



# Determining interface options to leverage existing System i code

• • • • • • • • •

*Cheryl Renner*  
*ISV Business Strategy and Enablement*  
*December 2006*



## Table of contents

<b>Abstract</b> .....	<b>1</b>
<b>Introduction</b> .....	<b>1</b>
Sample enhancements .....	1
<b>The interfaces</b> .....	<b>2</b>
Call interfaces .....	3
Messaging interfaces .....	3
<b>How to expose these interfaces</b> .....	<b>4</b>
Web services .....	4
J2EE Connector Architecture (JCA) connectors .....	5
WebSphere Business Integration Server Express .....	5
IBM Patterns for e-business .....	6
Ready to go into more details? .....	6
<b>Interface options: The details</b> .....	<b>7</b>
IBM Toolbox for Java .....	7
JTOpen.....	7
IBM Toolbox for Java concepts .....	7
<b>Call interfaces</b> .....	<b>8</b>
Distributed program call.....	8
ProgramCall class .....	8
Additional help for program calls .....	8
ServiceProgram call class .....	8
Calling programs from Java servlets .....	9
Calling programs from EJBs.....	9
Program Call Markup Language .....	9
Program Call wizard .....	10
Program Call wizard's advanced feature: J2EE JCA connector .....	10
Web services wizard.....	10
Program calls using IBM Client Access APIs.....	10
Stored-procedure calls.....	10
Java Native Interface .....	11
Java API.....	11
The java.lang.Runtime.getRuntime().exec() API.....	12
JNI Invocation API .....	12
<b>Messaging interface</b> .....	<b>13</b>
Data queues .....	13
Client Access APIs to access data queues.....	14
WebSphere MQ for iSeries support .....	14
Sockets.....	14
Files .....	15
<b>Triggers</b> .....	<b>15</b>
<b>Interfaces for RPG and COBOL to call Web services</b> .....	<b>15</b>



<b>Interfaces for RPG and COBOL to access Java logic.....</b>	<b>16</b>
Using a messaging interface .....	16
Using a call interface .....	16
RUNJVA or JAVA CL command.....	16
Existing language APIs: QCMDEXC, system() function or RUNJVA.....	17
JNI Invocation interface .....	17
Stored procedure calls .....	17
<b>Summary of all interface options .....</b>	<b>18</b>
<b>Data-passing considerations .....</b>	<b>21</b>
XML.....	21
XML Toolkit for iSeries.....	22
DB2 XML Extenders for iSeries.....	22
XML tools in WebSphere Development Studio Client .....	22
XML support in RPG .....	23
XML support in COBOL .....	23
<b>Data-sharing options .....</b>	<b>23</b>
Data areas .....	23
Users spaces .....	24
<b>Summary.....</b>	<b>24</b>
<b>Resources.....</b>	<b>25</b>
<b>About the author .....</b>	<b>25</b>
<b>Trademarks and special notices.....</b>	<b>26</b>



## Abstract

---

*This paper is designed to help those IBM System i developers who need to modernize their traditional RPG and COBOL applications and leverage the fundamental business logic that intricately reflects the outstanding business rules and processes for the enterprise. The reader will learn the benefits of call interfaces and messaging interfaces.*

## Introduction

---

The industry trend to make applications more extensible involves moving to a tiered application model as compared to the monolithic (all-in-one) architecture of the past. A tiered architecture supports the use of distinct user interfaces that are separate from the business logic, which in turn, is also separated from the data logic. This yields an application that adapts more quickly to technology changes and business needs. The process of separating the application logic and then defining the interfaces between the tiers is a core challenge for traditional IBM® System i™ developers. The IBM System i Developer Road Atlas (now in its fourth version, see [ibm.com/eserver/series/roadmap](http://ibm.com/eserver/series/roadmap)) provides the steps to accelerate developer skills and extend application capabilities. This paper provides more details to help you determine the interface options that support easier enhancement and extension of traditional applications.

### Sample enhancements

In the monolithic architecture, a typical step for extending an application involves adding parameters to an existing 5250 display-driven application. The program-call parameters serve as a vehicle for passing information into the program from another program (as contrasted with sending information to the program from the 5250 display). The developer adds a check into the 5250-driven program so that, if parameters are passed on the call, the display-checking logic is bypassed and the business-logic portion of the code handles the request. After the business-logic runs, another check sends the results back (passed as program-call parameters) instead of sending the results to the screen interface.

A variation on this option is to update the code so that it reads and writes parameters from a data queue instead of using the **execute format** op code to read and write to a display. Many client-server applications are written with a Visual Basic front end that reads and writes to a data queue. In contrast, an RPG or COBOL program listens on a data queue to process the request. It is easy to leverage back-end code through new applications that are based on Microsoft® Windows® .NET or Java™ technologies.

A popular option has long been, and still is, to create stored procedures from existing programs. The stored procedure of the past is called as part of a 5250-based, front-end application or as part of new front-end applications that use ODBC. Now, front-end applications that are written with JDBC or ADO.NET can call those stored procedures.

The creation of service programs provides even more application flexibility and extensibility. Developers can remove the business logic from the 5250-driven program altogether and put it into a service program that exports the input and output parameters. Then, to handle display requests, the 5250-driven program binds in the service program and calls to the business-logic code that resides in the service program.

From a development perspective, it is now possible to enjoy the flexibility of using new code (written in RPG, Java or other languages) to call the same routines in the service program, although information still passes back and forth through parameters. If the new code is RPG-based, it binds to the service program as the original 5250 code did. If the new code is written in Java, a set of classes that are provided in the



no-charge IBM Toolbox for Java licensed program product or JTOpen (the open-source version of the IBM Toolbox for Java) can do the service-program calls. If the new code runs from a Windows desktop, APIs from IBM iSeries™ Access for Windows can do the service-program calls.

Database enhancements are possible at the same time. You can use SQL within programs to handle some of the logic and move the logic closer to the data through triggers.

In many cases, the transaction flows are still code-driven. Although this separation of logic from the UI and data-access method is a step toward a better application architecture, the business logic must still know the state of the transaction (or the state of the interaction with the user).

The evolution of traditional code continues toward a more-multitiered architecture. The business logic becomes more componentized and service-oriented; it handles a request, processes it, then returns a message, a list of messages or a roster of error codes (if applicable). The application evolves to support a UI that drives the code. This is referred to as an “event-driven application.” Overall, the application is more flexible and adaptable to the changing needs of the business.

There is a decision point when separating business logic — whether to use a program interface, procedure-call interface or message interface. The next section explains why and when to use each interface.

## The interfaces

Interfaces are important for a couple of key reasons. First, an inordinate amount of tried-and-proven application code (an estimated 16 to 18 trillion lines of RPG code, alone) requires a huge (and imprudent) effort to toss out and rewrite. Interfaces are also important because the requirements of today’s forward-thinking and competitive enterprises change rapidly; this is also true of the foundational and supporting technologies that support these demanding organizations. Applications must be easily extended and enhanced to accommodate and leverage new environments and changing technologies. Well-defined interfaces make it easier to adapt. For example, an application that is modified to accept requests through electronic data interchange (EDI) can have a Web-service interface that wrappers that transaction request — without changes to the back-end code that handles that request.

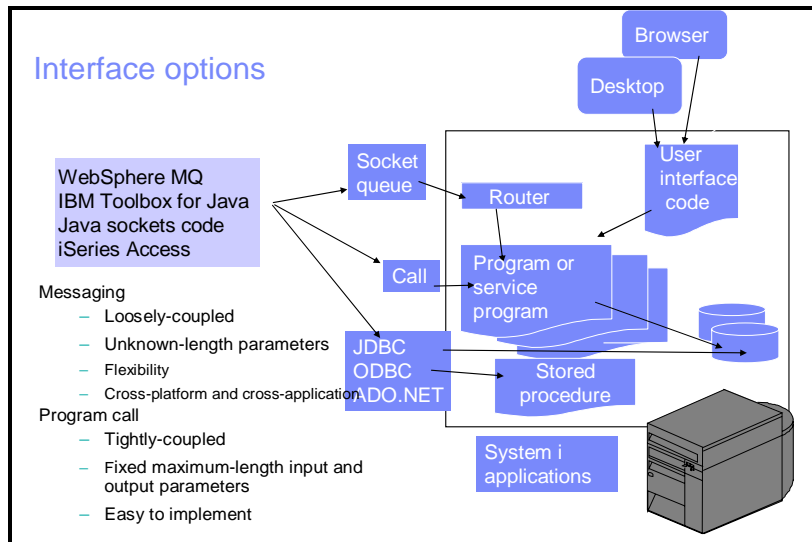


Figure 1. Example of interface options



## Call Interfaces

If the business and data-access logic is already separated into callable programs or service programs, distributed-program calls are easy to implement. For instance:

- Direct calls can be made from iSeries Access for Windows APIs, IBM Toolbox for Java classes, Java APIs or Java Native Invocation (JNI).
- Stored-procedure calls can be made through JDBC, ADO.NET or ODBC.

