PartnerWorld

Developer Connection

March 2000

Technical Magazine

PartnerWorld for Developers

e-business

IBM

It's full speed ahead...

by Jean Swanson

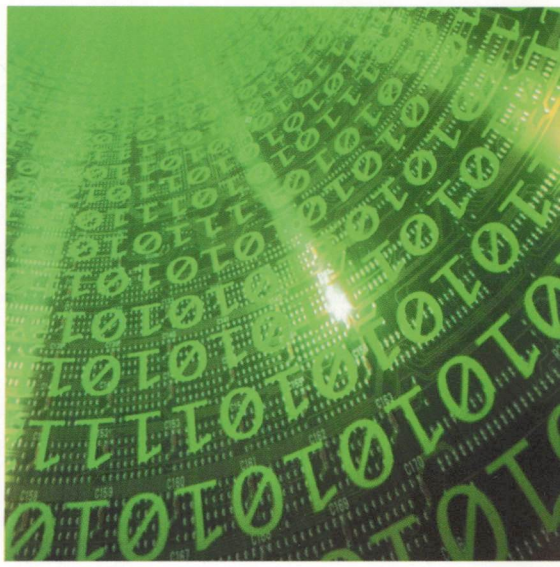
Now that the Y2K challenge is behind us, we can concentrate on how to grow the IBM* Developer Connection for you, our customers. Here are just a few of the ideas under consideration on how we can make our offering even better for the new millennium.

You spoke and we listened. We're adding more products like Linux, Lotus■ and Windows 2000▲, compilers and development tools for XML, WebSphere*, DB2* and Java● that you have told us you want and need. We're also looking into delivering the Developer Connection on media other than CDs.

We're also pursuing information to publish in the magazine that will help keep you ahead of the competition. In this issue of the magazine, you'll find more articles on hot new tools like "Command Framework for e-business" on page 2. The article introduces command beans that provide a structured way for the client interface to invoke server-side business logic or access data.

Find out how to build database servlets with WebSphere Studio in the article on page 6. The author describes how to build a servlet that updates an employee record in the DB2 sample database.

You'll also find articles on designing an architecture for a Data Warehouse Populating System, installing the Team Repository Server for VisualAge* for Java on Windows NT▲, using DTDs in XML, pervasive computing, Sash Weblication for Windows, and Java 2 Security.



During the past four years, IBM's technology leadership and view of e-business have helped transform the Internet. We're now on the brink of another shift in the technology world. Customers increasingly are demanding open standards for interoperability across disparate platforms.

To meet this demand, a developer needs access to the technical information and tools that can help accelerate development efforts, deliver solutions faster and stay ahead in the technology race. The Developer Connection team is working to ensure your development needs are met in the year 2000 and beyond.

In this issue

www.developer.ibm.com/devcon/

Command Framework for e-business

By George Copeland, Matthew McClain and Jim Hsu

Typically, e-business applications are driven by a set of client interactions to server-side business logic, as depicted in ***Figure 1***. As shown at the center of the diagram, Command Beans provide a structured way for the client interface to invoke server-side business logic or access data. This article introduces an extensible "Command Framework" and demonstrates how it can be an important component of e-business. It also provides some sample code to get you started in writing and using your very own Command Beans.

Why Command Beans?

There are two major problems that the Command Framework attempts to address. One problem is performance. The granularity of artifacts on the server (such as, objects, tables, procedure calls, files, etc.) often causes a single client-initiated business logic request to involve several round-trip messages between the client and server. This might entail extra calls to perform the business task and then impose additional calls to retrieve the results of that task.

If the client and target server are not in the same Java Virtual Machine (JVM), these calls go between processes and are, therefore, expensive in terms of computer resources. If the calls must go over a network, they are even more costly. To prevent unnecessary delays and improve application performance, it is advantageous to perform a business task in as few interactions between the client and server sides as is natural to the task. Command Beans provide the necessary building blocks to achieve this.

Another problem is that there are several possible styles for how business logic can be implemented, including Enterprise JavaBeans (EJB), JDBC direct database access, JDBC access to stored procedures, the Common Connector Framework, file system access, etc. In addition to different implementation programming models, each of these styles has a different way to invoke a request to execute business logic. Because the Command Framework is generic and extensible, it can hide all these different types of server invocation mechanisms under a simple and uniform mechanism.

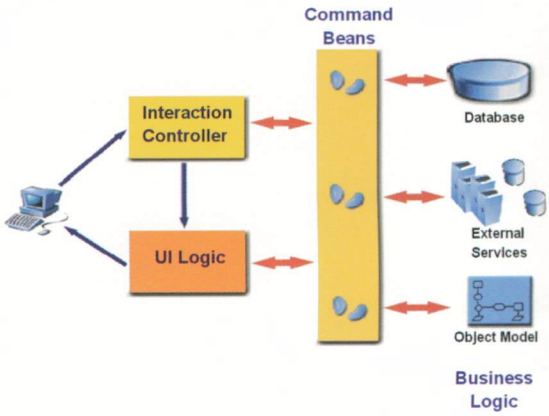


Figure 1. Web Interaction model

Server-side execution of Command Beans

A Command Bean (hereafter called a "command") is a JavaBean that encapsulates a single request to a "target server," which is where all commands are actually executed. The target server can run in the same JVM as the client or run in a separate JVM. The client instantiates the command, sets its input data and tells it to execute. The command infrastructure determines the target server and passes a copy of the command to that target server. The target server executes the command and returns a copy of the executed command back to the client. The client then can get any output data from the command.

This process allows the command to be executed within the environment of a target server, so that multiple accesses by the command to server resources avoids distributed overhead. Any server can be a target if it supports Java access to its resources and provides a protocol that allows a serialized command to be sent between a client JVM and the server JVM. ***Figure 2*** illustrates the interaction among objects in the Command Framework.

Command Beans provide the necessary building blocks to improve application performance and prevent unneccessary delays.

Commands in a transactional environment

One exciting use of commands is in a transactional environment. This example describes an IBM* WebSphere* funds transfer transaction that is initiated from a Lotus Domino agent.

