Solutions

Maximize the value of your IT investments with Veryant

Veryant offers software, services, solutions, and strong alliances to help your business meet its goals. Select a subject area below to learn more.

COBOL application and data migration

Move COBOL applications and legacy data to powerful and cost-effective Veryant COBOL offerings

- Read about our COBOL migration services

Java integration

Deliver powerful, flexible business solutions by seamlessly integrating your COBOL and Java code

- Learn more about Java and COBOL integration
- Read about Java Servlets and isCOBOL Evolve

Web enablement

Quickly develop or expand an online presence with multiple COBOL Web enablement options

- Learn about Web enablement options for isCOBOL Evolve

COBOL Migration Services

Quickly and efficiently migrate COBOL applications to Veryant

Veryant's innovative isCOBOL Evolve and vCOBOL Enterprise offerings are designed to support your current COBOL language syntax and features and are highly compatible with today's common COBOL dialects. The transition to Veryant COBOL technology is generally as easy as a version upgrade of your current COBOL development and deployment tools.

- When considering a move to Veryant, we can help you assess the level of effort required. After completing a brief technical survey, we can discuss the best approach to move to isCOBOL or vCOBOL software with you and your team.
- Veryant can also work with you, if desired, to build a limited representative prototype of your application running in a Veryant environment. In addition to experiencing a sample of your code executing on the isCOBOL or vCOBOL platform, this exercise includes a thorough analysis that determines how straightforward the transition to Veryant will be. Your recompiled code and application prototype are then delivered to you, along with software evaluation licenses.

Data migration tools

Moving data sources to a Veryant environment is made easy with feature-rich management utilities such as isMIGRATE that are included with the a Veryant Development system. isMIGRATE provides for one-step migration of ISAM data files to Veryant JISAM.

If you want to access an RDBMS such as Oracle or MySQL you can use existing ESQL code or the Veryant ESQL Generator from Veryant to generate an isCOBOL or vCOBOL file system interface implemented with embedded SQL in COBOL.

COBOL and Java Integration

Add the power and flexibility of the Java platform to existing COBOL programs and make COBOL investments easily accessible to the Java world with isCOBOL Evolve

Add Java to COBOL

Developers leveraging isCOBOL Evolve can quickly add a wealth of new functionality to existing applications by integrating modular Java solutions, such as those found in the Java SE and Java EE Development Kits as well as various open source communities, with their COBOL source code.
Since the isCOBOL Compiler generates pure Java output, once a desired third-party Java solution has been identified, a developer need only write the COBOL code required to interface with it. isCOBOL software allows COBOL user-words to reference Java classes and automatically converts Java data types to the COBOL type that will work in a particular statement, simplifying the integration process.

- [See below]e for more information and code samples

Expose COBOL to Java

Developers looking to quickly expose COBOL applications to the world of Java can use the isCOBOL platform to write programs in COBOL that can be called directly from Java as if they were written in the Java language.
The isCOBOL Compiler automatically converts existing COBOL programs into POJOs (Plain Old Java Objects). No COBOL code changes required. The Java developer receiving the objects needs only to know the class and method names, parameters, and return values to make use of them. The fact that the object class was written in COBOL or that it calls other COBOL subprograms is inconsequential.

- Click here for more information and code samples

How do you take your Java?

A little dash or a splash of COBOL can make for the perfect mix

isCOBOL allows programmers to take advantage of the best of both COBOL and Java by providing common ground between the two technologies. With isCOBOL, organizations can add the power and flexibility of the Java platform to existing COBOL programs and also make COBOL investments easily accessible to the Java world. This article briefly examines some of the synergies that exist between COBOL and Java technology and includes a few examples to illustrate how COBOL and Java programmers can incorporate each others technologies with the isCOBOL Application Platform Suite.

Common Business Oriented Language – 50 Years Young

One of the original design goals for COBOL was to offer a programming language that would be easy to read and understand by managers with no programming training. Although managers rarely took advantage of that -- either fifty years ago or today -- a direct outcome of this goal is that the COBOL language is comprised of familiar English building block structures such as verbs, clauses, sentences, sections, and divisions. The Environment Division, for instance, is a set area found in all COBOL programs that describes the particular environment for which the program is written and all external references, such as to devices, files, currency symbols, and object classes that a COBOL application requires. A COBOL developer looking to move a program to a new platform or region knows that this centralized area of code will have to be reviewed and potentially altered to accommodate the new target platform's requirements. COBOL's straightforward programming approach, clear organizational structure, ability to deal with business logic, adoption of latest technological paradigms, and portability are some of the primary factors that contributed to its widespread adoption and persistence in the industry.