System i programs or service programs can be called, with or without passing parameters, although it might be necessary to handle the conversion of data types that are passed as parameters. For example, if Java code passes parameters to an RPG or COBOL program, you need to handle conversion of the data that is passed. Fortunately, IBM development tools provide graphical wizards that assist in the creation of program-call and data-conversion code. By their nature, these interfaces are tightly coupled with the existing applications upon which they act. The program or service-program name must be known, as well as the number and type of parameters.

Calls	Ideal usage	Benefits
Program call or remote procedure call	With fixed maximum-length input parameters Tightly coupled applications	Easy to implement because any existing program can be called

Stored procedures are less-tightly coupled and are, practically speaking, an industry standard. It is very easy to create a stored procedure from an existing RPG or COBOL program.

Calls	Ideal usage	Benefits
Stored procedure calls	Can return variable size lists Less-tightly coupled applications	Easy to implement because existing programs can be used to create stored procedures Handles data conversions, supports result sets

## Messaging interfaces

Messaging is the more-sophisticated integration method. For example, using messaging to communicate with existing applications exploits Java thread-and-sockets capabilities. Leveraging queues and sockets can be very easy. If the business has already invested in a client-server architecture with queues or sockets, the code is already in good shape to leverage messaging options for advanced interoperability needs. For instance, it can replace a Microsoft Visual Basic front end with a Java front end. However, implementing messaging “from scratch” requires extra design and programming effort to structure the existing application with distributed objects in mind. Having done this, it is possible to stage the replacement of existing programs with Java or other language code gradually. In essence, messaging provides a much more-loosely coupled approach to interoperability than does a call interface.

This is especially useful for environments that need to connect with vendor and customer applications. This is the connector of choice for cross-platform application communication. See the “Interface options: The details” section of this paper for the various methods for using messaging interfaces.

Messaging	Ideal usage	Benefits
Messaging	Loosely coupled applications, cross-platform application communication, data exchange	Useful in an application, as well as across applications



How can you make a determination regarding which option (a call interface or a message interface) to implement? Should applications be tightly or loosely coupled? These decisions depend on the nature of the application, your skills set as the developer, the requirements for logic reuse and the extent to which the front-end UIs are modified or created. Other influencers include the answers to these questions: How critical is performance? Is scalability important? Is code portability an issue? And, is there a finite, or at least predictable, number of users and transactions?

Different implementations impact performance by adding more code to run. This can be offset because one implementation is easier to program and maintain than another. For example, a socket interface performs faster than a data queue interface that has a Web-service wrapper. There is less code path at run time. Deciding which to use is influenced by considerations regarding its speed of execution, ease of programming and maintenance, as well as the deployment environment. If you are the owner of both the calling and the called program and you need optimal performance, consider using a socket interface or program call. If you own the called program and need to provide an interface to other applications, you can create a Web-service wrapper for your interface that a calling program can then use.

Both a programming interface and a messaging interface let you implement pooling options to improve scalability. Messaging interfaces give added flexibility to let you support interaction between multiple programs by processing requests from queues.

When reviewing your options, take into consideration plans to run the code on another system architecture. Data queues are unique to the System i platform. You might need to modify stored procedures and triggers for use on other platforms. The use of sockets and products such as IBM WebSphere® MQ software are supported across multiple platforms.

As the code moves further toward unknown parameter volumes, the message interface becomes more attractive because of its greater flexibility. Messaging interfaces do not require the caller (that is, the program or procedure that makes the call) to know the name of the program or service program that is called. Instead, the program or procedure puts out a message and handles any return messages.

## How to expose these interfaces

---

The developer might prefer to have a set of connectors that wrappers these interfaces or exposes them to other applications. Various options are available to leverage existing programs and applications from Java applications, as well as to expose newly packaged functions to other applications. This section discusses some of those connectors — what they are and when to use one or another.

**Note:** For details on the differences and usage, see these articles: “Choosing among JCA, JMS and Web services for EAI” ([ibm.com/developerworks/webservices/library/ws-jcajms.html](http://ibm.com/developerworks/webservices/library/ws-jcajms.html)) and “Integrate enterprise applications with Web services and J2EE” ([ibm.com/developerworks/webservices/library/ws-eai/](http://ibm.com/developerworks/webservices/library/ws-eai/)).

### Web services

Web services refer to technologies that provide a language- and environment-neutral programming model to support rapid application integration, inside and outside the enterprise. The result is a set of flexible, loosely coupled business systems, connected by a collection of functions, packaged as a single entity, and published to the network for use by other programs. You can easily use Web services as a wrapping technology around existing applications and information technology assets. Similarly, they are



used to create wrappers for call interfaces or messaging interfaces. You can recompose Web service applications to address new business needs, invoking the concept of just-in-time application building.

**Note:** See the IBM developerWorks® Web services Web site: [ibm.com/developerworks/webservices/](http://ibm.com/developerworks/webservices/). For additional helpful reading, see the IBM Redbook entitled *System i Application Modernization: Building a New Interface to Your Legacy Applications* [SG24-6671-00], which you can find at [ibm.com/redbooks](http://ibm.com/redbooks).

## J2EE Connector Architecture (JCA) connectors

The J2EE Connector Architecture (JCA) establishes a standard methodology for connecting Web-based front ends (that have been designed using J2EE technologies) to application servers that interact with back-office applications. The developer can create a resource adapter for the existing application that can plug into any JCA-compliant application server. A resource adapter is a piece of code that supports an application server to have access to back-end business logic. It acts as a wrapper for the interface that was created for the existing code and works in conjunction with the application server to handle security, connection pooling and other transactional-support needs. For a primer on JCA, visit the Sun J2EE Connector Architecture Web site ([java.sun.com/j2ee/connector/](http://java.sun.com/j2ee/connector/)).

Connectors	Ideal usage	Benefits
JCA	Tightly coupled applications, J2EE application that needs to access existing non-Java code	Useful for transaction support

Java Message Service (JMS) and IBM WebSphere MQ classes for Java provide for the implementation of an industry-standard method for communicating with a message service, essentially providing an object-oriented interface to IBM WebSphere MQ for iSeries.

Connectors	Ideal usage	Benefits
JMS	Loosely coupled applications, use between existing messaging applications	Useful for asynchronous communication

## WebSphere Business Integration Server Express

IBM WebSphere Business Server Express and Express Plus on System i (V4.4) packages many technology adapters for developers to use, including the following: IBM Lotus® Domino®, e-mail, iSeries, JDBC, JMS, Jtext, Swift, Web services, WebSphere MQ and XML. The technology adapters run natively on the System i platform. If a business process needs to interface with an application that uses one of these technologies, it might be possible to use a predefined technology adapter rather than having to create an adapter that is specific to the application. This saves time in connecting with the application. For example, your System i application might be able to use the iSeries technology adapter.

IBM has predefined a set of adapters to interface with common solution-provider business applications. Some of the business-application adapters cannot run natively on the IBM i5/OS® operating system. However, they are packaged with IBM WebSphere Business Integrator and can be installed on the Windows or Linux® operating systems where a business application is installed and runs. To interface with the Windows or Linux application, these adapters call APIs that are provided by the application itself. The adapters that run natively on i5/OS can interface with their corresponding business applications through JDBC. Thus, the application does not have to run on i5/OS, even though the adapter can do so.

The iSeries adapter makes lets you integrate a System i program in a business process. The program must be a callable \*PGM object and can have parameters. When the adapter gains control, it calls the



\*PGM and synchronously passes in input-parameter values. The adapter gets control back from the program after it completes but receives no return-code value or other output-parameter values. An iSeries adapter can only receive a request to call a program; it cannot trigger a business process. Use the JDBC adapter or another adapter if the System i application must trigger a business process.

