LegacyJ™

Putting IT All Together

HP 3000 applications

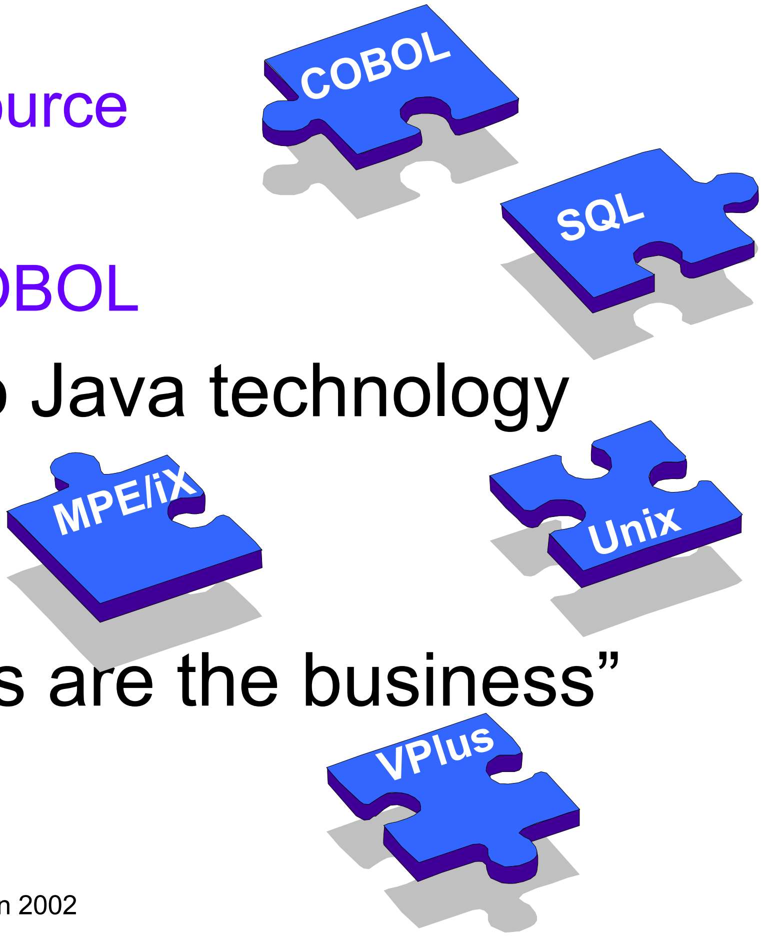
It's not Dead, Just Moved

Agenda

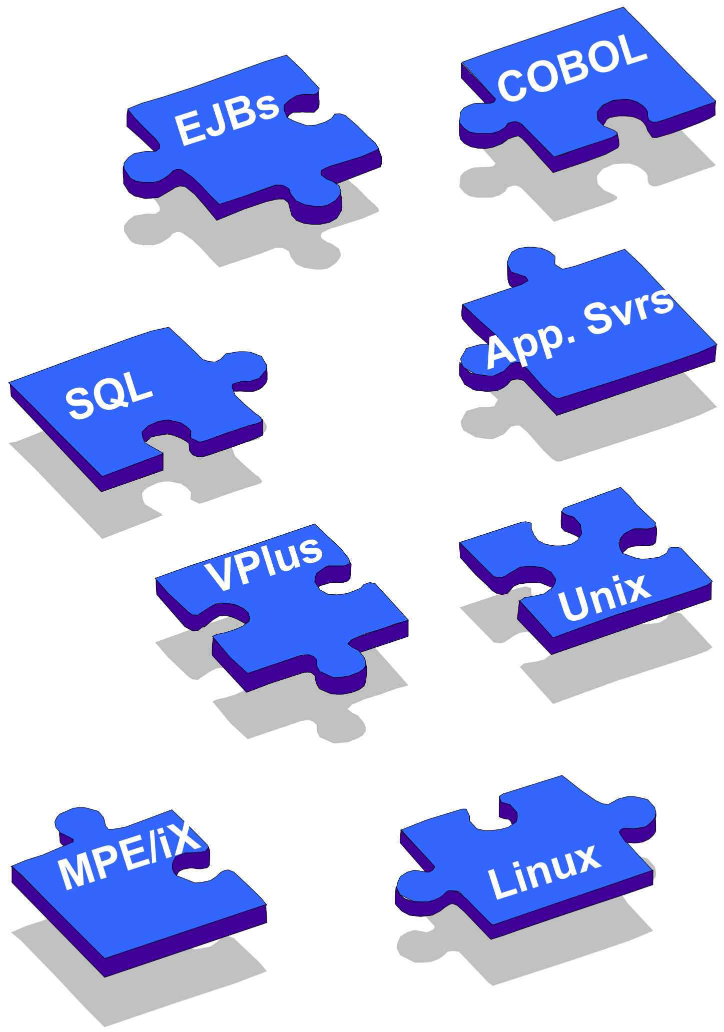
- LegacyJ Overview
- PERCobol 3.0 IDE
- HP e3000 Migration
- Propagating the Application Server (EJBs)
- Case Studies
- Summary

LegacyJ

- Solutions enable legacy modernization and mobilization
  - EJB directly from COBOL source
  - XML in COBOL files
  - Graphical Interfaces with COBOL
- COBOL anywhere . . . To Java technology everywhere!
  - Platform independence
- "Because the applications are the business"