Originally called Green, then changed to Oak, finally known as Java – 15 years mature

The Java programming language leverages small, modular building blocks (known as object classes) that are tied together to form a complete application. Two important concepts in object-oriented programming are encapsulation and inheritance:

- *Encapsulation* means that a Java class can be viewed as a "black box" whose internal workings are hidden. To use a class it is only necessary to know what the class does, not the details of how the class does it. As long as the public interface stays the same, the internal mechanisms of a class can be improved or the class can be replaced with a different one without impact on other components.

- *Inheritance* means that new classes can be formed using classes that have already been defined, leveraging and extending their functionality.

Object classes can be easily assembled, disassembled, and reused in new applications as business requirements change. Since Java programs run in any environment that supports the proper Java Runtime Environment (JRE) they are extremely portable and can be moved between platforms without any program code changes.
It is this modularity, portability, and reusability that contribute to the growing adoption of Java programming in the industry today.

COBOL with a dash of Java

With isCOBOL APS, developers can quickly add a wealth of new functionality to COBOL applications by integrating free, reusable, modular solutions found in the Java SE and Java EE Development Kits as well as in various open source communities such as The Apache Software Foundation and SourceForge. Solutions available to Java developers are now equally available to COBOL developers -- the challenge is only to identify the existing Java solution needed for a particular programming task and to write the COBOL code to interface with it.
It is important to emphasize that to integrate these Java solutions, a COBOL programmer does not need to be trained and experienced in the Java language. The programmer's only requirements are to be able to understand basic object-oriented concepts such as classes and methods, to know how to create an object and invoke a method in COBOL, and to understand how to read and use the Javadocs for the particular Java class to be used. These topics are covered in the isCOBOL APS Language Reference Manual and any beginner's book or online resource on Java programming.
As an example of the new functionality isCOBOL makes available to the COBOL developer, consider the full array of XML APIs and tools available to a Java programmer. To parse local XML files or XML from a URL, a COBOL developer could use the JDOM Java package. Here is a simple program that retrieves an XML document from a URL and parses the XML using JDOM.

```
program-id. XMLfromURL.

configuration section.
repository.
   class J_Iterator      as "java.util.Iterator"
   class J_Element       as "org.jdom.Element"
   class J_SAXBuilder    as "org.jdom.input.SAXBuilder"
   class J_Document      as "org.jdom.Document"
   class J_URL           as "java.net.URL"
   .

working-storage section.
77  W_SAXBuilder              object reference J_SAXBuilder.
77  W_Document                object reference J_Document.
77  W_Element                 object reference J_Element.
77  W_Iterator                object reference J_Iterator.
77  xml                       pic x any length.
```

```
procedure division.
main.
   try
   move "file:///C:/Program%20Files/Veryant/isCOBOL2008/sample
-          "/issamples/files/Members.xml" to xml
  set W_SAXBuilder to J_SAXBuilder:: "new"
  set W_Document to W_SAXBuilder:: "build"(J_URL:: "new"(xml))
  set W_Element to W_Document:: "getRootElement"
  set W_Iterator to W_Element:: "getChildren":: "iterator"
     perform until not W_Iterator:: "hasNext"
        set W_Element to W_Iterator:: "next" as J_Element
        display W_Element:: "getChildText"( "first_name")
                  " " W_Element:: "getChildText"( "name")
                  " = " W_Element:: "getChildText"( "age")

     end-perform
   catch exception
     display exception-object
   end-try.

stop run.
```

Developers can also take advantage of Java interfaces offered by third-party commercial software providers. For example, to add an electronic payment processing feature, a business could choose an electronic payment service provider and request the Java interface. The COBOL developer could use that Java interface directly from COBOL.