If the agent calls each entity EJB separately, many remote calls are made. First, the agent must create a transaction coordinator. Then, it makes the following remote calls:

1. Lookup account home
2. Narrow home object
3. Find source account
4. Call withdrawal method
5. Find destination account
6. Call deposit method
7. End transaction

If the agent calls a helper session EJB, the number of remote calls decreases somewhat:

1. Lookup fundsTransfer home
2. Narrow home object
3. Create session bean
4. Call transfer method with source account, destination account and amount
5. Destroy session bean

With commands, the agent can simply invoke an EJB command target. The only remote call in this case is: *Call execute method on command*

Instantiating a Command

There are several ways to instantiate a command. The fastest and most efficient way is with a constructor:

```
MyCommand myCommand = new MyCommand();
```

The standard JavaBean mechanism uses Beans.instantiate where the beanName is either a class or instance name:

```
MyCommand myCommand =
   (MyCommand) Beans.instantiate(null, beanName);
```

The most flexible way is a factory whose parameters can be used to determine the class or bean name:

```
MyCommand myCommand = (MyCommand) factory.create(...);
```

Because each of the above techniques have different advantages that fit different scenarios, the Command Framework does not specify how instantiation should be done. A given concrete command usually specifies the preferred method of its creation.

Command lifecycle

When a command has been instantiated, it is in the "new" state. (See ***Figure 3*** for an illustration of state transition in commands.) The command interface provides a minimum set of methods common to all commands that a client uses to control the life cycle of a command. For each specific command class, some input properties are required and others are optional. When all the required input properties have been set, the command is in the "initialized" state.

The command interface introduces the following three methods:

- public boolean isReadyToCallExecute();
- public void execute() throwsCommandException;
- public void reset();

The isReadyToCallExecute method returns true if all required inputs are set.

The execute method should not be called until all the required input properties have been set (that is, it is in the "initialized" state). After the execute method is called, the command goes from the "initialized" state to the "executed" state.

The reset method retains previously set input properties but resets output properties on the object back to their original values prior to any execute calls (such as, null or 0). Resetting the output properties prevents exposing outdated output values of previous command executions to later executions.

Reusing the same command instance is convenient if there are several complex input properties that do not need to be changed for the next invocation. This reuse also can avoid the overhead of instantiation and garbage collection.

Compensable Commands

The CompensableCommand interface is an extension of the Command interface, which allows a command to be associated with another command that compensates for its actions. This action can be thought of as "undoing" the command for which it provides compensation. Some actions, such as sending e-mail, cannot be literally "undone." In those cases, a CompensableCommand does the best possible job of reversing the action, such as sending out a cancellation notification through e-mail. CompensableCommands also are useful if holding locks across two events causes unacceptable data contention.

CompensableCommand introduces only one method:

```
public Command getCompensatingCommand();
```

A client wanting to reverse a Command called reservationCommand would do the following:

```
reservationCommand.getCompensatingCommand().
   execute();
```

The client calls the getCompensatingCommand method on an executed command as illustrated in ***Figure 3***. It does not change the state of the target command. The returned compensating command is in the initialized state.

Alternatively, the client could initialize a compensating command directly if its input properties (for example, the reservation number) were known.

For example, the implementation of makeReservation.getCompensatingCommand might be as follows:

```
Command getCompensatingCommand() {
    CancelReservation c = new CancelReservation();
    c.setReservationNumber(
      this.getReservationNumber());
    return c;
}
```

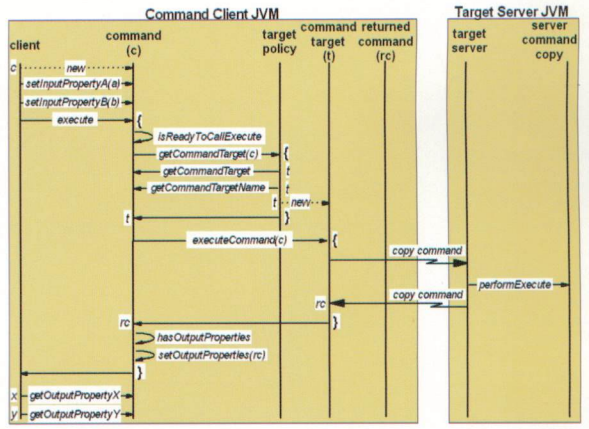


Figure 2. Targetable Command Interaction diagram (distributed case)

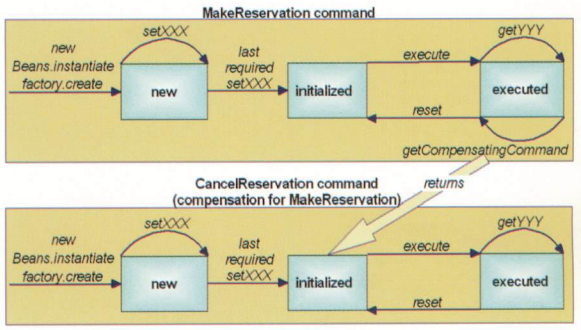


Figure 3. Command Suite diagram

Recommended client interface

To fit naturally into standard JavaBean programming, extenders of the Command Framework should follow the standard JavaBean naming guidelines. Input and output data for a command should be done using JavaBean properties. Each input property (such as, XXX) has a method whose signature is public void setXXX(XXX xxx);

A command's output data is presented as JavaBean output properties. Each output property (such as, YYY) has a method whose signature is public YYY getYYY();

The output methods should not be called until the execute method has been called and the command is in the "executed" state.

The property "get" and "set" methods are not part of the Command interface because they are different for each class, depending on what the properties are for the command. However, following the standard JavaBeans naming guidelines ensures interoperability with the widest range of tooling.

TargetableCommands and CommandTargets

A CommandTarget interface is the wrapper interface for a target server JVM where a

CONTINUED ON PAGE 4

CONTINUED FROM PAGE 3

command can be executed. The CommandTarget extends java.rmi.Remote, so that it can potentially be an EJBObject (either SessionBean or EntityBean). A CommandTarget implementation is responsible for the proper execution of a command in a particular target server environment.

This typically involves the following:

1. Copying the command over a server-specific protocol to the target server's JVM (such as RMI).
2. Executing the command in the target server's JVM.
3. Copying the executed command over a server-specific protocol back to the client's JVM.

Each target server environment will typically need just one class that implements the CommandTarget interface. The TargetableCommand interface is an extension of the Command interface that allows a command to be directed to a particular CommandTarget for execution. This interface has two ways to specify which CommandTarget a command is associated with a client.

If the client has references to server objects, it can set the CommandTarget directly on the command by calling:

```
public void setCommandTarget (
   CommandTarget commandTarget);
```

Otherwise, if the client is not directly aware of server objects, it can set the CommandTarget's bean name on the command by calling:

```
public void setCommandTargetName (
   String targetName);
```

There are corresponding "get" methods for these so that the command runtime can access them.