Other versions of WebSphere Business Integrator provide an iSeries Object Discovery Agent (ODA) and an iSeries Data Queue adapter. These are not currently packaged with WebSphere Business Integrator V4.4 on System i. The ODA is an agent that automatically discovers the list of parameters for the program that an adapter needs to interface effectively with that application. The iSeries Data Queue adapter makes it possible to place data on a queue or read from a queue as the interface approach.

In addition to running the iSeries adapter on System i, you can install the iSeries adapter in Linux or Windows operating environments to connect with the System i platform to interface with a System i program. The adapter uses the Access class and the ProgramCall class in the Toolbox for Java to call the System i program. The Remote Command and Distributed program-call server are used to run the program on the System i platform. The program's parameter values can be double-byte character set (DBCS) and the character conversions are handled automatically.

**Note:** You can find more details on the WebSphere Business Integration Server Express Web site ([ibm.com/software/integration/wbiserverexpress](http://ibm.com/software/integration/wbiserverexpress)).

## IBM Patterns for e-business

Patterns for e-business are a group of reusable assets that leverage the experience of IBM architects to create Web solutions quickly, for both small businesses and large multinational enterprises. These reusable assets break down into seven elements, as follows:

- Business patterns
- Integration patterns
- Composite patterns
- Custom designs
- Application and runtime patterns
- Guidelines for the design, development, deployment and management of e-business applications

**Note:** For details on using these connectors to develop or enhance applications for the Web, go to the IBM Patterns for e-business Web site ([ibm.com/developerworks/patterns](http://ibm.com/developerworks/patterns)). You can also read the IBM Redbooks™ related to the System i platform ([ibm.com/redbooks](http://ibm.com/redbooks)). Some examples include: *Patterns: Self-Service Application Solutions Using WebSphere V5.0 for iSeries* (REDP-3670-00) and *Enabling Web Services for the IBM eServer iSeries Server* (REDP-0192-00).

## Ready to go into more details?

All enterprise IT groups are under a great deal of pressure to accomplish a large number of missions, including moving the business processes to the Web. They must also support SOA and a new desktop, as well as new business-process flows with a quality presence.

Additionally, IT managers must avoid handicapping the company budget or back-office processes. You can read through the details provided in the rest of this document carefully, making sure to highlight unfamiliar technologies for later research on the IBM Web site ([ibm.com](http://ibm.com)). IBM is committed to helping its



clients in all their Web-enablement endeavors, as are the many excellent IBM Business Partners. After finishing this paper, you will have a more-solid understanding of the many options that are cleverly designed to assist your development efforts in extending and enhancing your organization's applications.

## Interface options: The details

---

This section delves into the fundamentals of the interfaces that are most likely to be important as you design interoperability processes between your company's Web or desktop-based client applications and your existing, native System i applications.

### IBM Toolbox for Java

The IBM Toolbox for Java is a library of Java classes that support the client-server and Internet program models on the System i platform. This collection of tools is very relevant when discussing interfaces. You can use classes from this toolbox to interface with the call interfaces as well as the messaging interfaces.

These classes are used in Java applets, servlets and applications and use IBM OS/400® Host Servers (which are a part of i5/OS) to access System i data and resources. Each host server runs in a separate job, talking to Java-client programs through designed data streams on socket connections that are then hidden from the programmer by the toolbox classes. JavaBeans provide processes for most public interfaces. The IBM Toolbox for Java is tested on many platforms, including IBM AIX®, i5/OS, OS/400, Linux, IBM Network Station®, IBM OS/2®, Solaris and Windows. Though generally used in System i applications that are unconcerned with portability, the classes will port to any Java-supported platform.

#### JTOpen

JTOpen is the open-source version of the IBM Toolbox for Java and contains the latest enhancements and fixes. In fact, IBM adds all fixes to JTOpen first, and then later releases them as PTFs to the IBM Toolbox for Java.

#### IBM Toolbox for Java concepts

The classes in this toolbox use the idea of a connection, which is represented by an AS400 object. When writing Java code to run on the System i platform and using the toolbox, the AS400 object defaults to the current System i model. Some toolbox classes, such as ProgramCall, use the AS400 object to connect to an i5/OS Host Server. You can improve connection time by pooling the connections and saving them for later use or by prestarting the jobs for each host server. In some sense, managing connections is similar to managing files in RPG or COBOL programs.

Security classes, such as secure sockets layers (SSL) and authentications services, provide security and allow identity assignment to the current i5/OS thread. The AS400 class uses the `setThreadUsed` method to see if the IBM Toolbox for Java uses threads to communicate with the host servers. IBM Toolbox for Java Systems-properties classes allow properties to be changed at run time.



## Call interfaces

---

This section explores the call interfaces in more detail.

### Distributed program call

You can call any existing System i program or service program object through a set of APIs that are provided in the IBM Toolbox for Java. These APIs are discussed in the following sections:

#### ProgramCall class

The ProgramCall class uses the Remote command and distributed program call server to call System i program objects. This is an easy way to access existing programs. A maximum of 35 parameters are supported for a program, which is a host-server limitation, not a limitation imposed by the IBM Toolbox for Java. You can pass parameters, inversely and repeatedly, between existing System i programs and the Java programs that call into them.

**Example:** If a Java program calls an existing program, a parameter is passed that states whether to retrieve all records (or just one record) in a file or to close the program. When the parameter is set for all records, the RPG code repositions to the start of the file and passes back a successful flag to the Java program. The Java code then passes a parameter for **fetch next**. Both programs continue interacting in this manner until the RPG program passes an end-of-file flag. The file remains open until the program closes. When the Java program is ready to exit, it passes a parameter to close the RPG program. The ProgramCall class uses the Remote command and distributed program call server to access existing code and data on the System i platform.

Because the JVM, Java Software Development Kit (SDK) and IBM Toolbox for Java are all resident in the i5/OS operating system, you can use the ProgramCall classes natively on the System i platform. When running on the server, the ProgramCall class uses a local sockets connection to the host servers to improve performance.

#### Additional help for program calls

As of OS/400 V5R2, the IBM Toolbox for Java contains ready-to-use JavaBeans for the ProgramCall class and a Program parameter. This function lets CommandCall and ProgramCall classes stay on the same thread (optionally) as the application when running on the i5/OS JVM — instead of performing their functions with a call to a server. This option requires the programs to be threadsafe.

#### ServiceProgram call class

This class lets the application make calls to existing procedures from within a service program — to pass data through input parameters and to access data returned through output parameters. Essentially, service programs provide a vehicle for grouping associated procedures in a library-like manner. Here are some limitations:

- The service program must be on a System i model that runs OS/400 V4R4 or later.
- A maximum of seven parameters can be passed to the service program.
- The return value must be void or numeric. There is no support for calling service programs that return a pointer.

- Parameters can be "pass by value" or "pass by reference." If they are pass by reference, data is copied from Java storage to System i storage, and a pointer to System i storage is passed to the service program.

### Calling programs from Java servlets

When an application server receives a request that must be handled by a specific servlet that is not already loaded, the server loads the servlet and runs the `init()` method of the servlet to initialize it. Because the servlet `init()` method runs only once during the life cycle of any servlet, it follows that there is only one instance of the servlet to serve concurrent HTTP requests. This means that when a multithreaded servlet talks to RPG programs or other language programs, you must be careful about synchronization issues.

**Example:** Thread A calls an RPG program to read records sequentially from the database. Thread B starts a session and calls the same RPG program and repositions the cursor to the start of the database. Thread A does a second read and erroneously retrieves the first record again. There are two options for resolving this conflict:

- The RPG program can read all the records and return them in one response. However, this might not be the most efficient way to retrieve the records.
- The servlet can implement the `SingleThreadedModel` interface. This interface is an empty-tag interface that flags the servlet as one that needs to run in a single-thread model. A server that loads a `SingleThreadModel` servlet must guarantee that no two threads run the service method of the servlet concurrently. This guarantee is ensured by maintaining a pool of servlet instances for each servlet and dispatching each service call to an available servlet.

The previous two options work for either a program call or a data queue. However, there is another option when using data queues:

- You can establish a unique ID for each servlet request that can then be written to the input queue along with the rest of the data. The RPG program writes the response to a keyed-output queue where the key is the call ID that is supplied by the servlet in the input queue. The Java program that writes its request to the input queue with the call ID uses this call ID as the key when reading the response from the keyed-output queue.