Java with a splash of COBOL

With isCOBOL APS, developers can now write programs in COBOL that can be called directly from Java as if they were written in the Java language. Java developers do not need to learn COBOL in order to make use of COBOL assets.
COBOL developers can present their subprograms as POJOs (Plain Old Java Objects). The Java developer receiving the objects needs only to know the class and method names, parameters, and return values to make use of them. The fact that the object class was written in COBOL or that it calls other COBOL subprograms is inconsequential. The isCOBOL Compiler automatically converts existing COBOL programs into POJOs. No COBOL code changes are required. The COBOL program name becomes the Java class name. The resulting Java class has a "call" method which Java code can invoke to call the COBOL program. If required, the COBOL developer can precisely define the program's object interface using object-oriented COBOL syntax. This COBOL object can accept and return Java data types to make the Java developer's task even easier. For instance, if an organization is looking to integrate with a JEE server and needs to deploy a Servlet, Web Service, or Enterprise JavaBean (EJB), etc. those can all now be created natively in COBOL.

So what does this look like on the ground level to a developer? Let's look at a few examples.

COBOL and Java Code Examples

Example 1 – COBOL calling Java
Here is the source code for a COBOL program that uses object-oriented COBOL syntax to access a Java class "java.lang.System" and use a Java data type "java.lang.String".

```
PROGRAM-ID. obj-system.
CONFIGURATION SECTION.
REPOSITORY.
   class Sys as "java.lang.System"
   class JString as "java.lang.String"
   .
WORKING-STORAGE SECTION.
77  pic-x-item  pic x(50).
* There are two styles for specifying classes in object references
77  jstring1    object reference "java.lang.String".
77  jstring2    object reference JString.
PROCEDURE DIVISION.
main.
* There are 3 styles of invoking an object method
* Use the INVOKE statement
   invoke Sys "getProperty" using "os.name" giving pic-x-item.
   display "Operating System: " pic-x-item.
* Use the SET statement with the double-colon operator
   set jstring1 to Sys::"getProperty" ( "os.arch" ).
   display "OS Architecture: " jstring1.
* Use the SET statement with the colon-greater-than operator
   set jstring2 to Sys:>getProperty ( "java.version" ).
   display "Java Version: " jstring2.
   goback.
```

This example demonstrates the use of the repository paragraph to define COBOL user-words that can be used to reference Java classes. It is also possible to specify the full Java class name when declaring a variable as an object reference. This example shows 3 styles of invoking methods. It shows the ability to use Java data types in COBOL statements taking advantage of the isCOBOL Framework to automatically convert the Java data type to the COBOL type that will work in that statement. This is just one example, any Java class and any Java data type can be used.

Example 2 - Java calling COBOL

COBOL developers can write programs in COBOL that can be called directly from Java as if they were written in the Java language. Here is the source code for an object class "isobject" written in COBOL.

```
IDENTIFICATION DIVISION.
CLASS-ID. isobj as "isobject".

IDENTIFICATION DIVISION.
FACTORY.

CONFIGURATION SECTION.
REPOSITORY.
   class jint as "int".

PROCEDURE DIVISION.

IDENTIFICATION DIVISION.
METHOD-ID. method1 as "add1".
WORKING-STORAGE SECTION.
77  var1     pic 9(9).
77  var2     pic 9(9).
77  result   object reference jint.
LINKAGE SECTION.
77  buffer object reference "java.lang.String".
77  num    object reference jint.
procedure division using buffer num returning result.
MAIN.
   display "1st parameter: " buffer.
   display "2nd parameter: " num.
   display "> isobject.add1: result = num + 1".
   set var1 to num.
   compute var2 = 1 + var1.
   set result to var2 as int.
   goback.

END METHOD.

END FACTORY.
```

This class has one method "add1" which takes a Java string and a Java integer, displays them to the standard output stream, adds 1 to the integer, and returns the result. The isCOBOL Compiler will output a file named isobject.class which can be used by the Java developer.

And here is the source code for a Java program which uses the COBOL object "isobject" just as if it were written in Java language.

```
public class callCobolObject {
  public static void main( String args[] ) {
    String PAR1 = "BUFFER";
    int PAR2 = 5;
    System.out.println ("I'm calling method isobject.add1 with "+PAR1+", "+PAR2);
    int RESULT = isobject.add1(PAR1, PAR2);
    System.out.println ("I'm back with: "+RESULT);
    System.exit(0);
  }
}
```

Notice the call to isobject.add1(PAR1, PAR2) is simple Java code. The IDE used by the Java developer will assist with code completion. Even the IDE doesn't know that isobject.class was written in COBOL because behind-the-scenes the isCOBOL Compiler converted the COBOL to Java source code.