Additional methods in the TargetableCommand interface are used to manage the TargetableCommand.

The TargetableCommand's implementation of the execute method is called in the client and handles the distributed issues. Because of this, a separate method is provided for the command writer to tell the command runtime the business logic of the command that should be called in the target server:

```
public void performExecute() throws Exception;
```

When the client and target server are in different JVMs, the returned command is a different instance than the one to which the client has a reference. It is a copy that has been executed, so its output properties are filled in. The following method allows the command writer to tell the command runtime how to copy the output properties from the returned command instance to the client's command instance:

```
public void setOutputProperties(
   TargetableCommandfromCommand);
```

There is a default implementation for this method that uses introspection. Due to the Java language security model, the implementation is often limited to copying only public fields. Therefore, for optimum results, write your own implementation for each command.

If a command has no output properties, then copying the command back to the client is not needed and copying the output properties into the client's command instance is not needed. The following method allows the command writer to tell the command runtime whether there are any output properties to allow these optimizations:

```
public boolean hasOutputProperties();
```

The default implementation for this method returns true.

TargetPolicy

Because there are multiple ways to determine which CommandTarget to use for a TargetableCommand, a policy has to be established about how these interact. Rather than burn this policy into the TargetableCommandImpl implementation, a separate TargetPolicy interface is used to allow different policies to be plugged into the framework. This has a single method:

```
public CommandTargetgetCommandTarget (
   TargetableCommand command);
```

The TargetableCommandImpl class adds the following static methods to allow a TargetPolicy class to be set and queried:

```
public static void setTargetPolicy (
   TargetPolicytargetPolicy);
public static TargetPolicy getTargetPolicy();
```

A default implementation of the TargetPolicy interface, named TargetPolicyDefault, is provided. TargetPolicyDefault adds the following methods to register and unregister the mapping between TargetableCommand and CommandTarget bean names:

The policy implemented by the TargetPolicyDefault class is the following:

```
public void registerCommand(String commandBeanName,
                            String targetBeanName);
public void unregisterCommand (
   String commandBeanName);
```

1. If the TargetableCommand parameter of the TargetPolicy.getCommandTarget method contains a CommandTarget (obtainable through the TargetableCommand.getCommandTarget method), use it.
2. Otherwise, if the TargetableCommand contains a CommandTarget bean name (obtainable via the TargetableCommand.getCommandTargetName method) use it.
3. Otherwise, if the TargetPolicyDefault contains a registered mapping between Command bean name and CommandTarget bean name (setable via the TargetPolicyDefault.registerCommand method) use it.
4. Otherwise, if a default target bean name has been set (setable via the TargetPolicyDefault.setDefaultTargetName method) use it.
5. Otherwise, return null.

TargetableCommandImpl

An implementation of the TargetableCommand interface is provided by the TargetableCommandImpl class. The TargetableCommandImpl provides a default implementation for the following methods:

- ***hasOutputProperties:*** This implementation returns the hasOutputProperties instance variable. False can be returned as an optimization that can eliminate unnecessary copying and message overhead. In this case, the hasOutputProperties instance variable should be set to false rather than overriding this method, because the instance variable also is used in the execute method. This method is made final to enforce this.
- ***setOutputProperties:*** This uses introspection to copy all instance variables, provided that all instance variables are non-private and non-package (that is, they must be public or protected). This implementation does not copy final, static or transient fields. If this default implementation is not acceptable, the command writer can override this method. A typical implementation just does the following:

```
this.outputPropertyA = fromCommand.outputPropertyA;
this.outputPropertyB = fromCommand.outputPropertyB;
```

- ***execute:*** This does the following:
  1. Throws an UnsetInputPropertiesException if this command's isReadyToCallExecute method returns false.
  2. Gets the CommandTarget for this command from the TargetPolicy.
  3. Calls the CommandTarget. executeCommand method to execute this command, which calls the TargetableCommand. performExecute method.
  4. If the hasOutputProperties method returns true and the returned command is not the same instance as this command, it calls the setOutputProperties method so that the results will be copied into this command.
- ***setCommandTarget:*** This sets the commandTarget instance variable.
- ***getCommandTarget:*** This returns the commandTarget instance variable.
- ***setCommandTargetName:*** This sets the commandTargetName instance variable.
- ***getCommandTargetName:*** This returns the commandTargetName instance variable.

The TargetableCommandImpl class expects the command writer to implement the following methods:

- performExecute
- reset
- isReadyToCallExecute

As an example, a simple command might look like:

```
import com.ibm.util.command.*;

public class HelloWorldCommand
    extendsTargetableCommandImpl
{

    public HelloWorldCommand() {
        hasOutputProperties = false;
    }

    public void performExecute()throws Exception {
        System.out.println("Hello World");
    }

    public void reset() {
    }

    public boolean isReadyToCallExecute() {
        return true;
    }
}
```

Sample code

The HelloWorldCommand class shown above is available along with a reference implementation of the Command Framework at the Developer Connection Web site at www.developer.ibm.com/devcon/.

The sample includes reference implementations of local, RMI and EJB command targets. Also included is a "QuickStart" client / server example. QuickStartServer simply registers an RMI command receiver. Run the server first with "java QuickStartServer."

```
import java.rmi.*;
import java.rmi.registry.*;
import com.ibm.util.command.rmi.*;

public class QuickStartServer
{

    public static void main(String args[])
        throws Exception {

        // Register CommandTarget
        Registry registry =
            LocateRegistry.createRegistry(
                Registry.REGISTRY_PORT);
        Naming.rebind("RMICommandReceiver",
                      new RMICommandReceiverImpl());

        System.out.println("RMICommandReceiver server
        ready and waiting");
    }
}
```

After the server has started, run the client with "java QuickStartClient." The QuickStartClient will run HelloWorldCommand first locally and then remotely in the QuickStartServer. The local command target is useful for early testing. As seen in the sample code, it is straightforward to alter the client to use different command targets as the e-business application evolves.

```
import java.io.*;
import java.rmi.*;
import java.util.*;
import com.ibm.util.command.*;

public class QuickStartClient
{

    public static void main(String args[])
        throws Exception {

        HelloWorldCommand helloWorldCommand =
            new HelloWorldCommand();

        // Run command locally
        helloWorldCommand.setCommandTargetName(
            "com.ibm.util.command.local.LocalCommandTarget");
        helloWorldCommand.execute();

        // Run command remotely via RMI
        helloWorldCommand.setCommandTargetName(
            "com.ibm.util.command.rmi.RMICommandTarget");
        helloWorldCommand.execute();
    }
}
```

Conclusion

This article demonstrates how commands can be used to partition an application into efficient units of client-server interaction.