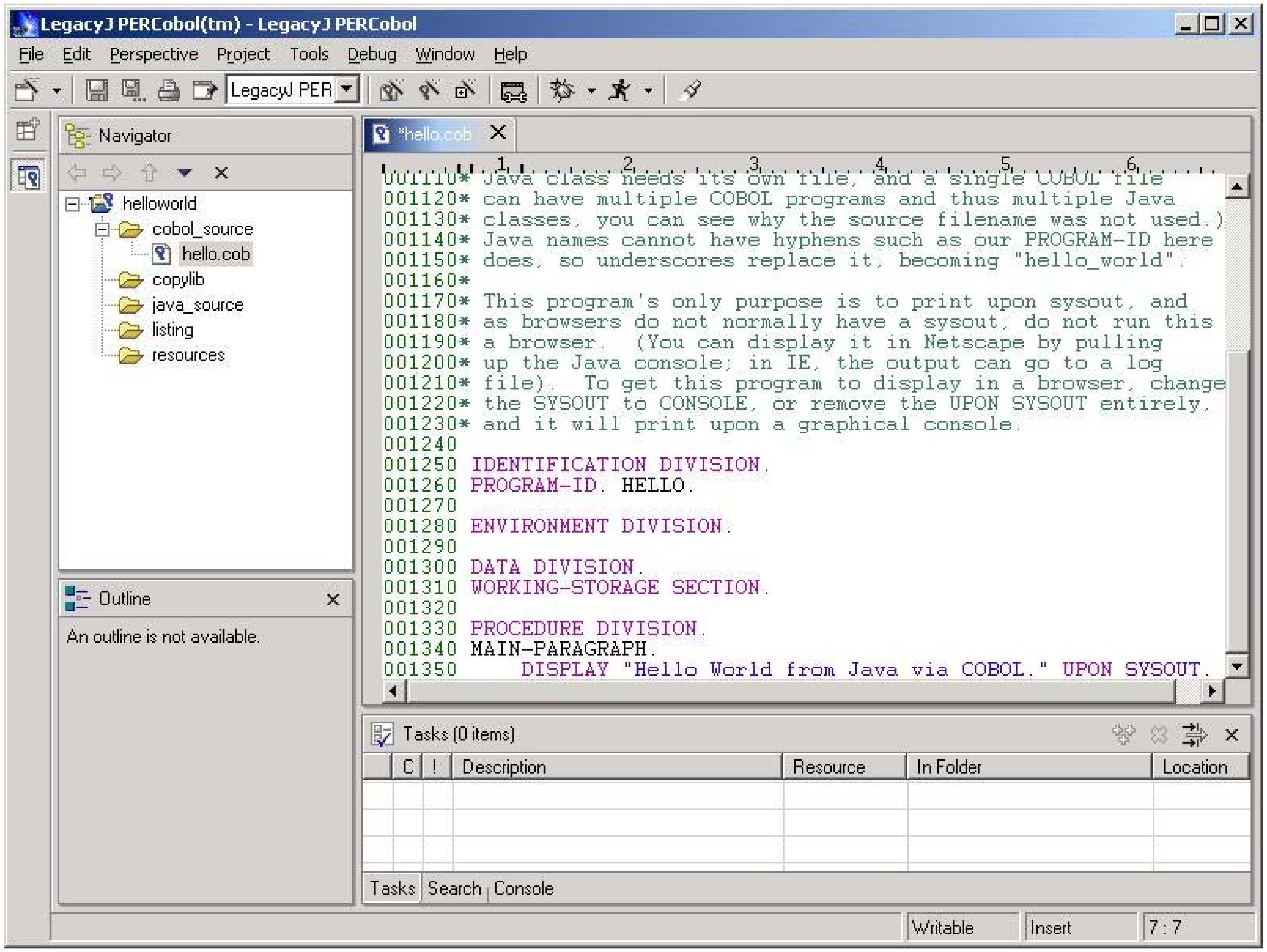
Putting it All Together



CHALLENGE: leverage legacy applications for platform independence and data to populate modern Application Servers

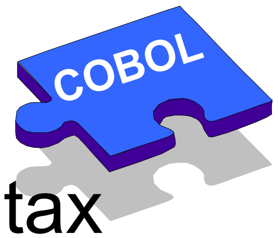
- COBOL to EJBs (PERCobol)
- J2EE Compliance
- Access legacy data
- Platform independence
- Reduce costs
- Remove proprietary restrictions
- XML, Java and other web enabling technologies

PERCobol 3.0 IDE


LegacyJ PERCobol(tm) - LegacyJ PERCobol
File Edit Perspective Project Tools Debug Window Help
LegacyJ PER
Navigator
helloworld
cobol_source
hello.cob
copylib
java_source
listing
resources
Outline
An outline is not available.
*hello.cob
Tasks (0 items)
C ! Description Resource In Folder Location
Tasks Search Console
Writable Insert 7 : 7

```
001110* Java class needs its own file, and a single COBOL file
001120* can have multiple COBOL programs and thus multiple Java
001130* classes, you can see why the source filename was not used.)
001140* Java names cannot have hyphens such as our PROGRAM-ID here
001150* does, so underscores replace it, becoming "hello_world".
001160*
001170* This program's only purpose is to print upon sysout, and
001180* as browsers do not normally have a sysout, do not run this
001190* a browser.  (You can display it in Netscape by pulling
001200* up the Java console; in IE, the output can go to a log
001210* file).  To get this program to display in a browser, change
001220* the SYSOUT to CONSOLE, or remove the UPON SYSOUT entirely,
001230* and it will print upon a graphical console.
001240
001250 IDENTIFICATION DIVISION.
001260 PROGRAM-ID. HELLO.
001270
001280 ENVIRONMENT DIVISION.
001290
001300 DATA DIVISION.
001310 WORKING-STORAGE SECTION.
001320
001330 PROCEDURE DIVISION.
001340 MAIN-PARAGRAPH.
001350     DISPLAY "Hello World from Java via COBOL." UPON SYSOUT.
```

LegacyJ PERCobol



- PERCobol compiles HP COBOL syntax and most other variants.
- Generates J2EE compliant EJBs *(PERCobol runtimes are Java based)*

