties (used for customizing connection properties)
  - Subtle SQL differences
- Most of the logic and code will be the same!



## Statements

**Statement "handles" are needed to issue SQL statements**

- `Statement statement = connection.createStatement ();`
- `statement.executeUpdate ("INSERT INTO MYTABLE (COL1) VALUES (45)");`
- `ResultSet rs = statement.executeQuery ("SELECT * FROM MYTABLE");`

**Use PreparedStatement when executing an SQL statement multiple times, or when parameters are needed**

- `PreparedStatement ps = connection.prepareStatement ("INSERT INTO MYTABLE (?");`
- `ps.setInt (1, 45);`
- `ps.executeUpdate ();`



## ***Statements (continued)***

**Use CallableStatements when calling a stored procedure**

```
CallableStatement cs = connection.prepareCall ("CALL MYPROC (?, ?, ?)");  
cs.setInt (1, 88);  
cs.setInt (2, 99);  
cs.registerOutParameter (2, Types.INTEGER);  
cs.registerOutParameter (3, Types.VARCHAR);  
cs.executeUpdate ();  
int n = cs.getInt (2);  
String x = cs.getString (3);
```

## ResultSets

**ResultSet** contain the result data from a query

- `ResultSet rs = statement.executeQuery ("SELECT * FROM MYTABLE");`
- `String value = rs.getString ("COLUMNNA");`

**ResultSetMetaData** objects describe the columns in a **ResultSet**

- `ResultSetMetaData rsmd = rs.getMetaData ();`
- `String columnName = rsmd.getColumnName (1);`
- `int displaySize = rsmd.getColumnDisplaySize (1);`

## ***What else is there?***

### **DatabaseMetaData**

- Information about tables, columns, procedures, ...

### **SQLExceptions and SQLWarnings**

- Used for error handling

### **JDBC 2.0 Core (only available under Java 2)**

- Updatable result sets
- Scrollable result sets
- LOBs (Large objects)
- Batch updates

### **JDBC 2.0 Optional Package (new in JTOpen)**

- Connection pooling
- Data sources
- Row sets
- Distributed transactions

# *AS/400 Toolbox for Java JDBC specifics*

## Connection properties

- Can be set in `DriverManager.getConnection()`:

```
Properties connProps = new Properties();  
connProps.put("cursor hold", "true");  
connProps.put("date format", "iso");  
  
Connection c = DriverManager.getConnection("jdbc:as400://mySystem", connProps);
```

- ...or in the URL:

- This is helpful because no code change/recompile is needed

```
Connection c = DriverManager.getConnection("jdbc:as400://mySystem;cursor  
hold=false;date format=iso", connProps);
```

# AS/400 Toolbox for Java JDBC specifics

Some helpful connection properties:

<b>Connection property</b>	<b>Description</b>
"libraries"	Specify a library list, e.g. "MYLIB,*LIBL,ANOTHER"
"date format", "time format"	Specify the format for String representations of dates and times, e.g. "iso", "mdy", "usa"
"naming"	Specify the naming convention for qualified table names, either "sql" (for collection.table) or "system" (for library/file)
"block criteria", "block size"	Define block size for fetching multiple rows, can greatly improve performance
"extended dynamic", "package cache", etc.	Use extended dynamic support. Improves performance when same statements are prepared repeatedly - even across different runs of the program
"secure"	Use Secure Sockets Layer (SSL)
"translate binary"	Specify "true" if you have text strings stored in binary columns (some legacy programs do this)
"trace"	Generate trace data - useful for debugging and problem reporting

*There are many other connection properties...*

## ***Fast access to AS/400 database files***

### **Provides access to database files:**

- Access records sequentially, by record number or key
- Support for locking
- Support for transactions (commit and rollback)

### **Options for describing the Record Format:**

- The programmer can write the code
- The Toolbox can retrieve the record format at development-time and output Java source code
- The Toolbox can retrieve the record format at run-time