### Calling programs from EJBs

As mentioned earlier, the AS400 class in the IBM Toolbox for Java has a method called `setThreadUsed`. This method establishes whether the IBM Toolbox for Java uses threads in communication with the host servers and defaults to **true**. Although it is possible to improve performance by allowing the IBM Toolbox for Java to use threads, it might be necessary to turn threads off if the application must be compliant with the EJB specification. The **thread used** property cannot be changed after a connection to the System i platform is established.

### Program Call Markup Language

The Program Call Markup Language (PCML) framework that is provided in the IBM Toolbox for Java is a state-of-the-art tag language and is based on XML. Its program-call framework is designed to make it easier to call an existing program from a Java program. You can use tags to describe fully all



the parameters, structures and field relationships for a given System i program call. The toolbox also provides an API to interpret the PCML, call the existing program on behalf of the Java program, and retrieve the data returned from the System i application easily. This is an alternate, and much less code-intensive method for the ProgramCall class.

### Program Call wizard

To assist you in writing code to call System i programs and to handle conversion of the parameters, it is possible to use Program Call wizards that are included in IBM WebSphere Development Studio Client for iSeries. The Program Call wizard prompts for the name of the program to call, as well as the needed parameters and their data types — after which, the wizard generates the code for calling the program. This wizard uses the ProgramCall class from the IBM Toolbox for Java.

### Program Call wizard's advanced feature: J2EE JCA connector

In the WebSphere Development Studio Client for iSeries Advanced Edition, an option in the Program Call wizard generates code that is JCA-compliant. The advantage of this is that Java and J2EE standards are honored. Thus, for further development purposes, you can deploy other JCA-compliant tools, such as those in IBM WebSphere Studio Application Developer.

### Web services wizard

You can then use the output from the Web-interaction wizard as input to the Web services wizard (found in WebSphere Development Studio Client).

**Note:** When application components are tightly coupled and are based on database records and flat files, they are rigid and sensitive to change. Web services are self-contained, modular applications that can be described, published, located and invoked over the Internet (on a just-in-time basis).

The Web services wizard creates a Web service (by using a JavaBean generated by the Program Call wizard) to call one or more System i server programs or service program procedures.

### Program calls using IBM Client Access APIs

For non-Java programs, you can leverage the program-call APIs in the IBM Client Access toolkit to call existing programs. See the following Web site for sample Client Access programs that use the program-call APIs: [ibm.com/servers/eserver/series/access/toolkit/program\\_call.htm](http://ibm.com/servers/eserver/series/access/toolkit/program_call.htm).

## Stored-procedure calls

Stored procedures are another way to leverage existing code. You can initiate a CREATE PROCEDURE statement on an original program model (OPM) or IBM Integrated Language Environment® (ILE) program to build a stored procedure. This stored procedure can then be used from existing applications and through JDBC, ADO, NET or ODBC. A stored procedure can contain business and database-access code. The Java application can benefit from improved database access through record-level access to the database or static and dynamic SQL that is invoked from within the stored procedure. (**Note:** The ability to use service programs as stored procedures was available with i5/OS V5R3.)

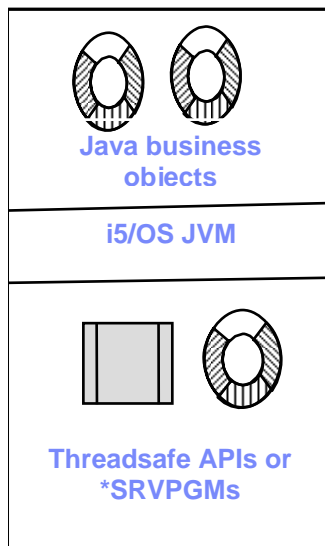
Stored procedures are programs, but you control when they are called; the database does not invoke them automatically. You can write stored procedures in RPG, COBOL or Java, or solely through SQL.

Stored procedures have the ability of returning a result set (that is, an array of output parameters) that none of the other methods can offer. A benefit of using JDBC is that it handles all data conversions on data sent from the stored procedure. It is possible to leverage existing stored procedures or to create them new from existing programs. In either case, existing applications and new Java or Windows applications can use them. The SQLCREATE PROCEDURE statement creates both external stored procedures and SQL stored procedures.

In a client-server environment, stored procedures reduce I/O communication and, thus, improve response time. You can compile a stored procedure, which then uses static SQL. Static SQL is inherently faster than any form of dynamic SQL (including JDBC). This is because the SQL processing and optimization is done at compile time and is then stored with the program object.

**Note:** For details on leveraging stored procedures, see the Redbook *Stored Procedures, Triggers and User Defined Functions on DB2 Universal Database for iSeries* (SG24-6503-01) ([ibm.com/redbooks](http://ibm.com/redbooks)).

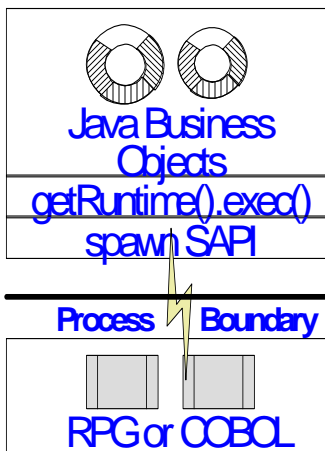
## Java Native Interface



The Java Native Interface (JNI) is a low-level, platform-independent, interface that is designed for ILE/C, ILE/C++, RPG and COBOL. Application writers do not generally make heavy use of JNI because of its complexities and portability issues. You can access native programs by defining a native method in Java and using the JNI functions. When calling a native method in Java, the program is running on a Java thread. Current limitations in RPG and COBOL can limit the scalability of multithreaded applications that use JNI. Threadsafe features must be considered for the programming language, command language (CL) commands, system APIs, service program procedures and all code called by those procedures (**Note:** JNI interfaces that are designed for C on one platform might not work on another platform because of implementation nuances and the potential need to handle data conversions. If used, JNI limits application portability.)

To use JNI with WebSphere-based projects, refer to the specific WebSphere on System i documentation for each release.

## Java API



The advantages of using a Java API to access existing code include: portability of the Java code, ease of coding and avoidance of threads issues (because a new process is being started). This option is useful when the code that is ported to the System i platform already uses a Java API. In this case, you need merely to replace the system function that is called. The Java API for calling another program is `runtime.exec()` — it starts a new job for the called program.

## The `java.lang.Runtime.getRuntime().exec()` API

The `Runtime.getRuntime().exec()` Java API is a standard Java call that initiates a new job and runs the system command in that process. Because a new job is created each time it is used, you must limit the use of this API to avoid any impact on system performance and resource usage. There is no support to get messages back from the system command. Therefore, this method uses input and output streams for interprogram communication: the C language uses `standard.in` (`stdin`) and `standard.out` (`stdout`) functions; RPG and COBOL use the `QtmhWrStin` and `QtmhWrStout` APIs.

Required APIs are available to support the environment variables as well as the standard input and output streams. The C language supports the environment variables through the `getenv()` API. The standard input and output streams are supported through many APIs — for example, `fgets()` and `fwrite()`. For C programs, the APIs are found in `QTMHCGI *SVCPGM`. The H file (in library `QSYSINC`) provides a C-language header-file member named `QTMHCGI`.

The APIs to support the IBM Integrated Language Environment for RPG (ILE RPG) and the IBM Integrated Language Environment for COBOL (ILE COBOL) are provided in source files (`QCBLLSRC` for ILE COBOL and `QRPGLESRC` for ILE RPG, both found in the `QSYINC` library). In RPG, the Create Program (`CRTPGM`) command must always specify `BNDSRVPGM QHTTSPVR/QZHBCGI` to ensure that it satisfies module-import requests for the `QtmhGetEnv`, `QtmhRdStin`, `QtmhWrStout` and `QtmhCvtDB` APIs. These APIs simplify parsing form data either from `stdin` or the `QUERY_STRING` environment variable.

## JNI Invocation API

The JNI Invocation API is a set of six functions that support “imbedding” a JVM in an application. These applications can then interact with the JVM using JNI. However, there are some platform-specific considerations. For instance, the developer must convert parameters that are related to the Invocation API functions from Extended Binary Coded Decimal Interchange Code (EBCDIC) to UTF-8 and conversion of data formats must be handled for passed literals.