These are just a few examples of the collaboration and synergy that isCOBOL enables between the COBOL and Java worlds. Making your own perfect mix of COBOL and Java with isCOBOL should be …well, a piece of cake. Enjoy!

[NOTE: this article first appeared in COBOL Magazine | February 9th, 2009 | Issue 2]

Taking Your COBOL to the Web 'Servlet-Style'

Servlets offer a streamlined deployment model over CGI for server-side implementations

isCOBOL Evolve brings the benefits of the Java platform to COBOL applications without rewriting code and retraining programmers. isCOBOL compiled programs are 100% portable and run on any device that supports a Java Virtual Machine. This provides an 'instant' solution for thin client deployments, but what if your business is looking for an even lighter weight approach? If you are looking to expose COBOL business logic directly to a browser or Web Service, deploying COBOL programs as Java servlets provides a convenient way to add custom functionality without changing back- end program code. This article briefly examines some of the differences between using CGI and servlets to deploy COBOL applications to the Web and includes an example to illustrate how COBOL code can be deployed in a servlet environment.
Historically, many businesses elected to use CGI programming to bring COBOL applications to the Web. Although this approach leverages existing back-end business logic, CGI-based solutions result in platform-specific implementations with high overhead that require a lot of code on the front end to pass stateful information within an application.

In addition to Java technology's platform independence and promise of write once, run anywhere, servlets offer several advantages over CGI programs:

- Servlets are persistent. Loaded only once by a Web server, servlets inherit processing state between invocations and use concurrency control. Servlets can maintain services such as database connections between requests.

- Servlets are fast. Servlets handle concurrent requests on multiple threads rather than in multiple processes and remain in memory to be reused when called.

- Servlets are flexible. Platform and Web server independent, servlets can be used with a variety of clients, not just Web browsers.

- Servlets are secure. Because servlets are invoked through a web server, business logic is not directly exposed to the client. In addition, servlets are isolated from each other so that an error in one servlet cannot corrupt any other servlet.

- Servlets are open. Servlets can be used with a variety of client-side and server-side Web programming techniques and languages.

The isCOBOL Runtime Framework goes one step further with servlets by using the concurrency capabilities of servlets to associate a user session with a COBOL thread context. This makes sure that the same instances of COBOL programs get used each time they are called during a particular user session. In other words, COBOL programs called during a particular user session retain their file states and working-storage data between

requests from that user session. If desired, the programmer can cancel the program at any time with the CANCEL statement.

Dish up your COBOL with Servlets

So what specifically does a COBOL programmer used to the CGI approach need to know in order to get some hands-on experience with servlets? The process for developing COBOL servlets can be very similar to that of developing CGI programs.
Here is a simple COBOL servlet developed with isCOBOL that takes a CGI form field, NAME, as input and sends an HTML response with "Hello NAME" as output:

```
*> Sample servlet that takes a NAME field and outputs "Hello <NAME>"
class-id. Sample         as "Sample" inherits HttpServlet.
configuration section.
repository.
class HttpServlet        as "javax.servlet.http.HttpServlet"
class HttpServletRequest  as "javax.servlet.http.HttpServletRequest"
class HttpServletResponse as "javax.servlet.http.HttpServletResponse"
class ServletOutputStream as "javax.servlet.ServletOutputStream"
.
*> Object Methods
identification division.
object.
procedure division.

*> doPost: Invoked when a POST type request occurs
identification division.
method-id. doPost            as "doPost" override.
data division.
working-storage section.
77 name-field              pic x any length.
77 out                     object reference ServletOutputStream.
linkage section.
77 req                     object reference HttpServletRequest.
77 res                     object reference HttpServletResponse.
procedure division using req res.
main.
  try
    set out to res::"getOutputStream"
  catch exception
    goback
  end-try.
  set name-field to req::"getParameter"("NAME").
  try
    out::"println"('<html>' as string)
```

```
    out::"println"('<head>' as string)
    out::"println"('<title>Sample</title>' as string)
    out::"println"('</head>' as string)
    out::"println"('<body>' as string)
    out::"print"('Hello ' as string)
    out::"print"(name-field as string)
    out::"println"('</body>' as string)
    out::"println"('</html>' as string)
  catch exception
    continue
  end-try.
end method.
end object.
```