**When running on the AS/400, direct API calls are made instead of using the server**

# *Record-level database access*

## ***Fast access to AS/400 database files***

```
QSYSObjectPathName fileName = new QSYSObjectPathName("QIWS", "QCUSTCDT", "FILE");

SequentialFile file = new SequentialFile(as400, fileName.getPath());

file.setRecordFormat();          // Loads the record format directly from the AS/400.

file.open();

Record data = file.readNext();

while (data != null)
{
    System.out.print((String)data.getField("INIT") + "  ");
    System.out.print((String)data.getField("LSTNAM") + "  ");
    System.out.println((BigDecimal)data.getField("BALDUE"));
    data = file.readNext();
}
```

## ***Fast access to AS/400 database files***

### **Performance tips**

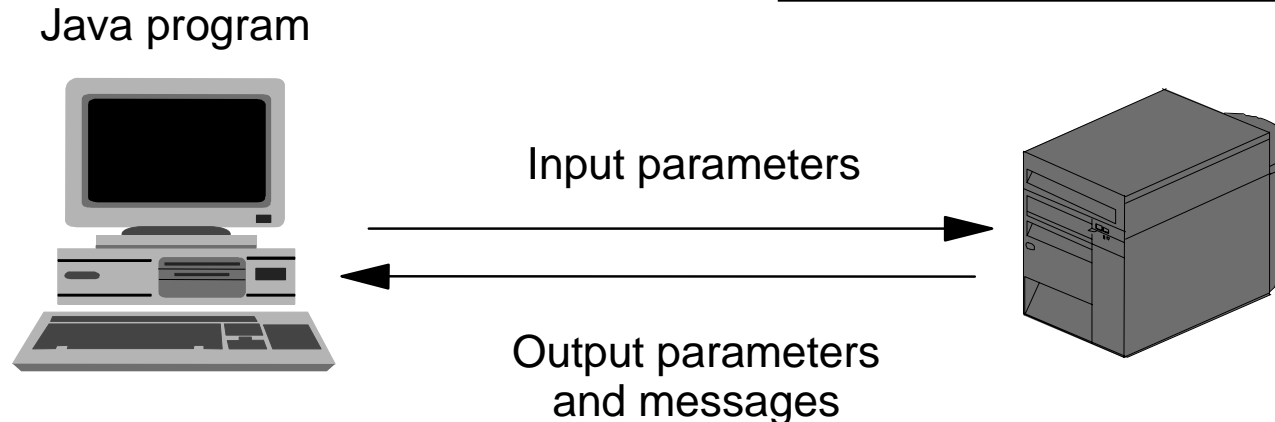
- Avoid retrieving the record format multiple times. Retrieve it once and save a reference to it or hard code the record format
- Blocking factor means record caching. Experiment with different sizes or specify zero and let the Toolbox determine the blocking factor.
- Blocking factor valid only when file is opened for READ\_ONLY or WRITE\_ONLY access.
- Opening keyed files is slower than opening sequential files so use sequential files unless you are going to search by key



# Command call and program call

## Make use of legacy code and system APIs

```
AS400 system = new AS400();  
CommandCall cc = new CommandCall(system);  
cc.run("CRTLIB NEWLIB");
```



```
AS400 system = new AS400();  
ProgramParameter[] parmList = new ProgramParameter[n];  
parmList[0] = new ProgramParameter(data);  
...  
ProgramCall pc = new ProgramCall(system,  
                                "/QSYS.LIB/MYLIB.LIB/MYPGM.PGM", parmList);  
pc.run();
```