The JNI Invocation API is typically used only when porting function from another platform to System i. In other words, The JNI should be used if it is the only way to access certain system functions. If porting C/C++ code that uses the invocation API, be aware of potential changes that might be needed for running on the System i platform (for example, code-page considerations and the way System i handles pointers). The following RPG and COBOL extensions were added as of OS/400 V5R1:

- Extensions that permit RPG and COBOL programs to call Java methods
- Extensions that mask some of the complexity of using JNI

**Note:** The *ILE RPG Programmer's Guide* and *ILE COBOL Programmer's Guide* shows how to code this.

Because a new JVM is created each time JNI is used, you should monitor and limit the use of the JNI Invocation API to ensure that there is no impact to system performance, resource usage and scalability. Java on System i is optimized by moving the JVM below the technology-independent machine interface (TIMI). Therefore, a native-method invocation requires a call to code that is above the TIMI, which can also necessitate performance-expensive JNI calls back into the JVM (below the MI). As already stated in this paper, the simpler, more-maintainable interoperability method, as compared to the JNI Invocation API, is to define interfaces between the Java programs and the existing languages.



## Messaging interface

---

This section will explore in more detail the messaging interfaces available to be used.

### Data queues

Data queues are a mechanism to pass information between programs and are unique to the System i platform. You are probably already familiar with them because they are one of the standards for passing information within client-server applications. You can also use data queues to pass information between programs where both programs reside on a System i platform. The IBM Toolbox for Java supports both regular- and keyed-data queues. The IBM Toolbox for Java provides the `DataQueue` and `KeyedDataQueue` classes to communicate between Java and existing programs. Both of these classes work through the creation of a System i `DataQueue` object and require that some message-passing conventions be established.

If changing code to handle data queues, you must design support for any portability need. If using the toolbox dataqueue classes, the Java program puts out an entry through an interface that is then implemented on the System i platform with a data queue. On other platforms, this can be done with WebSphere MQ or another messaging option. And, although WebSphere MQ is available on the System i platform, data queues (which, as mentioned, are unique to the System i platform) are usually more lightweight for messaging communications.

**Note:** You can find a list of advantages and data-queue examples in the Redbook, *Who Knew You Could Do That with RPG IV?* (SG24-5402-00) ([ibm.com/redbooks](http://ibm.com/redbooks)). Here is an excerpt:

- Data queues are a fast means of asynchronous communication between two jobs — requiring less system resource than using database files, message queues or data areas.
- Using data queues frees a job from performing some work. For interactive jobs, the data queue APIs can provide better response time and decrease the size of the interactive program. For example, if several users enter transactions that involve updating and adding to multiple files, the system performs better if the interactive jobs submit the request.
- Several jobs can receive data from the same data queue. This is beneficial when the number of entries to be processed is more than one job can handle (in desired performance restraints). For example, if multiple printers are available to print orders, several interactive jobs can send requests to a single data queue. A separate job for each printer can receive data from the data queue in first-in-first-out (FIFO), last-in-first-out (LIFO) or keyed-queue order.
- Data queues can attach a sender ID to messages put on a queue. The sender ID, an attribute established at data-queue creation, contains the qualified job name and current user profile.
- When receiving data from a data queue, you can set a time-out, such that the job waits until an entry arrives on the data queue.

When using the IBM Toolbox for Java queue classes, the program connects to host servers through sockets that pass and receive information through the queues. Since OS/400 V4R3, the Java classes have remained on the Java thread and do not use the host-server drivers when the data queue and Java program are on the same server. The data-queue classes do not alter data that is written to, or read from, the System i data queue. The Java program must correctly format the data, which can be accomplished through the data-conversion classes in the IBM Toolbox for Java. You can also use the record-format class in conjunction with the data-conversion classes.

Data-queue “best practices” dictate that you periodically (once a day) delete and re-create the queue at a safe point. Performance can be affected if too many entries exist without being removed. Re-creating the queue returns the data queue to its optimal size.

In addition, code is drastically simplified if there is only one message (each way) per transaction. To build high-performance data-queue applications, the amount of data in each data queue entry must be optimized. Sending a few very-large entries is much more efficient than sending many smaller entries. For example, sending a single-kilobyte data-queue entry is nearly 10 times faster than sending 10 entries that are each 100 bytes. This technique maximizes communications blocking and minimizes related overhead.

You must handle conversion of information to and from the queue (using the IBM Toolbox for Java conversion classes). Data queues simulate a distributed-call model but do not let applications take advantage of threading. The existing code can use the following APIs to work with data queues:

- Send to a Data Queue (QSNDDTAQ)
- Receive from Data Queue (QRCVDTAQ)
- Retrieve Data Queue Message (QMHRDQM)
- Clear Data Queue (QCLRDTAQ)
- Retrieve Data Queue Description (QMHQRDQD)

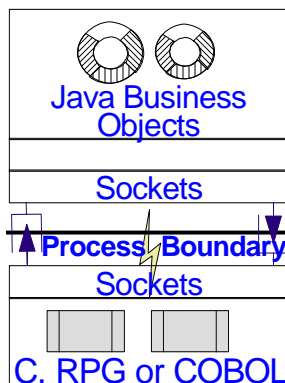
### Client Access APIs to access data queues

For non-Java programs, you can leverage the Data Queue APIs in the Client Access toolkit. These APIs let you put and receive messages on a data queue. (**Note:** See the following Web site for sample programs: [ibm.com/servers/eserver/iseries/access/toolkit/data\\_queues.htm](http://ibm.com/servers/eserver/iseries/access/toolkit/data_queues.htm).)

### WebSphere MQ for iSeries support

WebSphere MQ for iSeries provides the capability to communicate between programs on the same or other platforms using a single messaging product. This queue support is more robust than the use of i5/OS data queues and provides greater portability, but does contain extra overhead. WebSphere MQ is supported on more than 35 platforms and assures “once only” delivery of messages. For each of the major System i languages, API support is available to put and retrieve messages on queues. The WebSphere MQ for iSeries classes for JMS provide an interface to WebSphere MQ for iSeries, thereby offering an object-oriented interface.

### Sockets



Sockets enable the structuring of existing applications while keeping distributed objects in mind. They are easily accessible from both Java and back-office programs and offer excellent, even perhaps the best, performance. Sockets are also beneficial for interprogram communication. One program can act as the server program that listens on a sockets connection for input from the client program. The client program connects to the server through a socket. After the socket connection is established, the server and client programs can send or receive information. Sockets streams are used to communicate between programs that run in separate processes.



Most interoperability solutions rely on sockets as the underlying transport mechanism for communication between Java and existing applications. Programming to the sockets interface directly is a high-performance option at the expense of additional programming complexity. The sockets option offers the greatest potential for transaction throughput. For example, with a sockets solution, you must design the message protocol between client and server. Both the client and the server must parse messages, and data conversion must take place.

The IBM Client Technology Center (CTC) provides prewritten software components to simplify sockets programming greatly in order that you can create high-performance solutions while being shielded from the complexity of low-level sockets programming. Socket servers and APIs can jump-start the creation of defined interfaces into existing code. The prewritten software components (with or without the services) are available for trial runs and purchase.

### Files

Files are not typically viewed as a messaging sense, but they can be used as a way to interoperate between new Java code and existing code. Here is how it might work: The Java application places an Internet order request into an order-transaction file for later action by the back-end system. Files with trigger programs can be a good solution for asynchronous processing.

## Triggers

---

IBM DB2® for i5/OS database triggers are powerful user-written programs that are assigned to a database event (INSERT, UPDATE or DELETE). Although they are not an interface option in their own right, they are more-generally viewed as a modernization technique. They provide a perfect conduit for delivering data to applications because they capture all data updates, regardless of whether the updated data comes from Java, an existing application or an SQL function. Trigger programs are application-independent. The database manager activates triggers when the database performs a data-change event. The value implicit in this design is that triggers are initiated automatically, regardless of the type of program (Java, Visual Basic or an existing application) that generates the data change. Many new applications are written to access the database directly — by bypassing business logic in the application. In these cases, triggers enforce existing business logic (that is, they protect data integrity) without a program call. Additionally, triggers are used to validate input data, generate shadow records on different files and create audit trails.

In a client-server environment, triggers reduce I/O communications, thus improving response time (depending on the number and complexity of triggers). Because triggers are specific to DB2, using this advanced technique can mean that the trigger code does not easily port. You might have to build these database-protection mechanisms into the application or rewrite the code if the application must move across platforms.