As you can see from this example, object oriented COBOL (OOCOBOL) syntax is used to make the COBOL servlet class. There are 5 characteristics of this class that are common in servlets designed to replace CGI programs:

1. The servlet class "inherits" javax.servlet.http.HttpServlet which allows it to plug-in to any standard servlet container (i.e. web server that supports servlets)

2. It defines a doPost method override which takes as parameters the HttpServletRequest and HttpServletResponse object references. Servlets can also override the doGet method, or can override both doPost and doGet. These correspond to the GET and POST CGI methods. For example, doPost will be called if the HTML has <FORM method="post" ... >. For more information about get and post see the "Form submission method" section of the W3C HTML specification at http://www.w3.org/TR/html4/interact/forms.html#h-17.13.1.

3. The servlet class invokes the HttpServletResponse getOutputStream method to save the ServletOutputStream object in a variable named "out".

4. It invokes the HttpServletRequest getParameter method to get the value of the CGI field.

5. And finally it invokes the ServletOutputStream print and println methods to send the output back to the client.

Don't let this OOCOBOL scare you. The key parts do not change from servlet to servlet. You can copy/paste the code from the example above and use it again and again without learning additional object programming. Most tasks can be accomplished with just the HttpServletRequest and ServletOutputStream objects. You can put all of this OOCOBOL syntax in the COBOL servlet class alone and have the COBOL servlet class CALL other COBOL subprograms that do not have OOCOBOL syntax in them. Or you can limit the amount of OOCOBOL syntax that is required in the CALLed COBOL subprograms.

If you have an existing CGI program then you can pass in the HttpServletRequest and ServletOutputStream objects and make straightforward changes so you can use the CGI program for the most part as is. For example,

1. Get the CGI fields using the servlet request object getParameter method.

   ```
   set external-form-item-field-1 to
      servlet-request-obj::"getParameter"("FIELD_1")
   set external-form-item-field-2 to
      servlet-request-obj::"getParameter"("FIELD_2")
   ```

2. Change every DISPLAY UPON SYSOUT or DISPLAY external-form-item to code which will construct the output and send it using the output stream object println method.

   ```
   invoke servlet-output-obj "println" using
      html-output-line as string
      on exception continue
   end-invoke
   ```

Of course, prior to deploying servlets, your environment will need to be configured to support servlets. Veryant has tested and recommends the Apache Tomcat 6. For additional information on servlets and deployment guides, email info@veryant.com.

Learn more on the Web

There are also many excellent resources on the Web for learning about Java servlets and CGI programming. Just a few such links have been included below to help provide a foundation of information that can be built upon to learn how to develop and deploy COBOL servlets with isCOBOL Evolve.

- The Java EE 5 Tutorial - Chapter 4 Java Servlet Technology from Sun Microsystems, Inc
  http://java.sun.com/javaee/5/docs/tutorial/doc/bnafd.html

- What Is a Servlet?
  http://java.sun.com/javaee/5/docs/tutorial/doc/bnafe.html

- Writing Service Methods, Getting Information from Requests, Constructing Responses
  http://java.sun.com/javaee/5/docs/tutorial/doc/bnafv.html

COBOL Web Enablement

Enhance and expand your online presence

isCOBOL Evolve offers several options to leverage COBOL applications across the Web.

- Launch a COBOL application from a web browser running on any supported platform without changing source code or recompiling using the thin client processing capabilities offered with isCOBOL Server.
- Leverage AJAX technology with isCOBOL Web Direct to create and run fully interactive, zero client COBOL application GUIs inside a web browser.
- Enhance your GUI look and feel to meet or exceed the expectations of today's online users.
- Seamlessly integrate with JEE application servers, web servers, as well as web-enablement and mobile solutions.

Read more about each of these solution areas below.

Take applications straight to the Web with isCOBOL Application Server

isCOBOL Application Server, available standard with the isCOBOL Runtime Environment, offers a fast, efficient way to open up existing COBOL applications to the Web. With isCOBOL Application Server you can launch a COBOL application from a web browser running on any supported platform such as Windows, MacOS, UNIX® or Linux. This process requires no code changes or recompiling. The only software required on a end-user's machine is the Java Runtime Environment version 1.4.2 or later.