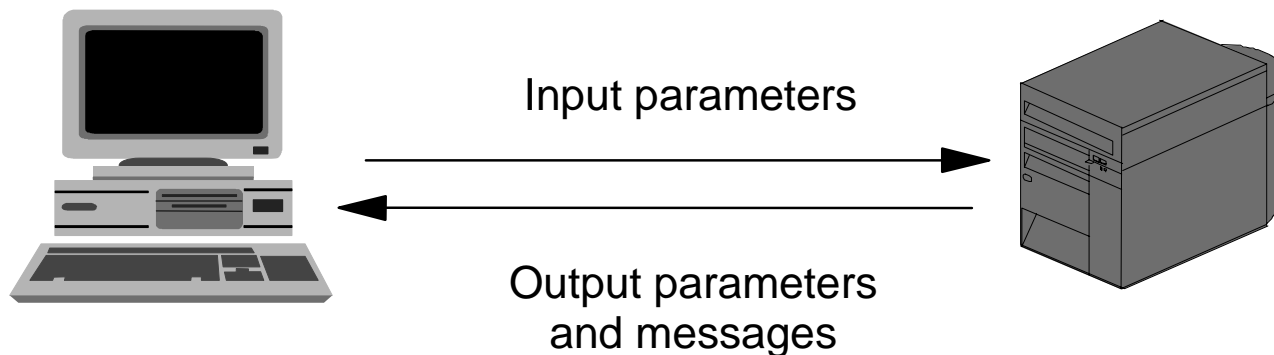
# *Program Call Markup Language (PCML)*

## ***Describe program calls using XML***

**Parameter handling in traditional Toolbox ProgramCall can be tedious**

### **PCML:**

- Simplifies data description and conversion
- Iterative development without recompile



# Traditional Program Call vs PCML

**(PCML)**

```
<pcml version="1.0">
  <struct name="usri0100">
    <data name="bytesReturned"           type="int"       length="4"   usage="output" />
    <data name="bytesAvailable"          type="int"       length="4"   usage="output" />
    <data name="userProfile"             type="char"      length="10"  usage="output" />
    <data name="previousSignonDate"      type="char"      length="7"   usage="output" />
    <data name="previousSignonTime"      type="char"      length="6"   usage="output" />
    <data name=" "                      type="byte"      length="1"   usage="output" />
    <data name="badSignonAttempts"       type="int"       length="4"   usage="output" />
    <data name="status"                  type="char"      length="10"  usage="output" />
    <data name="passwordChangeDate"      type="byte"      length="8"   usage="output" />
    <data name="noPassword"              type="char"      length="1"   usage="output" />
    <data name=" "                      type="byte"      length="1"   usage="output" />
    <data name="passwordExpirationInterval" type="int"       length="4"   usage="output" />
    <data name="datePasswordExpires"     type="byte"      length="8"   usage="output" />
    <data name="daysUntilPasswordExpires" type="int"       length="4"   usage="output" />
    <data name="setPasswordToExpire"     type="char"      length="1"   usage="output" />
    <data name="displaySignonInfo"       type="char"      length="10"  usage="output" />
  </struct>

  <program name="qsyrusri" path="/QSYS.lib/QSYRUSRI.pgm">
    <data name="receiver"                type="struct"    usage="output"  struct="usri0100" />
    <data name="receiverLength"          type="int"        length="4"      usage="input"    />
    <data name="format"                  type="char"       length="8"      usage="input"    init="USRI0100" />
    <data name="profileName"             type="char"       length="10"     usage="input"    init="*CURRENT" />
    <data name="errorCode"              type="int"        length="4"      usage="input"    init="0"         />
  </program>
</pcml>
```

- - - - -

```
pcml = new ProgramCallDocument(as400System, "qsyrusri");
pcml.setValue("qsyrusri.receiverLength", new Integer((pcml.getOutputsize("qsyrusri.receiver"))));
rc = pcml.callProgram("qsyrusri");
value = pcml.getValue("qsyrusri.receiver.bytesReturned");
```

**AS/400e**

# *Traditional Program Call vs PCML*

***(Traditional)***

```
AS400Bin4 bin4      = new AS400Bin4();
AS400Text char6     = new AS400Text(6,  as400System);
AS400Text char7     = new AS400Text(7,  as400System);
AS400Text char8     = new AS400Text(8,  as400System);
AS400Text char10    = new AS400Text(10, as400System);

ProgramCall pc = new ProgramCall(as400System);
pc.setProgram("/QSYS.LIB/QSYRUSRI.PGM");

ProgramParameter[] parms = new ProgramParameter[5];

parms[0] = new ProgramParameter(100);
parms[1] = new ProgramParameter(bin4.toBytes(100));
parms[2] = new ProgramParameter(char8.toBytes("USRI0100"));
parms[3] = new ProgramParameter(char10.toBytes("*CURRENT"));
byte[] errorArea = new byte[32];
parms[4] = new ProgramParameter(errorArea, 32);
pc.setParameterList(parms);
pc.run();
byte[] data = parms[0].getOutputData();
int value = ((Integer) bin4.toObject(data, 4)).intValue();
```

# HTML and servlet classes

- Classes for generating HTML output
- Useful for servlets, report generating, etc.