## Interfaces for RPG and COBOL to call Web services

---

As of i5/OS V5R4, the XML Toolkit for iSeries has been enhanced with a Web-services client API so that any ILE program can call Web services. For more information, review the topic, “XML Toolkit for IBM System i5” in the IBM eServer™ iSeries Information Center (V5R4) (<http://publib.boulder.ibm.com/series/>).

## Interfaces for RPG and COBOL to access Java logic

This section further explores the interface options for RPG and COBOL programs to access Java logic.

### Using a messaging interface

Thus far, this paper has focused on accessing business logic from a Java or Windows program, but an existing application might need to be the controlling program that accesses Java routines. The messaging techniques that were just discussed work well for this scenario because there is a clear separation of function and less concern with the differences between the RPG and Java languages.

### Using a call interface

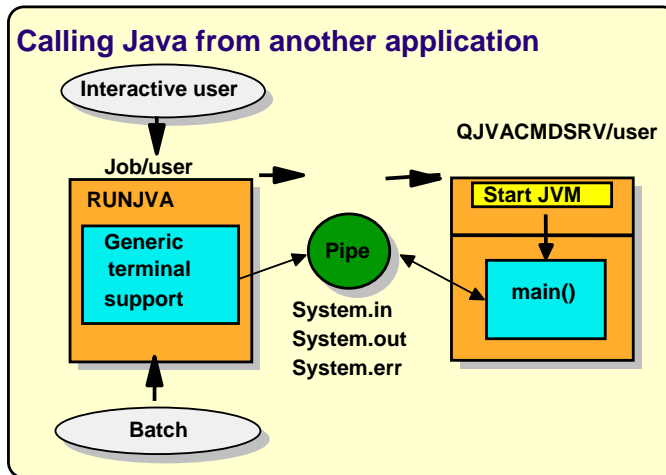


Figure 2. Calling Java from another application

To do direct calls to a Java program from an existing (traditional) System i program, one of following four techniques can be used: the RUNJVA or JAVA CL command, an existing language API, the JNI Invocation Interface or a stored procedure call.

### RUNJVA or JAVA CL command

Java is multithreaded and, thus, requires a multithread-capable job. However, not all System i job types are capable of multithreading. The Run Java (RUNJVA) and JAVA CL commands create a multithread-capable batch-immediate (BCI) job to start the JVM. You should limit the use of the RUNJVA or JAVA command to long-running Java programs because they create a second BCI job and start the JVM in that job. (See Figure 2.)

When the RUNJVA or JAVA command is used, the calling job stops and waits for the termination of the main() method in the Java class that was initiated. Starting the JVM requires loading several system classes along with the application classes. This is an expensive operation (from the perspective of system resources) and should not be repeated on a per-transaction basis. If the application requires fast, frequent entry to Java, you should instead consider a messaging option for communicating between the existing application and the Java program. To implement this, the RUNJVA or JAVA command must be used from an independent job. This independent,

or calling job, is held until the JVM ends. This supports communication between the existing programs and Java and is accomplished through the use of messaging options.

The `java.io` API that routes to `system.in` does not work if the starting job is a batch job (see Figure 2). Thus, for the initiation of batch jobs, write the output from the `system.out` and `system.err` APIs to a spool file for that job. For interactive jobs, the System i platform provides virtual-terminal support function that routes the Java input and output back to the interactive job user.

### Existing language APIs: QCMDEXC, `system()` function or RUNJVA

From RPG and COBOL programs, you can use the `QCMDEXC()` function to call Java. From C programs, the `system()` function can call a Java program. To call Java programs from other languages, you should first call a command to run the JVM. Various APIs can call a command. The one used depends on the language from which Java is called. For example, from the C language, the `system()` function runs the `RUNJVA` command. The `RUNJVA` command can be called directly from the command line. Other ILE languages can use the `QCMDEXC` call. Either call causes another process or job to be created in which the JVM runs. To communicate between processes, you must use a form of interprocess communication (for example, sockets, data queues or stream files). If data is passed between programs, it is necessary to ensure that the data is properly converted. If stream files are used, you can use the `system.in` and `system.out` APIs with RPG and COBOL (see previous section). The `JAVA` and `RUNJVA` commands create a new batch-immediate job to run the JVM. Use this operation only to initiate Java programs that have the purpose of listening on a queue or socket to handle message requests.

IBM suggests limited use of the `QCMDEXC` and `system()` functions and the `RUNJVA` command. They are resource-intensive and can reduce portability (because of their native nature). However, native language APIs are valid for invoking a JVM that is then available for long-running jobs.

### JNI Invocation interface

The JNI architecture defines the Java Invocation APIs. The systems that control the Java application models use the JNI internally to call Java programs. In OS/400 V5R1, ILE RPG and ILE COBOL were enhanced to support invocation of a JVM and to call Java methods. (ILE/C and C++ had long enjoyed access to the public Invocation APIs.) RPG or COBOL programs can start a JVM, or the program can be started the first time a Java method is called. For details, see the *ILE RPG Programmer's Guide* or the *ILE COBOL Programmer's Guide*. (**Important:** A strong caution goes with JNI as a method for calling to Java programs: Be careful to avoid creating a situation where a separate JVM starts for each user to that application.)

### Stored procedure calls

You can register a method in a Java class as a stored procedure, in which case, an RPG program can issue a stored procedure call to a Java stored procedure.



## Summary of all interface options

This section provides a quick reference to the interface options and considerations to help you decide which is a best fit for your situation.

Program Call options	Ideal usage	Time, skill set and cost to implement	Portability	Advantages	Considerations	Calling from existing programs to Java
<b>Program, Service Program and Command Call (IBM Toolbox for Java or iSeries Access APIs )</b>	<ul style="list-style-type: none"> <li>- Use with fixed maximum length of I/O parameters</li> <li>- If overhead does not cause performance concern, use PCML</li> <li>- Tightly coupled</li> </ul>	<ul style="list-style-type: none"> <li>- Quick and easy to implement</li> <li>- Low-cost to implement</li> </ul>	<ul style="list-style-type: none"> <li>- Toolbox code runs on any platform with JVM, target is System i only</li> <li>- Generates JCA-compliant code from WebSphere Development Studio Client</li> </ul>	<ul style="list-style-type: none"> <li>- WebSphere Development Studio Client wizards make it easy to do</li> <li>- Less overhead than PCML</li> </ul>	<ul style="list-style-type: none"> <li>- More complex to code with a variable unknown size of I/O tasks</li> <li>- If arbitrary length list, set maximum buffer length</li> <li>- Extra overhead</li> </ul>	N/A
<b>PCML</b>	<ul style="list-style-type: none"> <li>- Use with fixed maximum length of I/O parameters</li> <li>- Not good with unknown size of information or variables</li> <li>- If overhead does not cause performance concern</li> <li>- Tightly coupled</li> </ul>	<ul style="list-style-type: none"> <li>- Easy-to-use tag language based on XML</li> <li>- Requires less coding than program call</li> </ul>	<ul style="list-style-type: none"> <li>- Toolbox code runs on any platform with a JVM, target is System i only</li> </ul>	<ul style="list-style-type: none"> <li>- Describes existing program parameters, structures and field relationships</li> <li>- WebSphere Development Studio Client wizards create program call, handle data-type conversions</li> <li>- API interprets PCML, calls and gets existing program data</li> </ul>	<ul style="list-style-type: none"> <li>- Additional overhead</li> </ul>	N/A
<b>Stored Procedure</b>	<ul style="list-style-type: none"> <li>- Good for any clients supporting ODBC, JDBC or ADO.NET to call i5/OS programs</li> <li>- Good for returning result sets</li> <li>- Good for reusing existing programs</li> <li>- Tightly coupled</li> </ul>	<ul style="list-style-type: none"> <li>- Standardized way to call function</li> </ul>	<ul style="list-style-type: none"> <li>- Stored procedure invocation code is portable across JDBC, ODBC and ADO.NET clients</li> </ul>	<ul style="list-style-type: none"> <li>- Unofficial industry standard</li> <li>- Handles data conversions</li> <li>- Result-set capabilities</li> </ul>	<ul style="list-style-type: none"> <li>- Requires registration of program objects as stored procedures</li> <li>- Might need to change parameter passing design</li> </ul>	Yes