In the Command Framework, client code is independent of the style of the command's implementation and independent of where the command is physically executed. This makes it ideal for heterogeneous environments – for example, when the WebSphere Application Server interfaces with other applications, such as Domino or DB2. In fact, the Command Framework is under development for use with the WebSphere Application Server, although its potential also can be realized in other environments.

George Copeland *is currently a Senior Technical Staff Member at IBM Austin. He received his B.S. degree from Christian Brothers College and his M.E. and Ph.D. degrees from the University of Florida. He was chief architect of the DB2 UDB, a primary architect of Enterprise JavaBeans and helped the Net.Commerce product move to Java. He currently works in the area of caching dynamic content in Web applications. You can contact him at copeland@austin.ibm.com.*

Matthew McClain *joined the IBM Software Group Technology Center in June 1997, after completing a B.S. in Computer Engineering at the University of Illinois, Urbana-Champaign. Matthew has written a prototype EJB container and developed database logic for at least six different Web applications. Matthew's most recent work includes creating a cache for Commands and JSPs. He can be reached at mmcclain@austin.ibm.com.*

Jim Hsu *is a Software Engineer, who joined IBM in May 1997. He is a member of the Software Group Technology Center in Austin, Texas. Jim received his bachelor's and master's degrees in electrical engineering and computer science from MIT. He is interested in technologies with immediate widespread impact and has worked on the development of Java programs that involve the Internet. You can contact him at jimhsu@us.ibm.com.*

The authors would like to thank Michael Conner for the initial command concept. Steven Halter, Erik Vodal and Mark Hubbard also made significant contributions to the Command Framework design. Figure 1 was borrowed from the "Understanding Technology Choices" which can be viewed online at www.ibm.com/software/ebusiness/buildapps/understand.html. This document provides more details about the IBM Application Framework for e-business.

Building Database Servlets with Websphere Studio

by Morgan Kinne

One of the most common activities performed by a dynamic Web site is the creation, maintenance, query and display of data in SQL databases. IBM WebSphere Studio makes it easy to build these dynamic elements of your Web site through the use of its SQL Wizard and Database Wizard. These wizards generate SQL statements, HTML pages, Java Server Pages (JSP), Java servlets and Java Beans that can insert, update and delete data in your existing Java-enabled databases such as IBM's DB2 Universal Database. This article describes how to build a servlet that updates an employee record in the DB2 sample database.

What is WebSphere Studio?

IBM WebSphere Studio 3.0 is a development tool for creating, managing and debugging Web applications that can be executed on IBM WebSphere Application Server. WebSphere Studio 3.0 supports both WebSphere Application Server 2.x and 3.0 and includes a development copy of WebSphere Application Server 3.0 Standard Edition for testing your Web applications. Studio provides a workbench that allows development teams to organize, manage and publish their projects. The workbench automatically corrects links as pages are moved to different folders within a project. Additional components that are integrated with the workbench are:

- A WYSIWYG page designer that supports both JSP and HTML.
- A WYSIWYG applet designer for creating Java applets to be included in your Web pages.
- A Web art designer for creating banners and other graphics to be included in your pages.
- A remote debugger that eases debugging of your Java servlets, beans and JSP pages.
- Wizards for building dynamic interactive pages.

Studio also provides tight integration with, and a complimentary copy of, IBM VisualAge for Java Professional Edition 3.0. A complimentary copy of WebSphere Studio 3.0 Entry Edition can be downloaded from the Studio home page at http://www-4.ibm.com/software/webservers/studio/index.html. Note: Both products also are available from the Developer Connection. VisualAge for Java Professional Edition can be downloaded from the online catalog at the Advanced Level. WebSphere Studio 3.0 Entry Edition is available at the Guest Level.

Configuring your system

The Studio SQL Wizard and the generated Java Beans both use the Java Database Connectivity 1.0 API (JDBC) to access relational databases. Therefore, the JDBC database drivers for your vendor's database must be available in the system's classpath environment variable for Studio and in the classpath variable in d:\WebSphere\AppServer\bin\admin.config for the Application Server. The vendor's JDBC database drivers are normally installed along with the database client. In the case of DB2 these drivers are normally found in d:\sqllib\java\db2java.zip. Failure to add these drivers to the appropriate classpaths is the most common error when beginning to use Studio.

Of course the database server must also be installed somewhere on your network. ***Figure 1*** shows a typical network configuration of the necessary elements when building database servlets. These products also could be installed on a single machine, which is an excellent configuration for the development and test cycle.

WebSphere Studio 3.0 is a development tool for creating, managing and debugging Web applications.

Creating a Studio project

New projects can be created in Studio through its File menu. You must decide on which version of WebSphere Application Server you will deploy your new project. The Advanced page of the Project Properties dialog is where you make these selections. ***Figure 2*** shows the dialog as it is configured for this project.

There are various factors that enter into this decision. Servlets that are built for Application Server 2.x also execute successfully on Application Server 3.0. However, the generated Java Beans use the Application Server 2.x Connection Manager APIs that have been deprecated (although they are still functional) on Application Server V3.0 instead of the new Connection Pooling APIs. You also will be unable to exploit JSP 1.0 when generating for Application Server Version 2.x. However, if you have servers running various levels of Application Server then selecting Application Server Version 2.x is the most flexible choice because you can deploy your projects on any of your servers.

Regardless of your server selection, select JSP .91 for this project. Projects that use JSP 1.0 must be published differently than what is described in this article. If you are interested in exploring how JSP 1.0 projects are published, review the paper at http://www-4.ibm.com/software/webservers/studio/doc/articles/publish301.html.

Using the SQL Wizard

The SQL Wizard is launched from the Tools menu in Studio. The wizard requires that you log on to a database. The database URL field identifies the database to be used. We will update a record in the STAFF table in the DB2 SAMPLE database for this example and the URL is jdbc:db2:sample. DB2 offers a choice of connectivity options. Choosing the DB2 local (or .app) driver yields the best performance in a server-side environment. Of course a valid userid and password must be provided. Click the Connect button when you have completed the fields on the page.