CUSNUM	LSTNAM	STREET	CITY	STATE	ZIPCODE
938472	Henning	4859 Elm Ave	Dallas	TX	75219
839283	Jones	21B NW 135 St	Clay	NY	13027
392859	Vine	PO Box 79	Broton	VT	5602
938485	Johnson	3 Alpine Way	Helen	GA	30131
397267	Tyron	13 Murtl			

```
// Execute an SQL statement and get the result set.
Statement statement = connection.createStatement();
statement.execute("SELECT * FROM QIWS.QCUSTCDT");
ResultSet resultSet = statement.getResultSet();

// Create the SQLResultSetRowData object and initialize to the result set.
SQLResultSetRowData rowData = new SQLResultSetRowData(resultSet);

// Create an HTML converter object and convert the rowData to HTML.
HTMLTableConverter conv = new HTMLTableConverter();
HTMLTable[] html = conv.convertToTables(rowData);

// Display the HTML table generated by the converter.
out.println(html[0]);
```

# *JTOpen (Open Source)*

**Much of the Toolbox code is open source - known as "JTOpen"**

***<http://www.ibm.com/developerworks/opensource>***

- Part of IBM's open source development community
- Use source as a debug tool
- Submit new function
- Modify source for your use
- Submit problem reports and bug fixes



## **Two versions of the Toolbox:**

- Licensed program (AS/400 Toolbox for Java, LPP 57xxJC1)
  - Only IBM developed code
  - Supported by IBM
- Open source version (JTOpen)
  - Source from many non-IBM contributors
  - Supported by open source community
  - New functions and fixes available first here!

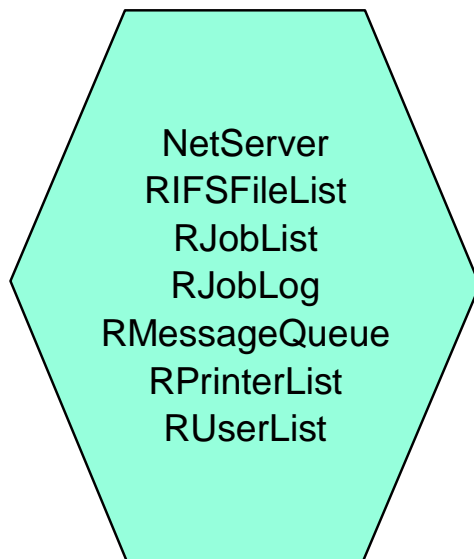
# Resource framework

## A generic framework for AS/400 lists and resources

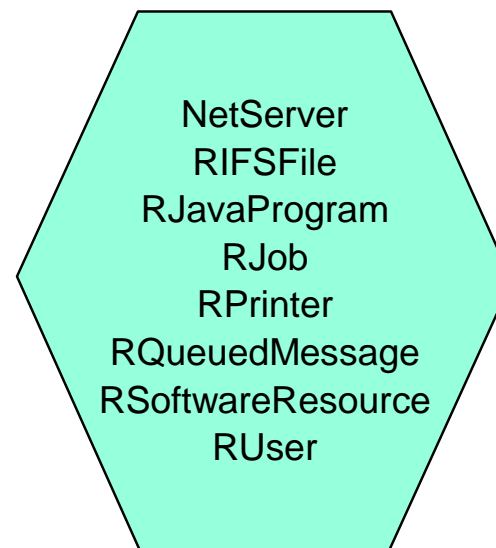
- Provides a consistent API for all such objects
- Efficient buffering and paging managed internally
- Self-describing components - lists of attributes
- Allows you to hardcode specific uses - e.g. get messages from a job log
- ... or write generic code - e.g. get resources from a list