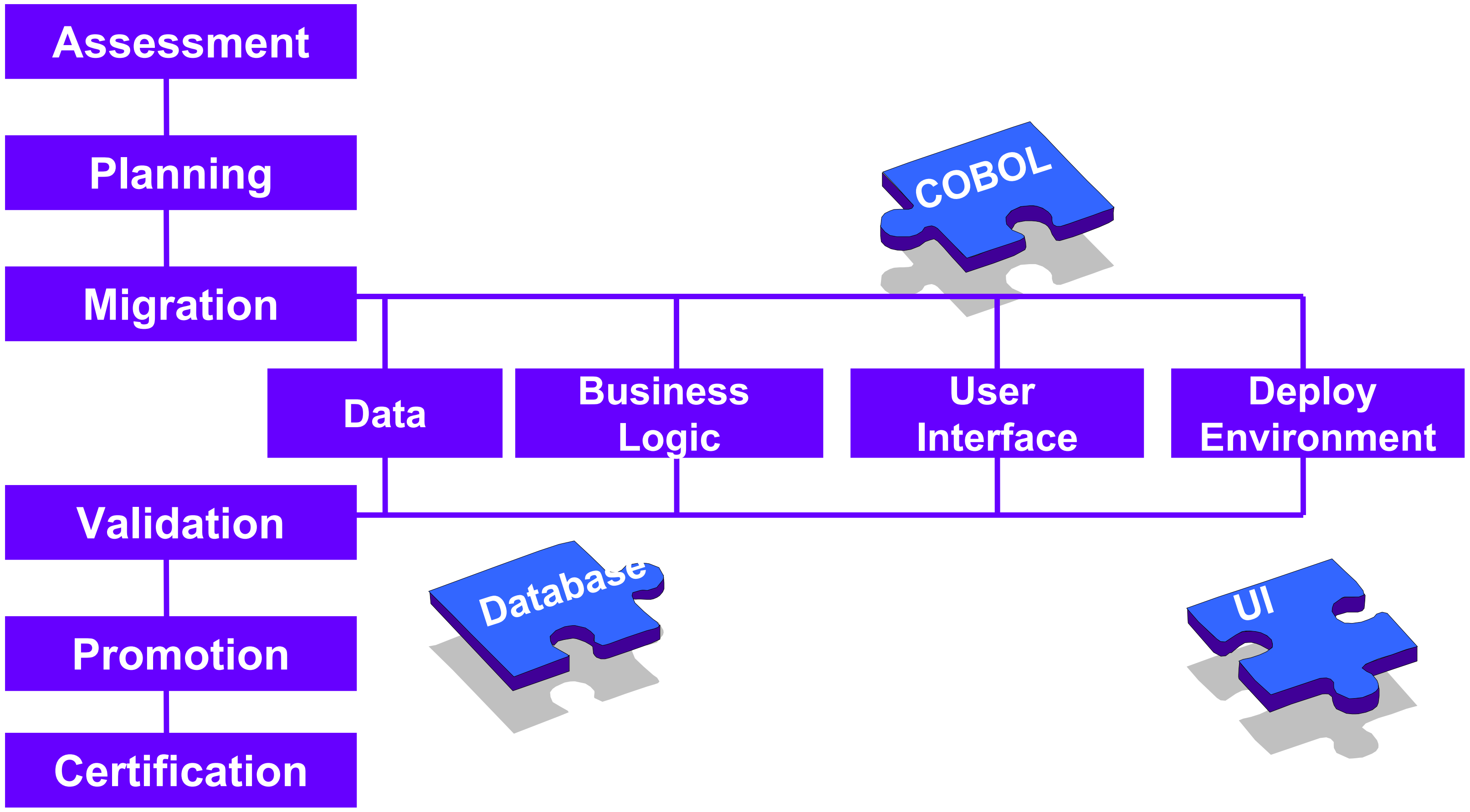
- Access to SQL data sources *(JDBC2 compliant)*

- *PERCobol read XML as COBOL Data*

HP 3000 Migration

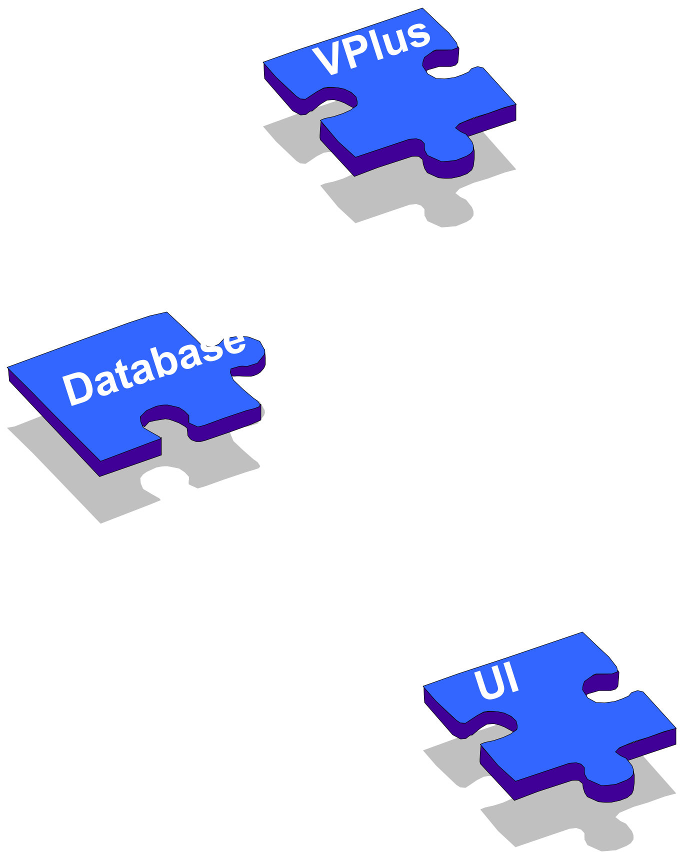
	Component	Solution	Target Result
UI	VPlus	ViewJ	Graphical Screens
UI	VPlus Intrinsics	PERCobol	Platform Independent Intrinsics
Business Logic	COBOL Application	PERCobol	Platform Independent COBOL
Business Logic	Intrinsics (Selected)	PERCobol	Platform Independent Intrinsics
Business Logic	Macros	PERCobol	Platform Independent Macros
Data	IMAGE	Image Intrinsics PERCobol	SQL Database*
Data	KSAM	KSAM Intrinsics PERCobol	Indexed Files (C-ISAM)

Migration Flow



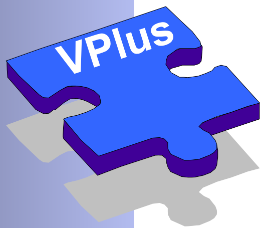
Intrinsics

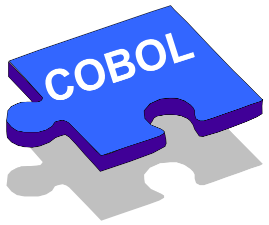
- General Intrinsics
  - Date/Time
  - Data Conversion
  - Message Catalog
  - Local locking
  - JCW
- Image Intrinsics
- VPlus Intrinsics



General Intrinsics

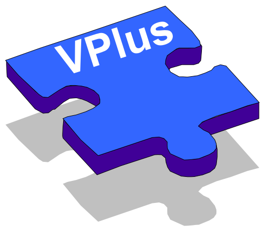
- ALMANAC
- ASCII
- BINARY
- CALENDAR
- CATCLOSE
- CATOPEN
- CATREAD
- CLOCK
- CTRANSLATE
- DASCII
- DATELINE
- DBINARY
- FINDJCW
- FMTCALENDAR
- FMTCLOCK
- FMTDATE
- GETJCW
- GETLOCRIN
- LOCKLOCRIN
- PAUSE
- PRINT
- PRINTOP
- PRINTOPREPLY
- PUTJCW
- SETJCW
- STACKDUMP
- TIMER
- UNLOCKLOCRIN