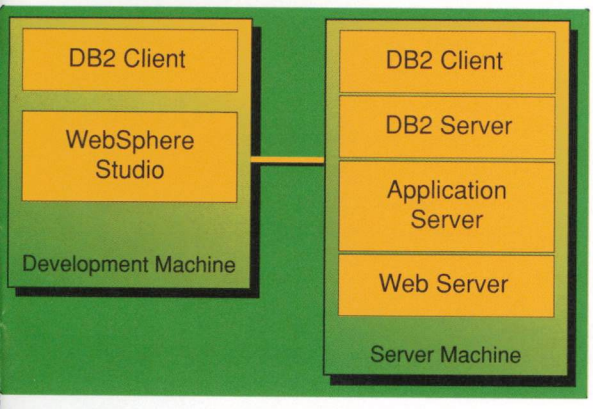


Figure 1. A typical network configuration for database servlets

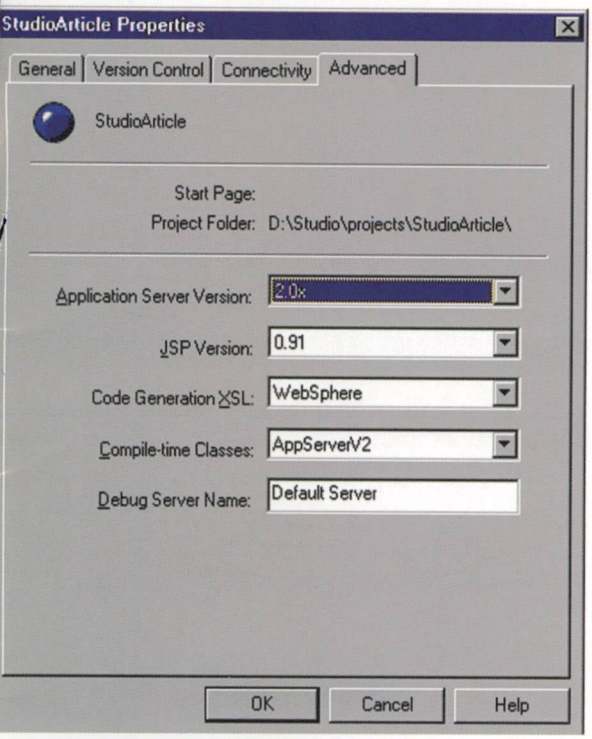


Figure 2. The project's properties dialog

When you have successfully connected to the database, the wizard shows the tables you are allowed to access. Choose the tables to use in your SQL statement here as well as the type of SQL statement you will build. There are two steps in the process of updating data. First, you must query the database to obtain the existing data through a Select statement and then later update the data using an SQL Update statement. We'll just build a Select statement here and later the Database Wizard builds the Update statement for you automatically. The trick is to make sure you access only one table and that your SQL Select statement returns only one record. Select the STAFF table on the Tables page of the SQL Wizard. ***Figure 3*** shows the completed page.

The Join Page is of no interest in this example because we are working with only one table. Also, by not selecting any specific columns on the Columns page, the query returns all the columns in the table.

The Condition page allows you to specify a parameter for the query that will select an individual row in the database. You normally will provide a condition on a key field to select a particular record but there is no requirement that you use a key field. Select the ID column and the "is exactly equal to" operator and then click the Parameter button. Type id in the popup dialog and then click OK. ***Figure 4*** shows the completed Condition page.

You can see the SQL statement that was built on the wizard's SQL page. Click the Finish button to save the .sql file containing the choices you made when building the query.

Using the Database Wizard

The next step in building our project is to use the Database Wizard to construct the servlets, beans and pages that will actually do the update operation based on the information in the .sql file. Studio uses two separate wizards in the anticipation that different people with different skill sets perform these tasks.

Launch the Database Wizard from the Studio Tools menu. The drop-down list allows you to select the .sql file to use. The SQL statement for the selected file is visible in the preview area to aid in selecting the proper file. This project contains only one .sql file so no action is required on this page. Click the Next button.

The Web Pages Window allows you to choose which pages the wizard should generate. The wizard always generates a servlet, bean and associated pages to execute the SQL statement, in this case a Select statement. The Allow Update check box is enabled because our SQL statement references a single table. Checking this option causes the wizard to generate a second servlet, bean and associated pages that updates the database after the user makes changes in the data from the original query.

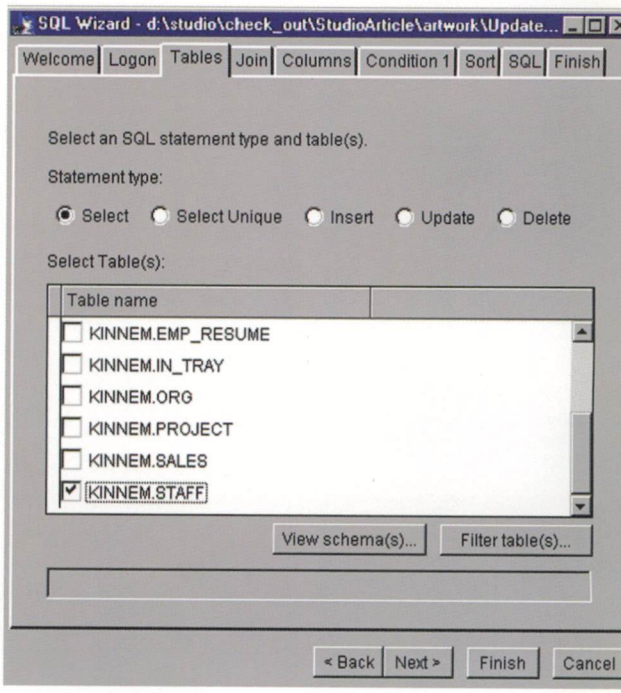


Figure 3. The completed Tables page

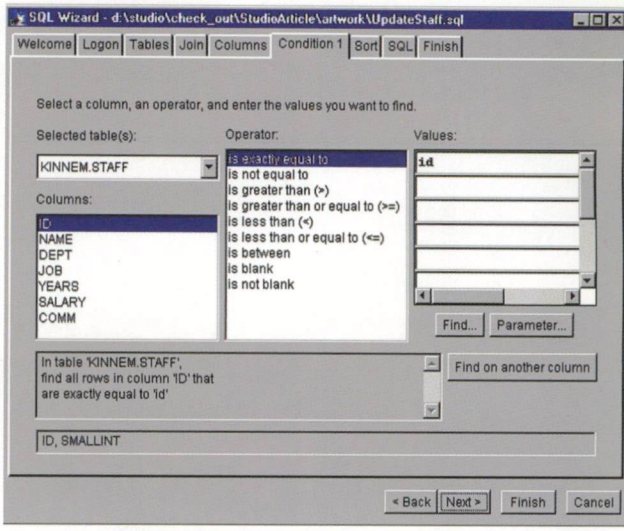


Figure 4. The completed Condition page

The generated code does not hold any locks on the database. Holding locks is probably not wise in Internet applications but if you wish to do so, explore using VisualAge for Java to build Enterprise Java Beans (EJB) for your database update operations. Instead of holding locks, the wizard generates hidden fields in the JSP page that shows the results of the query. These hidden fields hold the original data that was returned by the query and is used in the update operation to find the original database record. The Run as Transaction check box causes the wizard to generate an explicit JDBC commit in the bean as opposed to using JDBC