### *ResourceList subclasses*



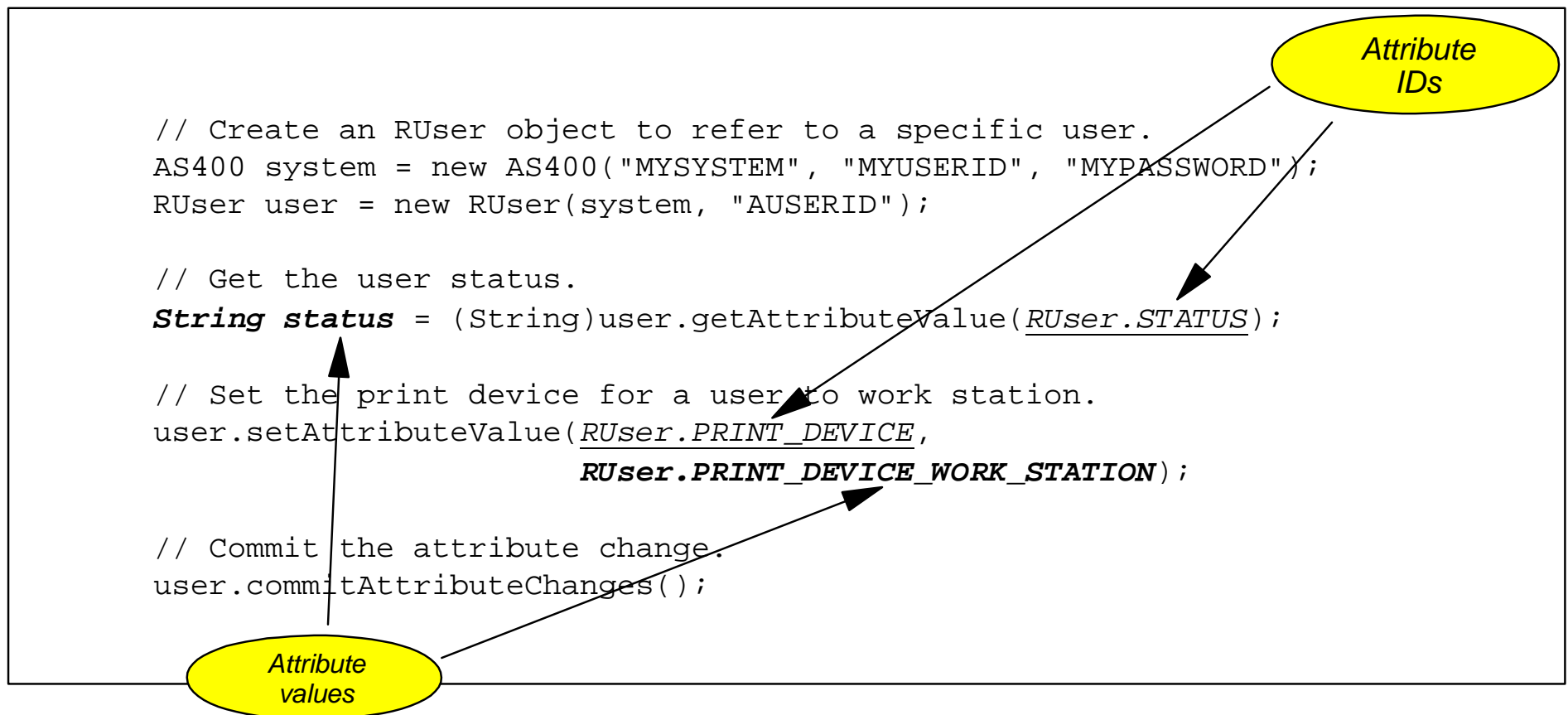
### *Resource subclasses*



# Resource framework

## Using a resource

- Each type of resource defines a set of *attributes* - use the attribute ID to get the attribute value
- Setting attribute values do not take effect until changes are committed.
- Example: Using an RUser