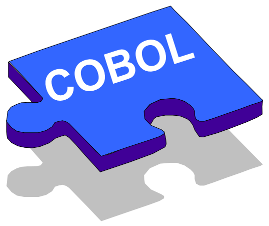




VPlus Intrinsics Supported

VCHANGEFIELD
VCLOSEFORMF
VCLOSETERM
VERRMSG
VFIELDEDITS
VFINISHFORM
VGETBUFFER
VGETDINT
VGETFIELD
VGETFIELDINFO
VGETFILEINFO
VGETFORMINFO
VGETINT
VGETKEYLABELS
VGETLANG
VGETLONG
VGETNEXTFORM
VGETPACKED
VGETREAL
VGETYYMMDD
VGETZONED
VINITFORM
VLOADFORMS
VOPENFORMF
VOPENTERM
VPLACECURSOR
VPRINTFORM
VPRINTSCREEN
VPUTBUFFER
VPUTDINT
VPUTFIELD
VPUTINT
VPUTLONG
VPUTPACKED
VPUTREAL
VPUTWINDOW
VPUTYYMMDD
VPUTZONED
VREADFIELDS
VSETERROR
VSETKEYLABEL
VSETKEYLABELS
VSETLANG
VSHOWFORM
VTURNOFF
VTURNON
VUNLOADFORM





ViewJ

Platform Independent VPlus screen execution

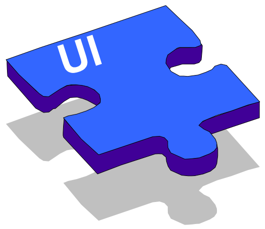
- Text
- Basic Graphic

VPlus Forms

- Slow or Fast Forms
- Process Specification
- Intrinsic Interface

Dynamic conversion (Execution Time)

- Application
- Web-enabled (Applet)



A Staged Approach

Data

Option → **Complete Intrinsics to HP e3000 (dbcalls, mpefile, system) from PERCobol to HP3000**

Available Today

Option → **Standard Database Intrinsics (DBOPEN, DBCLOSE, DBFIND…) from PERCobol to HP Eloquence on Linux, HP-UX and Windows**

Available Today

Option → **Standard Database Intrinsics (DBOPEN, DBCLOSE, DBFIND…) from PERCobol to Oracle, DB2, Sybase, SQL Server, etc.**

Available 2Q02

E3000 Database Summary

Flexibility is important with the ability to deploy anywhere

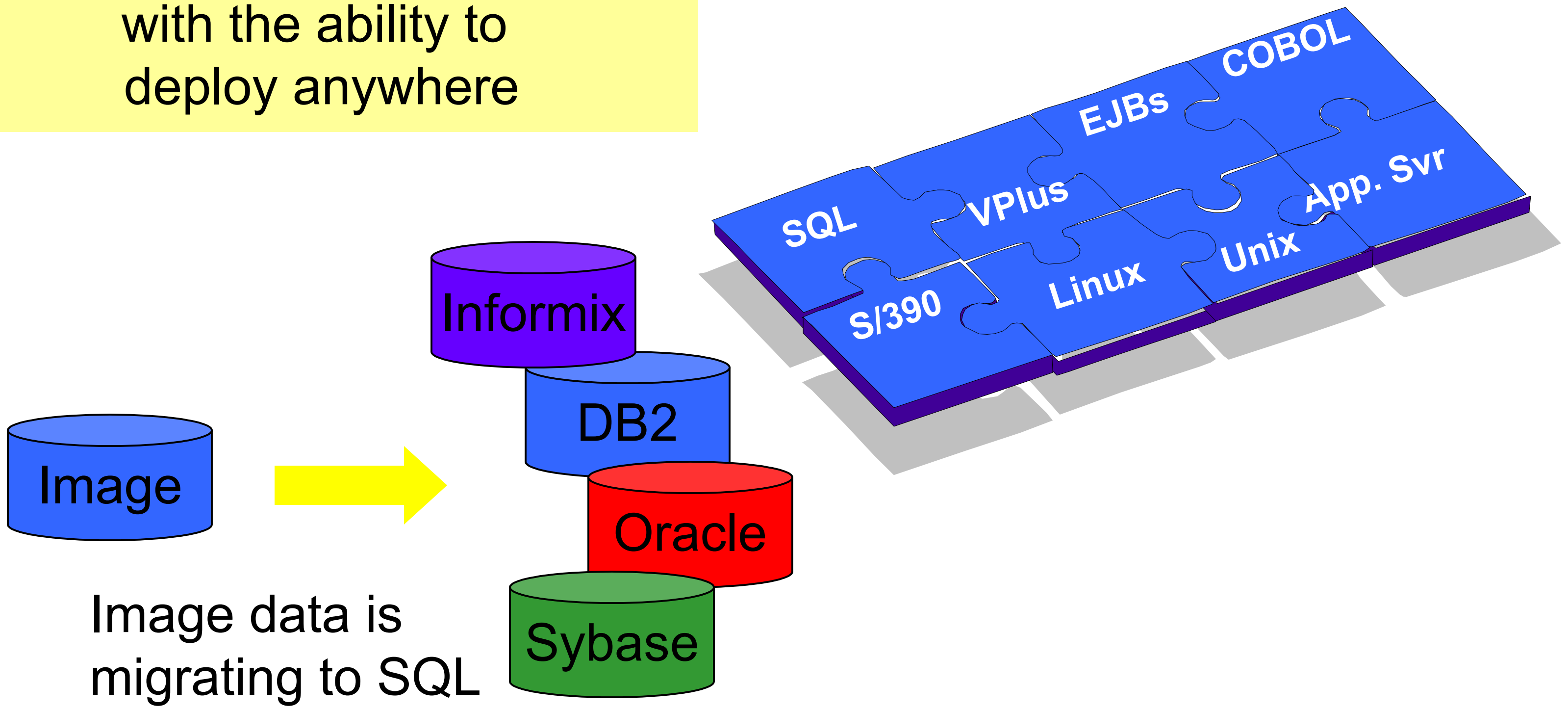
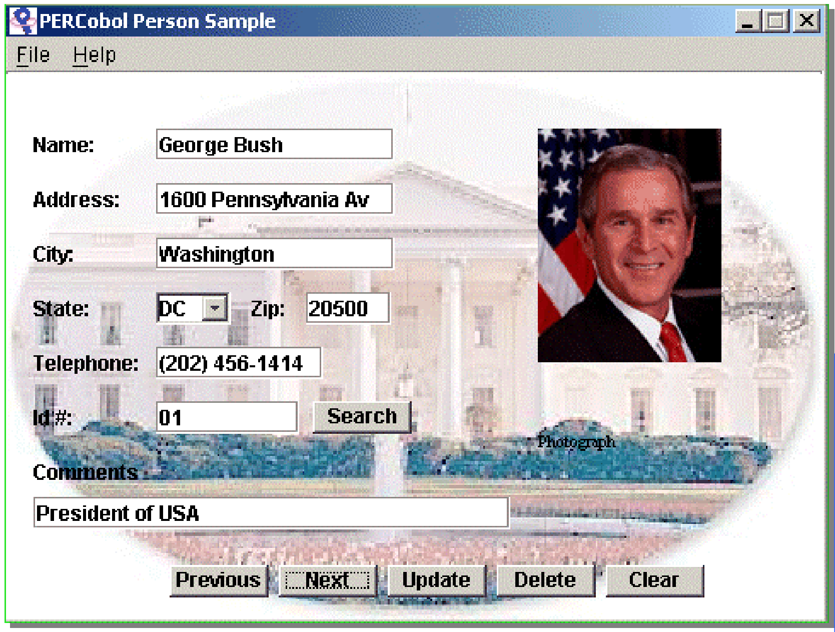