Program Call options	Ideal usage	Time, skill set and cost to implement	Portability	Advantages	Considerations	Calling from existing programs to Java
<b>RUNJVA or Java CL</b>	- To call long-running Java from existing programs			<ul style="list-style-type: none"> <li>Creates multi-thread-capable BCI job to start JVM</li> <li>Holds calling job until JVM ends, supporting data-queue communications</li> </ul>	<ul style="list-style-type: none"> <li>Expensive system-resource operation (if fast, frequent entry to Java is required, use messaging)</li> <li>Reduces application portability</li> </ul>	<ul style="list-style-type: none"> <li>Existing language APIs dynamically construct RUNJVA command</li> <li>• RPG and COBOL - QCMDEX EC()</li> <li>• C - system()</li> </ul>
<b>runtime.exec ()</b>	- To initiate Qshell function — best option for starting JVM from JVM		Portable	APIs available to support this call	<ul style="list-style-type: none"> <li>No support to get messages back from system command</li> <li>Uses I/O streams for inter-program communication</li> <li>Do NOT repeatedly start new process</li> </ul>	
<b>JNI</b> <ul style="list-style-type: none"> <li>• Java-to-ILE C/ C++</li> <li>• Java-to-ILE RPG and COBOL</li> </ul>	<ul style="list-style-type: none"> <li>- If existing C code has exported procedures from ILE, need to use service programs</li> <li>- Tightly coupled</li> </ul>	-	Implementation differences across platforms		<ul style="list-style-type: none"> <li>Threadsafe issues, impact to scalability</li> <li>Will not work from interactive job</li> <li>Must do conversion of literals passed</li> <li>Does not support future changes</li> </ul>	<ul style="list-style-type: none"> <li>Use to invoke or manage JVMs and call Java methods</li> <li>Be careful of activation groups ending unexpectedly</li> </ul>

Program Call options	Ideal usage	Time, skill set and cost to implement	Portability	Advantages	Considerations	Calling from existing programs to Java
<b>Invocation APIs</b>	<ul style="list-style-type: none"> <li>- If used one time to initiate a Java program listening on a queue or socket</li> </ul>	-			Potential performance and scalability impact because of the creation of multiple JVMs	

Messaging options	Ideal usage	Time, skill set and cost to implement	Portability	Advantages	Considerations	Calling from existing programs to Java
<b>Sockets</b>	<ul style="list-style-type: none"> <li>- Use with arbitrary-length list as I/O</li> <li>- Performance-critical situations</li> <li>- Loosely coupled</li> </ul>	<ul style="list-style-type: none"> <li>- Depends on sockets experience (creating from scratch or is framework prewritten)</li> <li>- Sockets servers and APIs simplify effort</li> </ul>	Highly portable	<ul style="list-style-type: none"> <li>- Best performance due to least amount of layers</li> <li>- Good separation between languages</li> <li>- Flexibility for future development</li> <li>- Easily accessed from Java and existing programs</li> </ul>	Message protocols and data conversions must be designed	Yes
<b>Data Queues (IBM Toolbox for Java)</b>	<ul style="list-style-type: none"> <li>- Use with arbitrary length list as I/O</li> <li>- Loosely coupled</li> </ul>	<ul style="list-style-type: none"> <li>- Quick and easy if using data queues today in client-server application</li> <li>- Low cost to implement</li> </ul>	Toolbox code will run on any platform with JVM, target is System i only	<ul style="list-style-type: none"> <li>- Better performance than Program Calls, files, message queues, data areas</li> <li>- Good separation between languages</li> <li>- Flexibility for future development</li> <li>- Simulates distributed calling model (does not use threads)</li> </ul>	<ul style="list-style-type: none"> <li>- Must establish message-passing conventions</li> <li>- Existing programs must talk to data queues</li> <li>- Must format data (conversion classes provided)</li> <li>- Re-create data queues daily for optimization</li> </ul>	Yes



Messaging options	Ideal usage	Time, skill set and cost to implement	Portability	Advantages	Considerations	Calling from existing programs to Java
<b>WebSphere MQ</b>	<ul style="list-style-type: none"> <li>- For "once only" delivery</li> <li>- Use with arbitrary length list as I/O</li> <li>- For cross-platform communications</li> <li>- When investment already made to use as company standard</li> <li>- Loosely coupled</li> </ul>	<ul style="list-style-type: none"> <li>- Additional cost for product</li> <li>- Extended learning curve</li> </ul>	Highly portable	<ul style="list-style-type: none"> <li>- Multiplatform standards-based, program communications through one messaging product</li> <li>- More robust and portable than data queues</li> <li>- Good separation between languages</li> <li>- Flexibility for future development</li> <li>- Supported on 35+ platforms</li> </ul>	<ul style="list-style-type: none"> <li>- Adds extra layer of code</li> <li>- Added product cost</li> </ul>	Yes
<b>Files</b>	<ul style="list-style-type: none"> <li>- Simple means of inter-operation</li> <li>- Guaranteed delivery</li> <li>- Loosely coupled</li> </ul>	<ul style="list-style-type: none"> <li>- Easy to do</li> </ul>		<ul style="list-style-type: none"> <li>- Familiar methodology for OPM programmers</li> <li>- Leverage triggers to call programs to handle message record</li> <li>- Good for asynchronous processing</li> </ul>		Yes

## Data-passing considerations

This paper has concentrated on interfaces, but another aspect of interoperability relates to the data that is passed on those interfaces. Some Java and native System i data types do not match. For instance, the System i JVM stores string data in two-byte Unicode (a universal data type that makes it possible for Java to exchange data in a more-global manner), but most System i character data is stored in one-byte EBCDIC format. Therefore, you must convert Java string data to EBCDIC when passing it between Java and RPG, COBOL or C programs on the System i platform. There are IBM Toolbox for Java data-conversion classes for this purpose.

If exchanging data between systems, you might also need to handle EBCDIC-to-ASCII conversions.

### XML

Because of the different data types within the languages and systems, you can use XML — a programming language and platform-independent method of data exchange.



In the messaging interfaces, XML can be the method for exchanging data. There is a trade off of extra overhead, in that both sides must parse the XML data or the documents that are exchanged. XML provides tagged data and tagged documents for business-to-business (B2B) applications.

The connectors that you use to create a wrapper for the call or messaging interface can receive an XML document as part of a transaction request. The connector uses an XML parser to transform the data contained in the document. The transformed XML data is passed through parameters on a call interface, or through queues or sockets in a message interface, to the back-end logic for processing.

An XML parser is a set of APIs that assist in the creation, navigation (retrieval) or modification of XML content. Ideally, an XML parser interface accommodates the application language of choice. The rest of this section of the paper includes a short summary of some of the products and product features you should become familiar with. **Note:** For details, read the Redbook *The Ins and Outs of XML and DB2 for iSeries* (SG24-7258) ([ibm.com/redbooks](http://ibm.com/redbooks)).

## XML Toolkit for iSeries

The XML Toolkit for iSeries (licensed program offering 5733XT1) and XML Toolkit for System i5™ (licensed program offering 5733XT2) provides XML parsers for both C++ and procedural languages.

**Note:** See the following Web site for more details: [ibm.com/systems/i/software/xml/index.html](http://ibm.com/systems/i/software/xml/index.html).

## DB2 XML Extenders for iSeries

On the System i platform, XML documents can also be transformed through the use of DB2 XML Extenders for iSeries (a no-charge download). These extenders provide data types to store XML documents in DB2 databases and also deliver functions to work with these structured documents. Managed by DB2 for i5/OS, these documents are stored in external files or, more probably, as character data (that is, they are stored as a collection of data items in multiple columns and tables). A visual administration tool is provided to help you easily define the mapping of elements and attributes from an XML document into columns and tables. After this is done, the DB2 XML Extender stored procedures can compose XML documents from data that is stored in DB2 for i5/OS tables, based on the mapping. You can send the resulting XML document over the Internet to a corresponding application server at another site.

XML documents can also be stored in DB2 for i5/OS in a single column with a new XML data type. As with other SQL data types, SQL is used to do fast and powerful searches. Specific XML elements or attributes can be extracted automatically into traditional SQL data types to leverage the sophisticated indexing and SQL-query capabilities within DB2 for i5/OS.

## XML tools in WebSphere Development Studio Client

WebSphere Development Studio Client for iSeries provides XML development tools that can be an ideal environment to create XML-based applications because it includes tools to perform the following actions:

- Build XML Document Type Definitions (DTDs), XML schemas, XML and Extensible Stylesheet Language Transformation (XSLT).
- Map between XML and different back-end files, such as an SQL query or relational tables.
- Debug Extensible Stylesheet Language (XSL) code.