- Code for other resources is very similar - only the attribute IDs change



# Resource framework

## Using a resource list

- Resource lists need to be *opened* and *closed*
- Resource lists are usually generated *asynchronously*
  - This saves time: A GUI usually only needs first 20 or so items
  - Entire list is not necessarily available after list is opened
  - Call *waitForResource()* or *waitForComplete()* to force a wait
- Use *resourceAt()* to get a particular resource from the list
- ... use *resources()*, which returns a Java Enumeration
- Most lists provide detailed filtering and sorting



# Resource framework

## Example: Using an RJobLog

```
// Create an RJobLog object to represent a specific job log.
AS400 system = new AS400("MYSYSTEM", "MYUSERID", "MYPASSWORD");
RJobLog jobLog = new RJobLog(system, "AJOBNAME", "AUSERID", "654124");

// Open the list and wait for it to complete loading.
jobLog.open();
jobLog.waitForComplete();

// Read and print the messages in the list.
long numberOfMessages = jobLog.getListLength();
for(long i = 0; i < numberOfMessages; ++i)
{
    RQueuedMessage message = (RQueuedMessage)jobLog.resourceAt(i);
    String text = message.getAttributeValue(RQueuedMessage.MESSAGE_TEXT);
    System.out.println(text);
}

// Close the list.
jobLog.close();
```

# Resource framework

## Resource list accessories

- Several classes are available for presenting resources and lists
- Such classes are written generically - will work with any ResourceList or Resource subclass
  - Those provided in JTOpen
  - You can write your own!

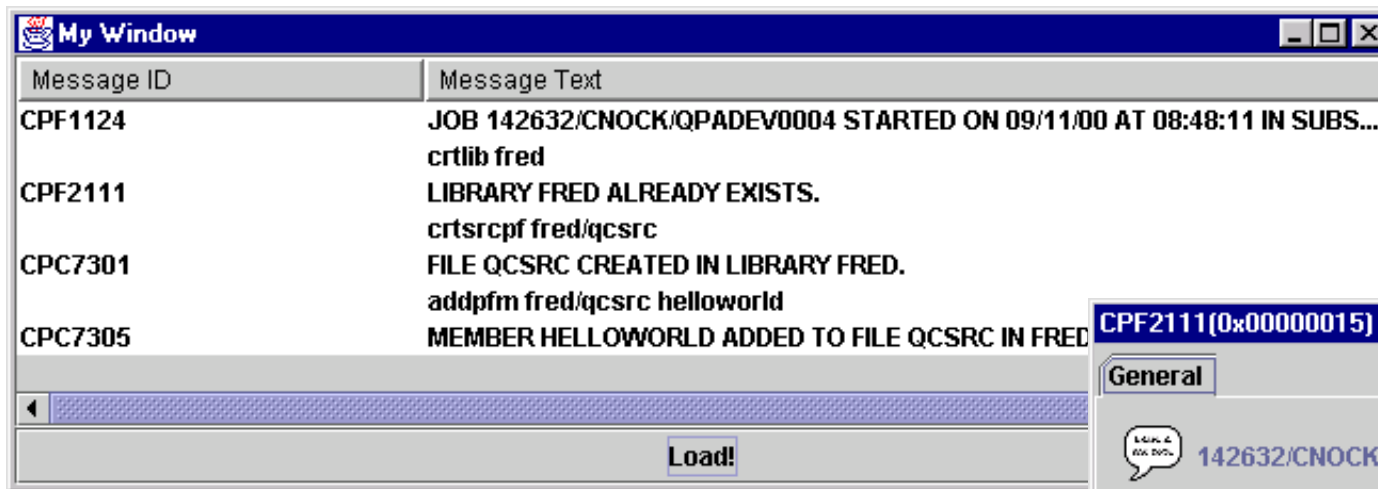
com.ibm.as400.util.servlet.ResourceListRowData	Display the contents of any ResourceList in an HTML table or form for use in a servlet. Use in conjunction with HTMLTableConverter or HTMLFormConverter
com.ibm.as400.vaccess.ResourceListDetailsPane	Display the contents of any ResourceList in a Swing JTable - integrated into any graphical user interface
com.ibm.as400.vaccess.ResourceListPane	Display the contents of any ResourceList in a Swing JList - integrated into any graphical user interface
com.ibm.as400.vaccess.ResourceProperties	Define an editable "properties" pane for any Resource.