Image data is migrating to SQL

Graphical COBOL



Name: George Bush

Address: 1600 Pennsylvania Av

City: Washington

State: DC Zip: 20500

Telephone: (202) 456-1414

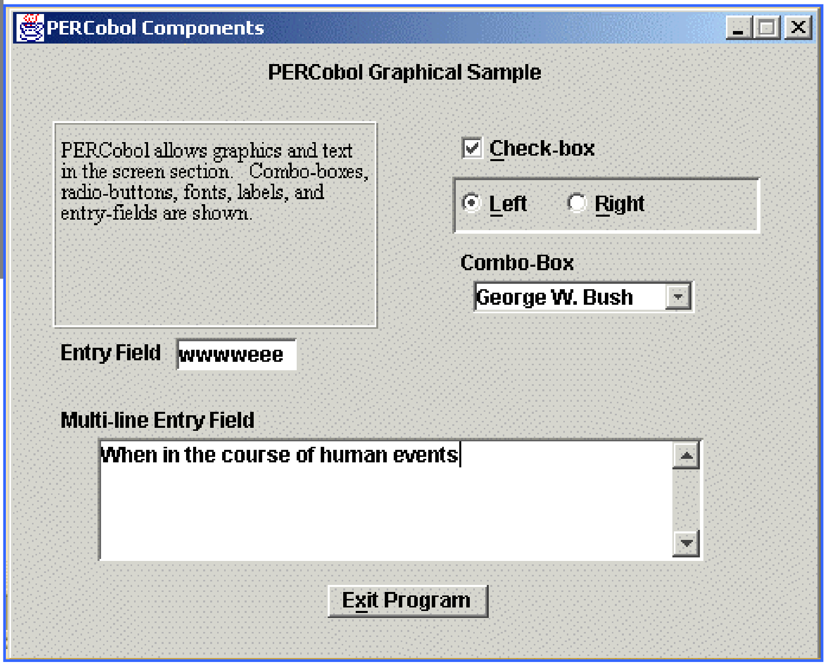
Id #: 01

Comments: President of USA

PERCobol™

COBOL or Java GUIs

COBOL graphical Screen Section



PERCobol allows graphics and text in the screen section. Combo-boxes, radio-buttons, fonts, labels, and entry-fields are shown.