CONTINUED ON PAGE 8

CONTINUED FROM PAGE 7

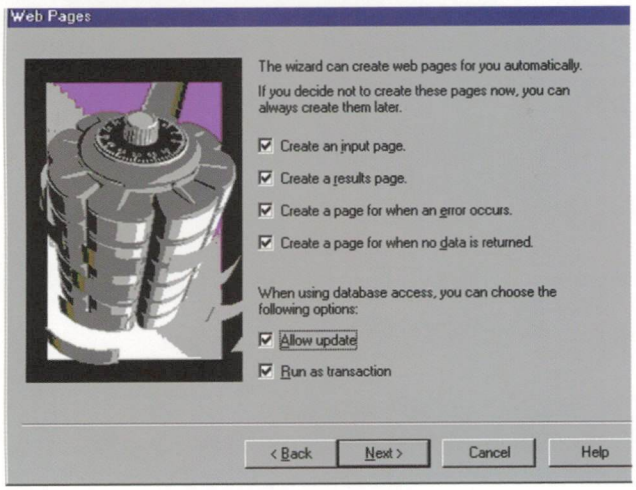


Figure 5. The completed Web Pages page

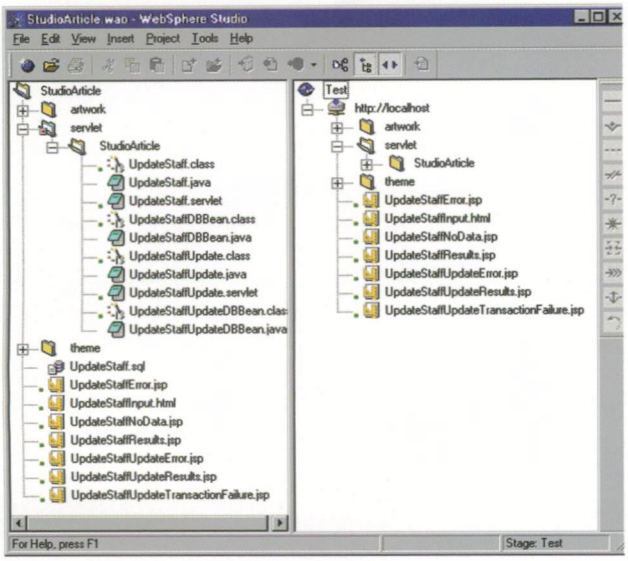


Figure 6. The Studio workbench

autocommit. If the explicit commit fails, the servlet presents the user with a generated transaction failure page that you can tailor using the Studio Page Designer. Not all databases support explicit commits but you should use them wherever possible. Of course DB2 does support this, so you should select all the check boxes on this page and click the Next button. ***Figure 5*** shows the completed page.

The Input page shows the pre-checked parameter for the Select statement. If you choose, you also can have the user specify a userid and password that is used to connect to the database instead of obtaining it from the .servlet file it will generate. Leave the id parameter checked and click Next.

The Results Page allows you to choose what data from the query is actually shown to the user. All the columns returned from the query are preselected for you.

Keep clicking the Next button until you reach the Successful Update page. This page allows you to choose what information the user sees after a successful update. Click the Uncheck All button and then check the numAffectedRows property. This property has a value of 1 when the update is successful and a value of either 0 or -1 for an unsuccessful update.

Keep clicking the Next button until you get to the Finish Page. The Finish page shows you the files that are generated by the wizard. You may want to click the Rename button to give your files a meaningful name. Go ahead and click the button and change the prefix from Query1 to UpdateStaff. Use uppercase characters as the first character in a prefix to conform with Java naming conventions. Click the Finish button when you're done. ***Figure 6*** shows the generated files that the wizard added to the Studio workbench.

Publishing and testing the project

Before publishing your project, you must define your publishing targets. Select the server in the right pane and open its properties dialog from its pop-up menu. If you are publishing to a server on the same machine, you can use a file system publish option; otherwise use the File Transfer Protocol (FTP) publish option. You specify the actual targets by clicking the Define Publishing Targets button. Regardless of the publish option you use, select your Web server's document root directory for your HTML publishing target. Your servlets normally are published to your Application Server's servlets subdirectory.

You can test your project after publishing by selecting the UpdateStaffInput.html file in the Studio workbench and then selecting Preview from its pop-up menu. Then, the initial input page of the project opens in your Web browser. Try using a value of 90 for the id and changing the number of years the employee has been with the company. The first time you run the servlets it will take some time to complete the first query because the JSP pages need to be compiled. Subsequent accesses should be much faster. Rerun the pages from the beginning, the number of years in the initial output page is displayed.

Conclusion

The Studio Wizards provide a useful starting point for building Web applications that can make your legacy data available to users of your Web site. The same users can be allowed to update appropriate data such as user profiles, preferences and other useful information. However, the wizards are only a starting point. After the basic pages have been built, you might want the graphic artists and content providers on your team to further enhance the generated pages using the other tools in Studio, and in particular the Page Designer, to make these pages both useful and attractive parts of your Web site.

Morgan Kinne *is an Advisory Programmer and a team leader on the WebSphere Studio development team responsible for database access and code generation aspects of the product. Morgan received his BA degree in Computer Science from SUNY Potsdam (1974) and joined IBM immediately afterwards. Since then, he has worked in many divisions with his focus in recent years on Java and Web technologies.*

Designing an architecture *for a Data Warehouse* Populating System

by Scott Howard

The article is a follow-on to "Building the Data Warehouse" published in the December 1998 issue of the Developer Connection Technical Magazine. It continues the discussion begun in that article by providing the specifics of a customizable data warehouse (DW) populating system architecture. Using the framework and disciplines of this approach, customers can weave in a diverse confederation of packaged tools and customized modules to create a consistent, enterprise class DW populating system. This system would be responsible for all aspects of the DW Extract, Transform and Load (ETL) processes including cleansing, transformation and volume data movement.