Using XML development tools makes it possible to accomplish the following tasks:

- Generate JavaBeans from XML documents and do the reverse (that is, generate XML documents from JavaBeans).
- Use the Xalan processor to create HTML and XML documents with XSL. (XALAN is a specification for transforming XML documents into HTML or other XML document types.)
- Create and run an XPath with the XPath wizard. (XPath describes how to find and process items in XML documents by means of an addressing syntax that is based on a path through the document's logical structure or hierarchy.)
- Generate document access-definition scripts for use with DB2 for i5/OS.
- Create XML documents from DB2 for i5/OS data and do the reverse (that is, to create DB2 for i5/OS data from XML documents).
- Validate XML code for use with System i applications and processes.

## XML support in RPG

As of i5/OS V5R4, new opcodes (XML-SAX and XML-INTO) are available to support the parsing of XML documents within an RPG program. Search the eServer iSeries Information Center for details (<http://publib.boulder.ibm.com/series/>).

The IBM Client Technology Center (CTC) also has a tool to simplify the use of XML in RPG, making it possible for the generation and SAX (Simple API for XML) parsing at the record level: [ibm.com/servers/eserver/services/reusablecomponents.html](http://ibm.com/servers/eserver/services/reusablecomponents.html)

## XML support in COBOL

Use the XML Parse statement to process XML documents in your COBOL programs. Use the XML Generate statement to create XML documents. Search the eServer iSeries Information Center for details (<http://publib.boulder.ibm.com/series/>).

## Data-sharing options

---

The System i platform and its deeply integrated operating system (i5/OS) provide several ways to share data, including data areas and user spaces.

### Data areas

A data area is an i5/OS object that you can use to share data between programs, whether the data is character, local, decimal or logical data-area object. RPG and COBOL code can read and write to these objects.

Java programs can use the IBM Toolbox for Java DataArea class to access a data area. Based on the amount and type of data to be passed, data areas can be very helpful. A Decimal data area contains decimal data, which has a maximum 24-digit value with nine decimal positions. A character data area contains up to 2000 characters, but does not have a facility for tagging data with the proper Coded Character Set Identifier (CCSID). Therefore, the data area object assumes that the data is in the user's CCSID. When writing data, the data area object converts the data from a string (Unicode) to the user's



CCSID format before writing it to DB2 for i5/OS. When reading, the data area object assumes that the data is stored in the user's CCSID format and converts from that CCSID encoding to Unicode before returning the string to the program. When reading data from the data area, the amount of data read is based on the number of characters (not the number of bytes). A logical data area contains one character: a one (**1**) or a zero (**0**) to represent two opposing conditions (for example, **true** or **false**, **on** or **off**). The local data area is associated with a server job and cannot be accessed from another job. The i5/OS local data areas have the following restrictions:

- They are a fixed size (1024 characters).
- They cannot be created or deleted.
- When the server job ends, its associated local data area is automatically deleted, and the LocalDataArea object that refers to the job is no longer valid.

## Users spaces

You can also use user spaces to share data. *User spaces* are permanent objects that are located in the system or user domain and that contain streams of user-defined data.

RPG and COBOL programs can read and write information to a user space. Java programs can use the `UserSpace` class (provided in the IBM Toolbox for Java) to access a user space. It is useful with system APIs that put result information in a user space and when large amounts of data pass between programs. An example is the system list APIs that take advantage of user spaces to store the created lists.

**Note:** For details about the differences between user and system domains, see the "Application Programming Interfaces" section of the eServer iSeries Information Center (<http://publib.boulder.ibm.com/series/>). User spaces have an object type of \*USRSPC (with a 16-megabyte maximum size) that you can save and restore to other systems.

## Summary

---

You should treat the information presented here as an overview that serves as a starting position for those who are tasked with making their core-business functionality more accessible, both inside and outside of the enterprise. The large number of available programming interfaces provides enormous flexibility for connecting your existing RPG, COBOL, C and C++ application code with new user interfaces and applications that can be written in other languages. This makes it possible for you to draw on the unique advantages for a particular programming language when writing a new application or building a Web service, for example. You are no longer tied to a single coding technique or a particular platform. And, equally important, you can leverage code that has long been strategic to your organization, without having to completely rewrite it.

When there is a new requirement to present business functionality over the Web or to provide a way to invoke a transaction programmatically, you can review your options to determine a best fit for your requirements. As this document has emphasized, your choices for presenting the new business function are diverse, flexible and easier to achieve than you might imagine.

Visit some of the Web sites listed in the Resources section of this paper. What will become apparent is that, if you work on the System i platform, there are an enormous number of resources and the opportunities for gaining in-depth understandings of all the technologies mentioned here.



## Resources

---

These Web sites provide useful references to supplement the information contained in this document:

- IBM eServer iSeries [System i] Information Center  
<http://publib.boulder.ibm.com/series/>
  - System i thread support (search on “threads”)
  - JNI architectures and the Invocation API (search on “JNI”)
  - Host servers (search on “host servers”)
  - Native JDBC driver (search on “jdbc”)
  - Data Queues (search on “Data Queue APIs”)
  - Thread safety (search on “thread safety”)
- IBM Redbooks  
**ibm.com/redbooks**
  - *Modernizing and Improving the Maintainability of RPG Applications* (REDP-4046)
  - *Enabling Web Services for the IBM eServer iSeries Server* (REDP-0192)
  - *Stored Procedures, Triggers and User-defined Functions on DB2 for iSeries* (samples of stored procedures and sockets) (SG24-6503)
  - *Who Knew You Could Do That with RPG IV? A Sorcerer's Guide to System Access and More* (samples of stored procedures) (SG24-5402)
  - *WebSphere Development Studio Client for iSeries V5.1.2* (SG24-6961)
  - Search on “SOA” for more than 40 entries
- Tools Innovation (for System i development tools)  
[www-304.ibm.com/jct09002c/isv/spc/tools.html](http://www-304.ibm.com/jct09002c/isv/spc/tools.html)
- XML  
**ibm.com/eserver/series/software/xml/**
- The IBM Toolbox for Java  
**ibm.com/eserver/series/toolbox/**
- *Java for RPG Programmers*, Phil Coulthard and George Farr (ISBN: 1-931182-06)
- IBM Client Technology Center Lab Services (CTC) (IBM prewritten software components)  
**ibm.com/eserver/services/reusablecomponents.html**
- JTOpen  
[oss.software.ibm.com/developerworks/opensource/jt400/](http://oss.software.ibm.com/developerworks/opensource/jt400/)
- SOA and Web services  
**ibm.com/developerworks/webservices/**
- *Using IBM Developer Kit for Java* (sockets and the `java.lang.Runtime.exec()` command)  
<http://publib.boulder.ibm.com/infocenter/series/v5r4/topic/rzaha/rzaha.pdf>
- *Sockets Programming* manual (System i socket examples)  
<http://publib.boulder.ibm.com/infocenter/series/v5r4/topic/rzab6/rzab6.pdf>

## About the author

---

Cheryl Renner is a senior engineer in the ISV Business Strategy and Enablement organization within IBM. She has worked in the midrange area for more than 20 years and has held a wide variety of positions at IBM, from IBM System/36™ RPG developer to IBM Query/400 team lead to assessor on the System i Application Innovation program. Her focus is helping System i solution providers enhance and extend their applications.



## Trademarks and special notices

---

© Copyright. IBM Corporation 1994-2006. All rights reserved.

References in this document to IBM products or services do not imply that IBM intends to make them available in every country.

AIX, DB2, developerWorks, Domino, eServer, i5/OS, IBM, the IBM logo, Integrated Language Environment, iSeries, Lotus, Network Station, OS/2, OS/400, Redbooks, System 36, System i, System i5 and WebSphere are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both.

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

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

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

Information is provided "AS IS" without warranty of any kind.

All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics may vary by customer.

Information concerning non-IBM products was obtained from a supplier of these products, published announcement material, or other publicly available sources and does not constitute an endorsement of such products by IBM. Sources for non-IBM list prices and performance numbers are taken from publicly available information, including vendor announcements and vendor worldwide homepages. IBM has not tested these products and cannot confirm the accuracy of performance, capability, or any other claims related to non-IBM products. Questions on the capability of non-IBM products should be addressed to the supplier of those products.