In thin client mode, only the user interface runs on a desktop, your application and data stay under your control on the server. With isCOBOL Server, an application runs exactly the same way across a network or over the Internet as it does when running locally. Java Web Start technology deploys the thin client and launches the COBOL application. The first time a user visits the web page on a particular machine, Java Web Start will ask whether the user trusts the web site, and will then proceed to launch the application in thin client mode. On subsequent visits, the application will launch immediately. There are no additional download, installation or application execution steps required of the user.

To enable this functionality, simple JavaScript is added to a web page to detect the presence of a JRE on the user's machine and download and install a proper version if necessary. Veryant provides clear step-by-step instructions for setting up the web site and Java Web Start to get your COBOL applications quickly to the Web.

Deliver zero-client, fully interactive online COBOL applications with isCOBOL Web Direct 2.0

isCOBOL Web Direct 2.0 leverages AJAX technology to allow you to create and run a fully interactive COBOL application GUI inside a web browser in a "zero" client solution. Zero client in this case meaning that the only technology required by an end-user to view an application is a web browser -- no plug-in, ActiveX, Java or other software needs to be installed on the client machine.

With isCOBOL Web Direct 2.0, the user interface can run on PCs, Macintosh, X Windows, Mobile Devices or any other system that includes a web browser. User login is simplified through single sign-on and state is maintained throughout an entire user session without special programming or an external state server database.

Developers create web applications entirely in COBOL with isCOBOL Web Direct 2.0 by leveraging COBOL graphical screen section syntax and event paragraphs. In addition, developers can use the isCOBOL IDE's screen designer and Web Direct 2.0 launcher features to design and test screens and logic in the web user interface programs without ever leaving the IDE. The application logic, user interface and the event handling are all written with COBOL statements, providing true interactive web applications without having to learn JavaScript, HTML or any other web tools or languages. Since isCOBOL Web Direct 2.0 preserves the main structure of the COBOL application, your application does not have to be split up into standalone CGI programs or web services.

With isCOBOL Web Direct 2.0, applications are hosted centrally using a Software as a Service (SaaS) model. Offer the convenience of single sign-on and take advantage of n-tier architectures and secure transport such as https. From a security perspective, the web server is the only port open in the firewall, and with https the communications are encrypted and secure. From an administrative perspective, the application and data are hosted centrally which makes deploying patches and upgrades to the application transparent. The system is scalable because the back-end servers and instances can be load-balanced and increased if necessary without requiring changes to application code.

Produce 100% portable, modern COBOL application GUIs

The isCOBOL Evolve application GUI is completely portable and can run anywhere -- from mainframe to mobile devices. The portable GUI produced by isCOBOL software runs character-based as well as graphical programs, enabling you to easily modernize the look and feel of your GUI environment to meet the expectations of today’s online users.

isCOBOL Evolve provides support for unique graphical capabilities in COBOL such as:

- Including Java Swing or 3rd party JavaBeans technology graphical controls in screen sections.
- Using HTML (conforming to v3.2 with some 4.0 extensions) instead of plain text in any graphical control that displays text (e.g. TITLE property).
- Displaying graphical controls in grid cells.
- Allowing users to sort a grid by clicking on column headings, or drag-and-drop grid columns to move them with a simple one-line code change.

- Utilizing LM-SCALE layout-manager to automatically adjust application appearance for different screen resolutions and resized windows.
- Allowing users to rename items in-place, similar to how items are renamed in Windows Explorer, using Tree-view control.
- Adding bitmap images for automatic mouse-rollover functionality in push-button, radio-button and check-box controls.
- Displaying animated GIF files.
- Displaying bitmap images in menu-items and combo boxes.
- Providing hints on every control.
- Allowing the ability to combine text labels and images on check-boxes, push-buttons, and radio buttons.

Deliver consumable objects that integrate easily with JEE environments

The isCOBOL Compiler produces true Java classes, so there is no interoperability layer or additional mechanism needed between isCOBOL and Java components. And the isCOBOL Compiler naturally supports Object-Oriented COBOL. This means that COBOL programs can use objects written in the Java language, and also create objects that can be used by Java programmers as if those objects were written using the Java language.