Overview

The most critical technical success factor related to a data warehouse's initial implementation and to its ongoing operational vitality is the success of the warehouse's populating system. The populating system is not only responsible for moving data into and throughout a data warehouse but also for physically enabling the warehouse data model, maintaining data integrity with respect to the business' process model, satisfying user latency requirements and integrating with the warehouse metadata model.

Enterprise class data warehouse populating systems are often a combination of many off-the-shelf tools and custom programs. These separate implementations can lead to a management nightmare unless implemented under a consistent, well-defined integration architecture. Tool vendors often use simple extract, transform and load (ETL) architectures to model the function provided by their tools. The many functions required and provided by these populating systems cannot be adequately represented by ETL alone. A more granular architecture is required to ensure that this combination of populating system components can be operated and managed as a single entity. This more granular architecture also can be used to assess a tool or custom approach's implementation and adequacy in addressing a warehouse's unique populating needs.

The management nightmare

Unfortunately, the last DW physical element that the warehouse implementor builds is the populating system. This final build occurs only after the DW requirements have been gathered, data models materialized and validated, and decision support tools prototyped and adopted. It also is the phase that is most subject to technical problems, due to its many interrelated responsibilities. Any business intelligence practitioner is quick to point out that, on average, 80 percent of the cost of building and maintaining an enterprise class data warehouse usually relates to the populating system. If all of this is true, then why isn't more attention focused on the Data Warehouse Populating System? Again the experienced practitioner will tell you that it eventually is, usually resulting in an expensive DW re-evaluation.

Many decision support tools, including today's modern (OLAP) tools, provide some sort of population or materialization process. These are fine for the dependent data marts that we explored in my previous article "Building the Data Warehouse" published in the December 1998 issue of the Developer Connection Technical Magazine. There also are many easy-to-implement extract, transform and load (ETL) tools on which implementors rely to provide robust populating processes. Most of these tools also offer control point facilities that can be used to control the many ETL processes from a single interface. However, few single tools can do it all. Unless your ETL tool can offer integrated control of all internal and external (non-tool initiated) ETL function, it becomes just another requirement for total integration. Implementing too many tools can lead to a management nightmare requiring staffs of professionals to ensure that the many populating processes function as designed and produce the desired DW results.

80 percent of the cost of building and maintaining an enterprise class data warehouse usually relates to the populating system.

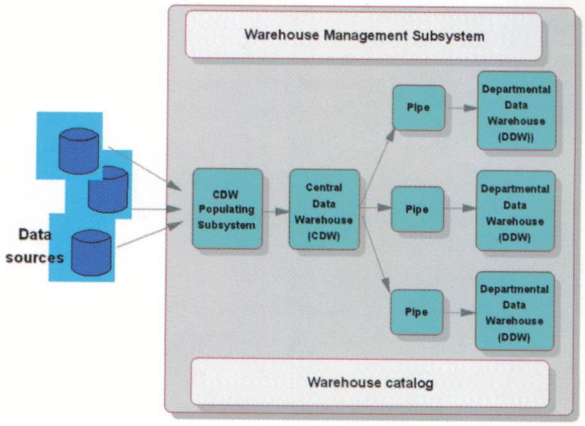


Figure 1. Support Components

Robust and manageable

Before you implement, you must develop an architecture to avoid the management nightmare. If you choose a tools-based approach, you should ensure that the various tools used for your Populating System's components can be integrated in a complementary fashion, maximizing reuse and forming a single, manageable solution. You should first define the parameters for the Populating System that will exist throughout your warehouse.

Figure 1 illustrates the multi-tier, enterprise class DW architecture advocated by many DW practitioners. Note that there is one and only one Populating System responsible for consolidating data from the operational and classified data sources. This single Populating System actually may be comprised of many tools, products and customer routines, but must be treated

CONTINUED ON PAGE 10

CONTINUED FROM PAGE 9

and managed as a single logical entity. This requires a discipline and a further refined architecture that we'll discuss shortly.

Figure 1 also illustrates a DW populating component called "Pipes" for lack of a better term. These are actually Data Mart Populating Systems. These too require unique architectures.

The Central Warehouse Populating System

There should be one, and only one, Central Warehouse Populating System. I risk overstating this but it is important. Should multiple tools and custom approaches be used, and they often are, they need to be arranged to form a common integrated solution. Suppose you had 12 different data sources from which you feed your central warehouse – a small number by most enterprise standards. That number of sources could require 12 independent populating processes. Some tools can integrate many data sources but, as stated earlier, no tool can do it all or satisfy all of your populating requirements. In either case, you need to establish an architecture that is the glue for all these independent tools and routines.

The architecture that we begin with in our Data Warehouse and Populating System courses is illustrated in ***Figure 2***. This is a simple representation of the traditional extract, transform and load architecture. Notice the common file format advocated between each of the major architectural components. Keeping data in a standard, common format ensures maximum reuse of existing tools and utilities. For example, if a standard sequential format is chosen, the many utilities that operate on sequential files can be integrated into your Populating System enhancing existing tool capabilities.

The Populating System has many responsibilities including accommodating one-to-many and many-to-one mappings, possibly consolidating them from heterogeneous sources. They also might need to reconcile or reallocate natural source keys, while recognizing source data change operations and properly representing those operations within a warehouse's life span model. That was all just technical jargon for accommodating corporate and, thus, system mergers, consolidating data from multiple platforms vendor tools and products, fixing problems introduced by source system key reuse over time and handling situations where the history of a modeled item is not contiguous.

Many other intricate complexities need to be addressed by the Populating System. The Populating System also is responsible for remodeling the business logic represented by the referential integrity (RI) implemented in your source operational applications. You may ask, why don't you just implement RI in the Central Data Warehouse? The Populating System cannot guarantee that dependent children records will arrive before their parents. These timing anomalies would result in integrity orphans and, thus, RI violations. Your CDW database would be in a constant check pending state; that's unacceptable. You'll need to design a mechanism for key reallocation and reconciliation under the auspices of the Populating System, or design complex timing dependencies and verifications. Other intricacies include support for populating cycle backout in the event of source system point-in-time recovery and the need to address VLDB issues.