[✓] Check-box

[● Left] ○ Right

Combo-Box: George W. Bush

Entry Field: wwwweee

Multi-line Entry Field: When in the course of human events

Graphical Cobol

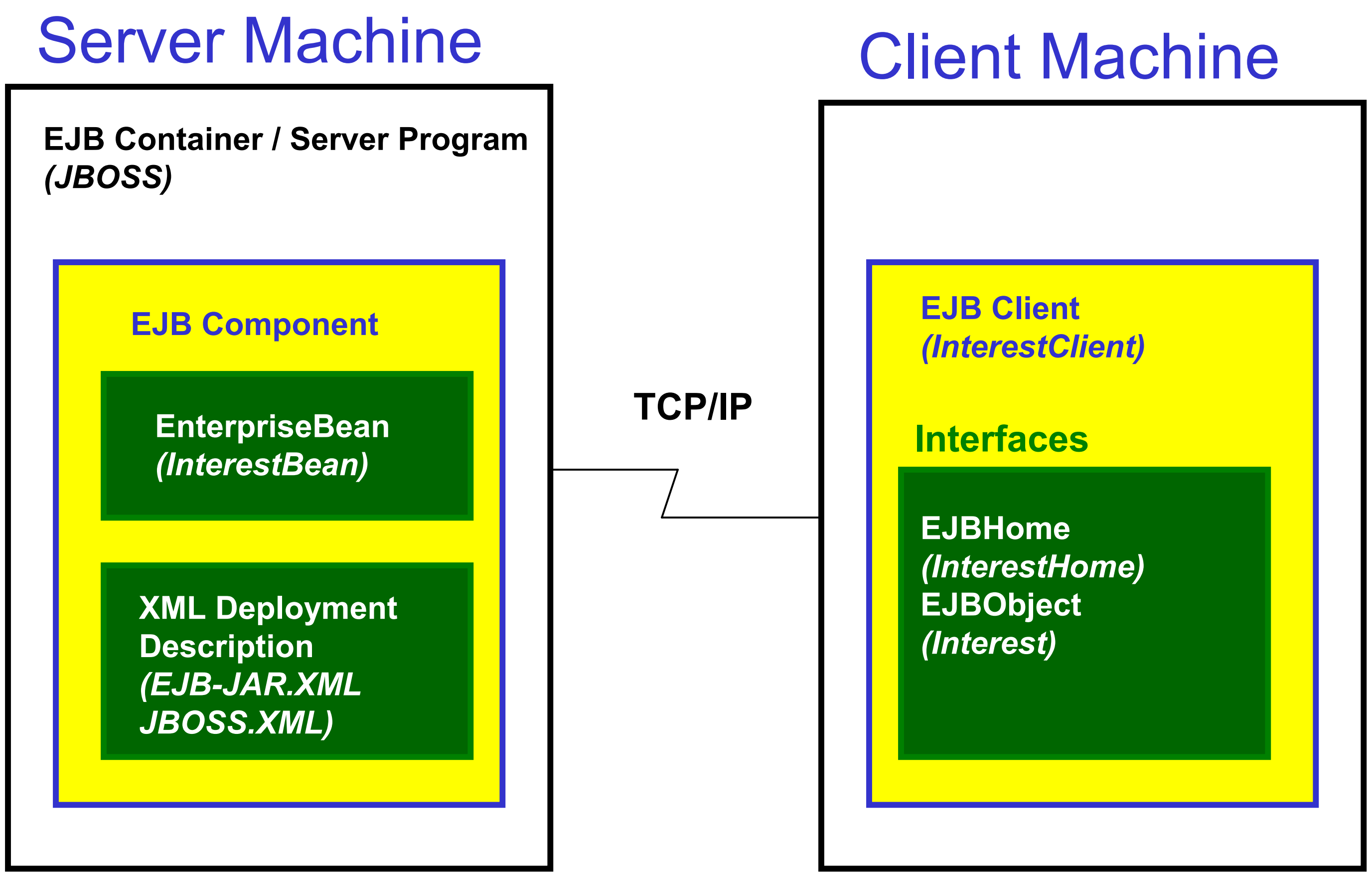
```
screen section.

 01  screen-1.

*    the items in a graphical screen section may have non-integer
*    positions, column and line.   Tthe name of the variable
*   (optional, FILLER otherwise) is followed by the control-type
*   (if graphical), such as label or entry-field.
*   then the property [[=] value]  clauses follow.

     03  label "PERCobol Graphical Sample",
          line 1.5, column 21, size 25,
          font large-font, center.
      03  frame, rimmed, font small-font
          line 4, column 4, size 32, lines 9.
       03  label, title frame-text, font small-font,
            line 5, column 5, size 30, lines 7.

      03  label "&Entry Field", line 14, column 5.

      03  entry-field, using entry-data-1
           column + 2, 3-d.

      03  label "&Multi-line Entry Field",
           line + 3, column 5, cline + 2.

      03  entry-field, using multiple entry-table
           line + 1.5, cline + 1, column 8, size 50, lines 5,
           max-lines = 20, vscroll-bar, 3-d,
           no-autosel, use-return.

      03  check-box "&Check-box",
           using check-box-data, line 5, column 38.

      03  frame, lowered,
           line + 1.5, column 37,
           lines 3, csize 28, size 26.
```

```
*   radio-button-data is assigned the group-value of the
*   selected radio-button.

    03  radio-button, "&Left"
         using radio-button-data,
         line + 1, column 38, group-value = 1.

    03  radio-button, "&Right"
         using radio-button-data,
         column + 3, group-value = 2.
   03  label "&Combo-Box" line + 2.5, column 38.

   03  combo-1, combo-box using combo-data
         line + 1.5, column 39, lines 5, size 16, 3-d.

   03  push-button, "E&xit Program",
         ok-button, line 25, cline 23, column 27, size 13.
```

Object Oriented COBOL

```
CLASS-ID. name INHERITS "class" IMPLEMENTS "class" ... .
        METHOD-ID. name.
            DATA ...
            PROCEDURE...
        END METHOD.
        METHOD-ID. name.
        END METHOD.
        METHOD-ID. name.
        END METHOD.
        ...
END CLASS.
```

Enterprise JavaBeans



Home Interface

The home interface describes the creation method.

```
* This example is coded for the JBoss EJB Container.  The naming
* provider and setup information may differ for other EJB Containers.
* The logic flow remains the same. (http://www.jboss.org)
*
* This example requires J2EE.  JDK 1.2+ plus JBoss is sufficient.
*
* The CLASSPATH to compile must include \jboss\lib\ext\ejb.jar in
* addition to the percobol.jar required for any Cobol program.  This
* is the javax.ejb.* package; its location may vary in other EJB
* vendors.
*
* Set the package name; this could be done from the command line
* using the '-package name' directive instead.  Setting a package
* is generally not necessary for an EJB, but it's good practice.

$SET PACKAGE com.web_tomorrow.interest

*
* An INTERFACE-ID is the same as a CLASS-ID, but without any
* method definitions.  No special-names, no data other than
* linkage, and nothing in the procedure division other than
* the procedure division using is allowed.  This defines a
* contract which another class-id program may implement.
*

IDENTIFICATION DIVISION.
INTERFACE-ID. "InterestHome" INHERITS "javax.ejb.EJBHome".

*
* Only a create method is necessary to define for this interface,
* allowing the client to create the Enterprise JavaBean.
*

METHOD-ID. "create" THROWS "java.rmi.RemoteException",
"javax.ejb.CreateException".
DATA DIVISION.
LINKAGE SECTION.
    77 result OBJECT REFERENCE "com.web_tomorrow.interest.Interest".
PROCEDURE DIVISION GIVING result.
END-METHOD.

END-INTERFACE.
```

EJB Class (part 1)

The Bean class is the only one that does any real work in this example.

```
* This example is coded for the JBoss EJB Container.  The naming
* provider and setup information may differ for other EJB Containers.
* The logic flow remains the same. (http://www.jboss.org)
*
* This example requires J2EE.  JDK 1.2+ plus JBoss is sufficient.
*
* The CLASSPATH to compile must include \jboss\lib\ext\ejb.jar in
* addition to the percobol.jar required for any Cobol program.  This
* is the javax.ejb.* package; its location may vary in other EJB
* vendors.
*
* Set the package name; this could be done from the command line
* using the '-package name' directive instead.  Setting a package
* is generally not necessary for an EJB, but it's good practice.

$SET PACKAGE com.web_tomorrow.interest

*
* This class-id program is the main Enterprise JavaBean.  It contains
* the business logic which is exposed to the outside world and enough
* hooks to allow the EJB Container to control the bean.
**
* A SessionBean is a particular type of EJB; this is the type most
* corresponding to a CICS transaction and the type most Cobol programs
* will implement.
*
* A SessionBean is a logic bean; an EntityBean is a data bean.

IDENTIFICATION DIVISION.
CLASS-ID. "InterestBean" IMPLEMENTS "javax.ejb.SessionBean".

* IDENTIFICATION DIVISION. is optional for a method-id.

METHOD-ID. "calculateCompoundInterest".

DATA DIVISION.
LINKAGE SECTION.
    01 principle COMP-2.
    01 rate      COMP-2.
    01 periods   COMP-2.
    01 result    COMP-2.
```