COBOL Language Localization

Take your applications on a quick spin around the world

isCOBOL Evolve allows businesses to easily deliver and maintain localized, multiple-language software distributions. By incorporating Veryant technology such as the isCOBOL language resource feature, organizations can seamlessly distribute multi-language, global programs. This article briefly examines the language localization capabilities included with isCOBOL and offers an example illustrating how programmers can incorporate such technologies within their COBOL applications.

Full support of Unicode, the ability to incorporate Java Swing and standard Web programming techniques using only the COBOL language, and an efficient language resource feature make isCOBOL an ideal platform to develop and distribute applications requiring natural language localization.

Unicode support

isCOBOL software includes full support for Unicode, both as an encoding for COBOL source code and for data processing within COBOL, for example with the PIC N and USAGE NATIONAL syntax elements. By default all strings and usage display items such as PIC X and PIC 9 are encoded in UTF-8. So a Unicode encoded language is supported without requiring any modifications to existing COBOL code.

Java Swing and Web programming

isCOBOL supports multiple languages natively because it makes use of Java Swing and standard Web programming technologies designed for multi-language and international applications. For example, isCOBOL provides graphical user interface (GUI) programming capabilities to COBOL programmers for deployment as a Java Swing based application, using a Java Swing thin client or an AJAX-based Web 2.0 user interface. Java Swing, Servlets, HTML and Javascript technologies all support multiple natural languages.

- isCOBOL supports all of the internationalization features of Java SE and J2EE Web-tier components. See the Java Internationalization FAQ at http://java.sun.com/javase/technologies/core/basic/intl/faq.jsp

- See Supported Locales at http://java.sun.com/javase/6/docs/technotes/guides/intl/locale.doc.html

- For a discussion on entering bidirectional text, see IBM's article "Globalize your On Demand Business - Entering bidirectional text" at http://www-01.ibm.com/software/globalization/guidelines/h1.jsp

Overview of the isCOBOL language resource feature

The isCOBOL language resource feature provides an easy, yet powerful way to localize an application by configuring the text to be displayed for a particular user interface at runtime by selecting it from different languages.

During the initial program initialization, and optionally before calling a subprogram, the isCOBOL Runtime Framework loads a specified language resource file. This file is a Java properties file which contains any number of name=value pairs. A developer specifies where in the program he or she wants these resource values to be used with a letter R followed by the quoted name of the resource. For example,
DISPLAY R"Greeting".

will use a resource named Greeting that has been loaded from the language resource file. If a resource file contains the following:

Greeting=Hello World

Then the program will display:

Hello World

A resource name can be used anywhere in program source code where it is valid to use a string literal.

Getting started with isCOBOL language resource feature

The basic steps required to create a multi-language version of a program using the isCOBOL language feature are:

1. Search for string literals in the program source code
2. Copy each string literal to a text file and remove the quotes
3. Add a resource name followed by an equal sign to the beginning of the line in the text file
4. Replace the original string literal in the source code with the letter R (or lowercase r) followed by the quoted resource name

For example, a developer could replace "Please enter a valid credit card number" with r"ValidCCNumberMsg" and then add the following line to the resource file:
ValidCCNumberMsg=Please enter a valid credit card number

To assist the translator, a comment line can be added before or after each text resource value and describe the usage and/or context. For example,

```
# Program: CCVALIDATE
# Context: Message displayed on line 24 when user has entered an invalid credit card
number
ValidCCNumberMsg=Please enter a valid credit card number
```

The name of the language resource file is constructed from the values of the following framework properties:

iscobol.resource.file
iscobol.resource.country
iscobol.resource.language
iscobol.resource.variant

as follows (square brackets enclose optional elements):
file[_language[_country[variant]]].properties

For example, if iscobol.resource.file=RES and iscobol.resource.language=en then the isCOBOL Runtime Framework will attempt to load a file named RES_en.properties.

The resource file must be located in a directory or jar file listed in the class path.

The above framework properties can be set programmatically using SET ENVIRONMENT leaving off the "iscobol." prefix.

isCOBOL language resource feature in action

The following example uses Hebrew text to illustrate the capabilities of the isCOBOL language resource feature:

שם משתמש:
סיסמה:

It is possible to create a text resource file, RES_iw.properties, containing this text for use with isCOBOL's multilanguage sample program distributed with isCOBOL in the sample\multilanguage folder.