- You can also write you own generic utilities

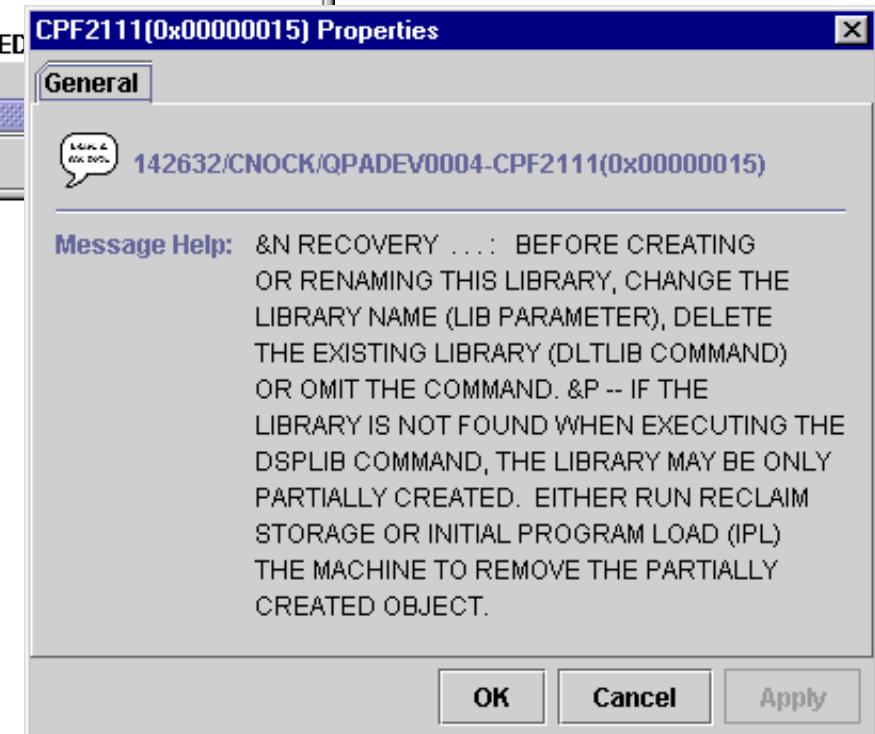
# Resource framework

## Resource accessories

- ResourceListDetailsPane



- ResourceProperties



- ResourceListRowData

Job Name	Job Number
SCPF	000000
QSYSARB	138662
QSYSARB2	138663
QSYSARB3	138664

# Resource framework

## Tips for using the resource framework

- `import com.ibm.as400.resource.*;`
- When using the javadoc documentation, don't forget to look at the superclass's documentation
  - The resource framework forms an inheritance hierarchy
  - Most public methods are common - defined in `Resource` or `ResourceList`
  - The classes that you use are subclasses - e.g. `RUser`, `RJobList`
- You can extend the framework to include your own lists and resources
- Some classes have flavors in the `com.ibm.as400.access` package, too
  - The resource classes are recommended, since...
    - ▶ The resource classes will be more efficient
    - ▶ The resource classes have more function
  - The access classes are still available for backwards-compatibility:

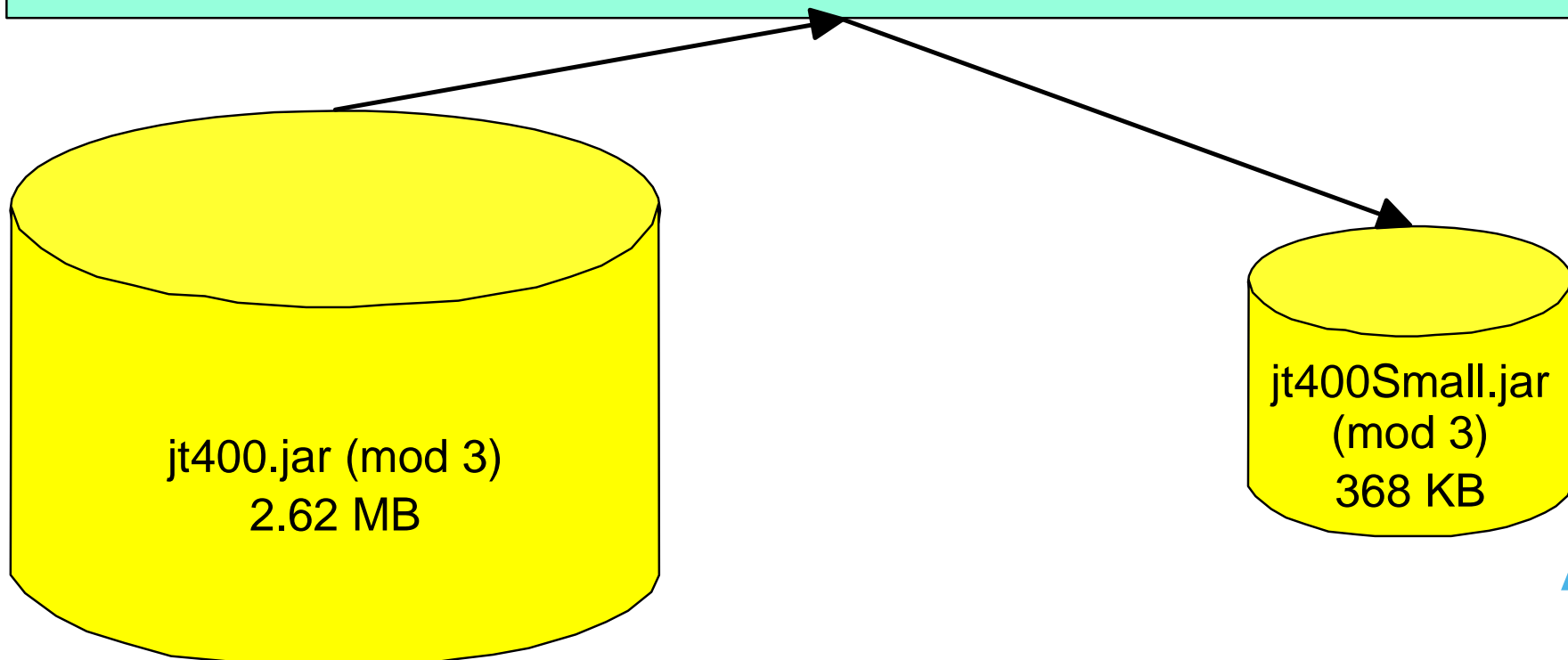
<b>com.ibm.as400.access</b> <i>(for backwards compatibility only)</i>	<b>com.ibm.as400.resource</b> <i>(Use this instead!)</i>
Job, JobList, JobLog	RJob, RJobList, RJobLog
MessageQueue, QueuedMessage	RMessageQueue, RQueuedMessage
User, UserList	RUser, RUserList

# JarMaker

## Reduce .jar file sizes

- The jt400.jar file is > 2.5 MB - This contains nearly all of the functionality described here
- A given program typically only needs a small portion of the code (e.g. only CommandCall)
- AS400ToolboxJarMaker pares down jt400.jar to contain only the code you need
- JarMaker also works on jar files other than jt400.jar

```
java utilities.AS400ToolboxJarMaker -component CommandCall  
-ccsid 37 -source jt400.jar -destination jt400Small.jar
```





## ***Where can I get more information?***

**<http://ibm.com/as400/toolbox>**

- News, downloads, FAQs, service packs, articles, COMMON labs

**<http://oss.software.ibm.com/developerworks/opensource/jt400/>**

- JTOpen - open source, bug reporting, feature requests

**<http://as400service.ibm.com/>**

- AS/400 Technical Forums - including AS/400 Toolbox for Java/JTOpen Forum

### ***AS/400 Toolbox for Java Programmers Guide***

- Shipped with the AS/400 Toolbox for Java
- Contains overview, full API documentation (javadoc), and code examples
- Available at AS/400 Information Center
- Download from <http://ibm.com/as400/toolbox>

### ***Building AS/400 Client/Server Applications with Java***

- Redbook SG24-2152-02