EJB Class (part 2)

```
PROCEDURE DIVISION USING BY VALUE principle, rate, periods GIVING result.
MAIN-PARAGRAPH.

* The DISPLAY UPON SYSOUT goes to the main log file of the EJB Container.
* This may just be printed on the EJB Container sysout.  It will _not_
* be visible to the client.  Only the GIVING result piece will be
* returned and made visible to the client.

    DISPLAY "Someone called 'calculateCompoundInterest' in PERCobol!"
        UPON SYSOUT
    DISPLAY " principle=" principle
        UPON SYSOUT
    DISPLAY " rate     =" rate
        UPON SYSOUT
    DISPLAY " periods  =" periods
        UPON SYSOUT

* There are other ways of computing this, but this example demonstrates
* the similarities between the Java and Cobol implementations of
* EJBs.

    COMPUTE result = principle * ((1+rate) ** periods) - principle

    DISPLAY " result   =" result
        UPON SYSOUT
    DISPLAY " formula is principle*((1+rate)**periods)-principle"
        UPON SYSOUT
    .

END-METHOD.
```

EJB Class (part 3)

```
* Many EJB's can just copy the remaining code into their code.
*
* All remaining methods in this class are structural methods,
* necessary not for the business logic but rather for the EJB
* Container to be able to control this bean.
*
* Add in some DISPLAY UPON SYSOUT's to the procedure division
* of the methods in order to gain some feel over when these
* methods are called.
*
* We are given the session context, but we don't need it in this
* program so we ignore it.  This method is required to fulfill
* the interface of javax.ejb.SessionBean.
*

METHOD-ID. "setSessionContext".
DATA DIVISION.
LINKAGE SECTION.
    01 sc OBJECT REFERENCE "javax.ejb.SessionContext".

PROCEDURE DIVISION USING BY VALUE sc.
END-METHOD.

* The following methods are required to fulfill the interface
* of javax.ejb.SessionBean.  We don't need to do anything special,
* though, so we just create the method without a body.  When
* the method has no parameters and no result, this is the barest
* possible method definition.

METHOD-ID. "ejbCreate". END-METHOD.
METHOD-ID. "ejbRemove". END-METHOD.
METHOD-ID. "ejbActivate". END-METHOD.
METHOD-ID. "ejbPassivate". END-METHOD.

END-CLASS.
```

EJB Client (part 1)

```
* Enterprise JavaBean Client Sample
*
* for the JBOSS EJB Container.
*
* The original Java version of this EJB client is available at:
* http://www.jboss.org/documentation/jboss_win32_5.html
*
* This simple application tests the `Interest' Enterprise JavaBean which is
* implemented in the package `com.web_tomorrow.interest'. For this to work,
*  the Bean must be deployed on an EJB server.
*
* IMPORTANT If you want to test this in a real client-server
* configuration, this class goes on the client; the URL of the naming provider
* specifed in the class must be changed from `localhost:1099' to the URL of
* the naming service on the server.
*
* Note: In Cobol, this program may also be used as a servlet if compiling
* with the -servlet flag, or using the com.legacyj.run.servlet as the
* name of the servlet with the servlet parameter 'servlet' pointing
* to interest_client.  The only differences for servlets would be
* to add some additional DISPLAYs of HTML elements surrounding the
* text, such as DISPLAY "<HTML><HEAD><TITLE>EJB Client</TITLE></HEAD>"...
*
* The CLASSPATH must include ejb.jar (found in \jboss\lib\ext\ejb.jar
* and other J2EE implementations), the server-side classes (only
* the pieces actually referenced), and percobol.jar (already setup)
* for compilation.

IDENTIFICATION DIVISION.
PROGRAM-ID. interest-client.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.

* All classes are defined in either the CONFIGURATION SECTION/
* ENVIRONMENT DIVISION/REPOSITORY, or in the DATA DIVISION/CLASS-CONTROL.
* The format of either is identical.
*
  CLASS InitialContext IS "javax.naming.InitialContext"
  CLASS InterestHome IS "com.web_tomorrow.interest.InterestHome"
  CLASS InterestClass IS "com.web_tomorrow.interest.Interest"
  CLASS PortableRemoteObject IS "javax.rmi.PortableRemoteObject"
  .
```

An EJB Client can be any number of types of program. It can be an application, a server, a servlet, an applet, a JSP page, or another EJB Component. EJB Components communicate with each other as if they were clients; by loosely connecting components in this manner, the EJB Components are free to be fully controlled by the EJB Container.

EJB Client (part 2)

```
DATA DIVISION.

* In the Java original, the variable declarations are intermixed with
* the procedural code.
*

WORKING-STORAGE SECTION.

* Declare the variables actually used by the program.  The COMPOUND-INTEREST
* was originally done in a different format for Java, but the INVOKE GIVING
* may reference a traditional Cobol variable, allowing a more appropriate
* display format to be chosen, such as the numeric-edited item below.

       77 compoundInterest PIC $$$$,$$$,$$$.99.

       77 jndiContext OBJECT REFERENCE InitialContext.
       77 home OBJECT REFERENCE InterestHome.
       77 ref OBJECT REFERENCE.
       77 interest OBJECT REFERENCE InterestClass.
```