To create the file save the above text into a utf-8 encoded file named "RES_iw.txt" using notepad and then run native2ascii to create a properties file.
native2ascii -encoding utf8 RES_iw.txt RES_iw.properties

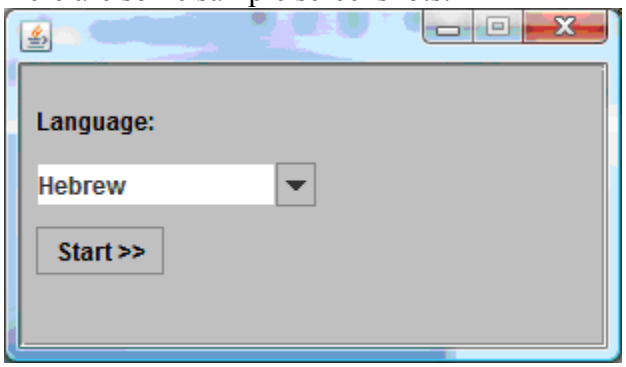
Edit the RES_iw.properties file to add the text resource names such as "usr=", "pwd=", etc, to the beginning of each line, and then reference these text resources in the COBOL program. For example, the resulting text resource properties might have the following:

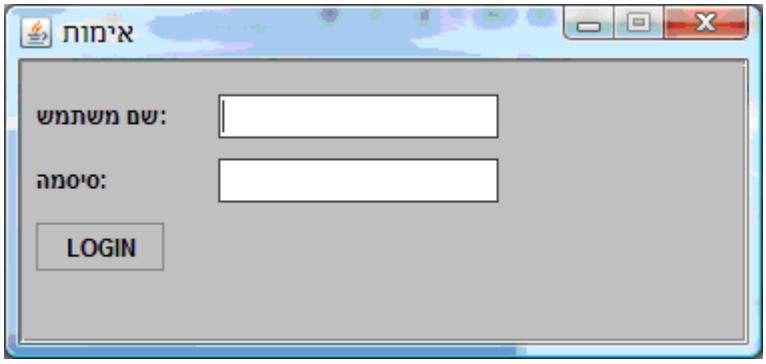
```
usr=:\ufeff\u05e9\u05dd \u05de\u05e9\u05ea\u05de\u05e9
pwd=:\u05e1\u05d9\u05e1\u05de\u05d4
```

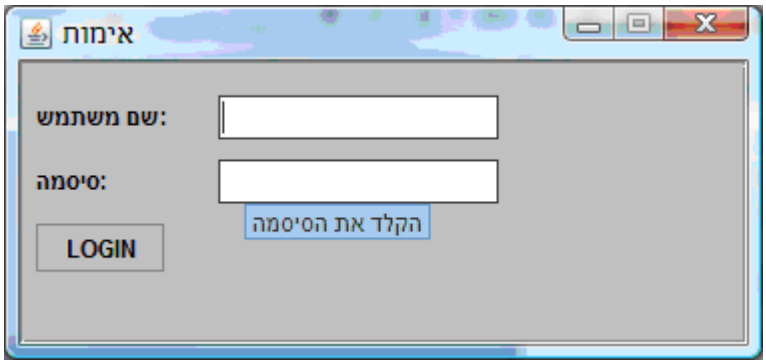
And the COBOL screen section would look like this:

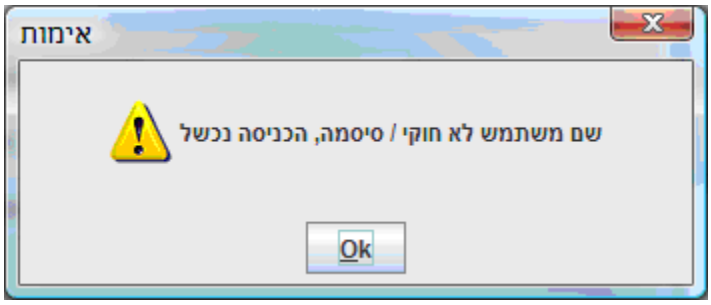
```
01   Login-Screen.
     03  label
         line      2
         col       2
         size      12
         title     r"usr"
         .
     03  label
         line      4
         col       2
         size      12
         title     r"pwd"
         .
```

Here are some sample screenshots:






אימות
שם משתמש:
סיסמה:
הקלד את הסיסמה
LOGIN


אימות
שם משתמש לא חוקי / סיסמה, הכניסה נכשל
Ok