EJB Client (part 3)

```
PROCEDURE DIVISION.

* This method does all the work. It creates an instance of the
* Interest EJB on the EJB server, and calls its
•  `calculateCompoundInterest' method, then prints the result of
•  the calculation.

MAIN.
       SET ENVIRONMENT "java.naming.factory.initial"
           TO          "org.jnp.interfaces.NamingContextFactory"

* Set up the naming provider; this may not always be necessary,
* depending on how your Java system is configured.

       SET ENVIRONMENT "java.naming.provider.url"
           TO          "localhost:1099"

* Java Note:
* Enclosing the whole process in a single `try' block is not an
* ideal way to do exception handling, but I don't want to clutter
* the program up with catch blocks

* Cobol Note:
* No TRY block is necessary for this.  Rather, each INVOKE is
* automatically safe, catching its own exceptions.  If you want
•  notification when an invoke fails, code the individual ON
•  EXCEPTION clause for the INVOKE.

* Get a naming context
       INVOKE InitialContext GIVING jndiContext
       ON EXCEPTION
           DISPLAY "Could not create InitialContext."
           GOBACK
       END-INVOKE

       DISPLAY "Got context"

* Get a reference to the Interest Bean
       INVOKE jndiContext "lookup"
           USING BY VALUE "interest/Interest" GIVING ref
       ON EXCEPTION
           DISPLAY "Could not lookup interest/Interest"
           GOBACK
       END-INVOKE

* Note that if you did not use jboss.xml to overwrite JNDI
* naming the object will be available under "Interest" its
•  ejb-name INVOKE jndiContext "lookup" USING BY VALUE
•  "Interest" GIVING ref

       DISPLAY "Got reference"

* Get a reference from this to the Bean's Home interface

       INVOKE PortableRemoteObject "narrow"
           USING BY VALUE ref InterestHome GIVING home
       ON EXCEPTION
           DISPLAY "Could not narrow."
           GOBACK
       END-INVOKE

* Create an Interest object from the Home interface
       INVOKE home "create" GIVING interest
       ON EXCEPTION
           DISPLAY "Could not create home."
           GOBACK
       END-INVOKE

•  call the calculateCompoundInterest() method to do the
•  calculation

       INVOKE interest "calculateCompoundInterest"
           USING BY VALUE 1000 0.10 2
           GIVING compoundInterest
       ON EXCEPTION
           DISPLAY "Could not calculateCompoundInterest"
           GOBACK
       END-INVOKE

       DISPLAY "Interest on 1000 units, at 10% per period, "
             & "compounded over 2 periods is:" compoundInterest
       .
```

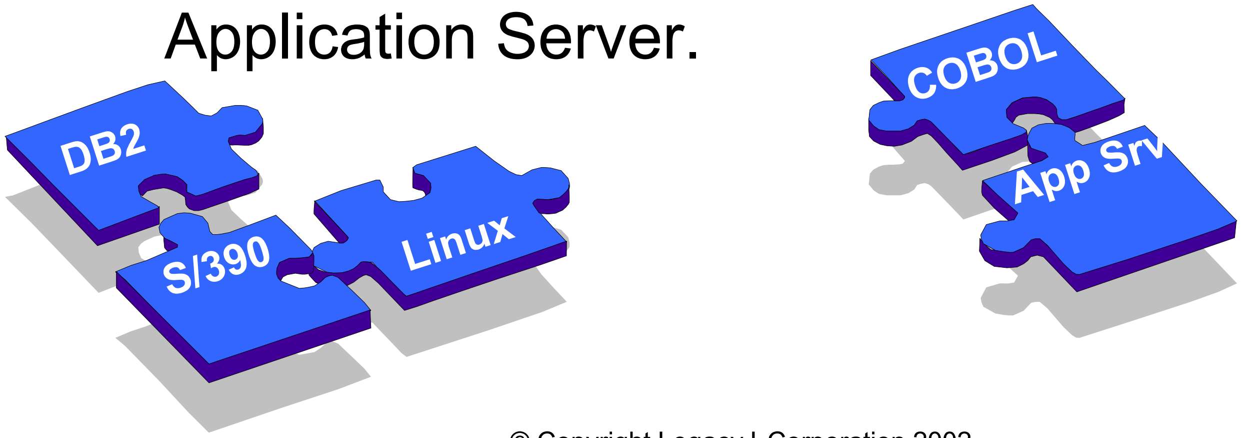
EJB Execution Process

- Copy EJB Component to server deploy directory
- Start Server
- Start Client

Case Study #1

PERCobol used to deploy COBOL from HP-UX and Oracle to S/390 Linux with DB2 connections, by a Brazilian Bank

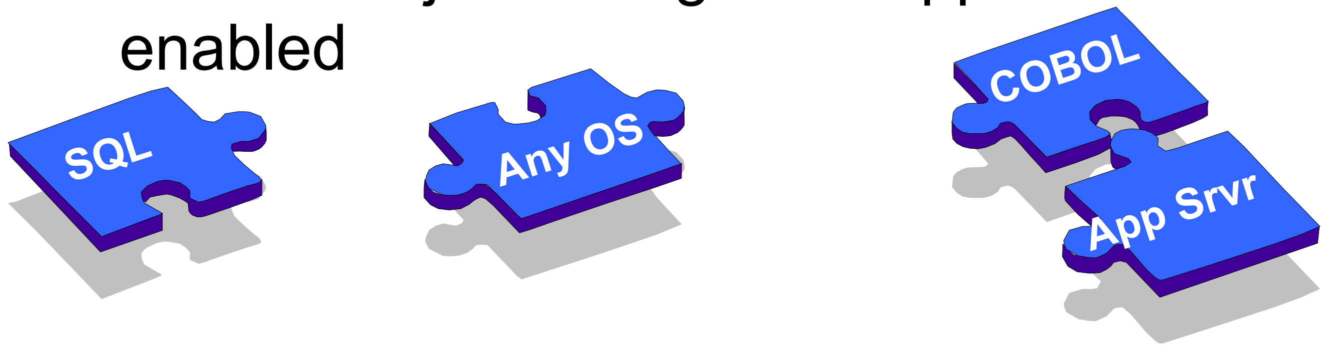
- PERCobol only COBOL compiler supporting IBM COBOL syntax on S/390 Linux with connectivity to DB2.
- Result: Migrated COBOL for deployment on Application Server.



Case Study #2

Migrate S/390 CICS COBOL application for platform independence, and App-Server execution, for California Court System

- PERCobol compiles IBM COBOL syntax and produces J2EE compliant EJBs
- Execution is platform independent, running on Compaq Server H/W, with Oracle
- Result: Major Savings and application server enabled



Case Study # 3

Mainframe CICS COBOL application migrated to App-Server for platform independence, and integrated with Java components, for web enablement while maintaining DB2 connection.

- CICS replaced with J2EE compliant Application Server.
- Result: CICS independent COBOL application with dramatic reduction in cost.