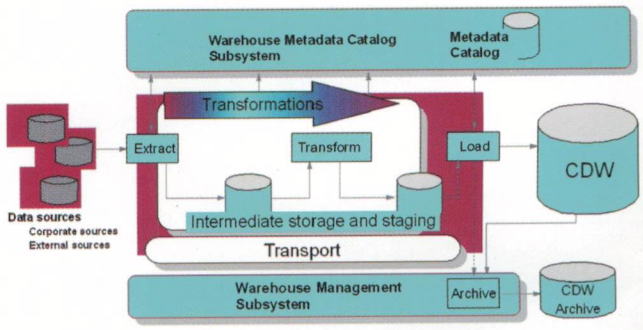


Figure 2. DW Populating Subsystem Reference Architecture

- Nine base ETL service layers

Layer 1	Layer 2	Layer 3	Layer 4	Layer 5	Layer 6	Layer 7	Layer 8	Layer 9
Extract	PreFormat	Filter	Intelligent Merge	Delta	Cleaning	Transform	Build	Load

Figure 3. Populating subsystem — Logical architecture

As you can see, the responsibilities of the Populating System are many and complex and too difficult to adequately model completely using the simple ETL architecture. In our "Populating the Data Warehouse" workshop, students discover six additional layers that can be added to the ETL model to help address these complexities while still providing an architecture that ensures maximum reuse and manageability. These layers are illustrated in ***Figure 3***.

The first layer is extract, responsible for efficiently reading data from the source files or database in a non-disruptive manner. This extract layer is frequently implemented using proprietary vendor supplied tools or utilities. The next layer, pre-format is responsible for converting the proprietary extracted data into the standard common file format, if necessary. Filter is responsible for eliminating undesired records, while intelligent merge is intended to handle many-to-one source to target mappings. Delta is responsible for detecting changed data and eliminating unchanged records. Cleaning handles all data cleansing tasks including scrubbing, format changes and re-engineering. Transform is the Populating System workhorse responsible for data derivations and conversions. It is where the source-to-target mappings are implemented. Build creates load input for whatever tool or utility will be used by Load to efficiently load data quickly and in a non-disruptive fashion.

What about my tools?

In our "Populating the Data Warehouse" workshop students spend four days evaluating and exploring these additional layers in addition to mapping these layers onto some of today's most popular ETL tools. I can't duplicate that in this article, but if I've stirred your interest on this topic, I've done my job; call it evangelism. You will most likely employ a set of tools to perform your populating system's needs. You should ensure that these tools integrate into an architecture like the one illustrated in ***Figure 3***. This allows maximum reuse, simpler management and thus a more economical and efficient implementation.

Your chosen ETL tools may reorder or bypass some of the illustrated layers. For example, some tools perform snapshot extracts. These extracts are not usable for temporal warehouse needs because they do not guarantee complete transactional capture integrity and the changes that occur between snapshots would be missed. However, these snapshot tools provide fast batch type func-

tion. Our delta layer provides a mechanism for detecting changed data and thus only transforming and loading these changes. Should you choose a replication tool that provides full transactional capture capabilities, like IBM's Data Propagator technology, then the delta layer becomes irrelevant because Data Propagator's capture component, by definition, is a delta layer. The delta layer must then be bypassed for this type of tool solution. You also should realize that the delta function is actually performed far sooner than ***Figure 3*** illustrates. The delta layer has been reordered in this example. Many other variation possibilities exist depending on the specific tool chosen and must be documented in the PS architectural solution.

Pipes

Pipes tend to be simpler to implement than the one central warehouse populating system because they generally don't deal with heterogeneous source, but rather one central DW source. Data in the central warehouse should already be clean, so that's usually not an issue. Time and other dimensional data already has been infused into the central warehouse and just needs to be replicated to the data marts. In fact, one of the toughest tasks of the pipe is remapping data from a normalized CDW structure to a denormalized star schema usually prevalent in multi-dimentional data marts. This is still not a trivial task but can often be performed by built-in processes provided with most data mart tool suites or multi-dimentional or OLAP engines.

Pipes tend to be more sensitive to service levels needed to ensure maximum availability of the data mart. Decisions need to be made about using batch type populating techniques as opposed to the alternative trickle load methods. Redundant data mart materialization combined with access switching mechanisms can also ensure 24-hour, seven-day-per-week availability, if required.

Conclusion

Building warehouse populating systems must be considered one of the most important and critical phases of warehouse construction. Ultimate Business Intelligent solution success depends on the populating system effectively and efficiently performing the nine layered functions illustrated in ***Figure 3***. Designing your populating systems following our nine layers can ensure maximum flexibility, consistency and simpler management. Consider seeking help from experienced BI practitioners or seek training specializing in populating system architecture and construction.

There are plenty of technical issues that I plan to address in future articles. Watch for discussions on natural key reallocation and reconciliation, continuous transaction capture vs. snapshots, trickle feed versus batch approaches and techniques to map source system updates and deletions into data warehouse inserts to preserve complete histories.

Scott Howard *has been with IBM for over 20 years. His experiences include staff and management assignments ranging from micro application programming to mainframe systems programming. He is an internationally recognized expert on business intelligence, data warehousing, DRDA, distributed databases and multi-vendor database implementations and a contributor to many publications. Scott is an IBM Certified Advanced Technical Expert for DB2 UDB, an IBM Certified Business Intelligence Specialist and Certified Technical Trainer. Scott is currently with IBM Learning Services and is its Business Intelligence and Data Warehouse Curricula worldwide leader. He has worked with the Santa Teresa, Toronto, Rochester and Austin IBM development labs for the past 12 years developing client/server database and business intelligence courses. Scott can be contacted at scottho@us.ibm.com.*

Populating the Data Warehouse workshop

SCOTT HAS DESIGNED AND DEVELOPED MANY BUSINESS INTELLIGENCE AND DATA WAREHOUSING WORKSHOPS FOR IBM LEARNING SERVICES. THIS ARTICLE IS BASED ON IBM COURSE DW13, POPULATING THE DATA WAREHOUSE. ARCHITECTING DATA WAREHOUSE POPULATING SYSTEMS IS ONLY ONE FACET OF THE COURSE. PRACTICAL METHODS AND TECHNIQUES FOR ACQUIRING, TRANSFORMING AND POPULATING AND A REPRESENTATIVE LOOK AT SEVERAL POPULAR IBM AND NON-IBM ETL TOOLS ALSO ARE COVERED. ALL GENERIC IBM DATA WAREHOUSE AND BUSINESS INTELLIGENCE OFFERINGS, SUCH AS DW13, ARE PRODUCT INDEPENDENT AND BASED ON THE BEST PRACTICES AND COMBINED EXPERIENCES OF IBM GLOBAL SERVICES WORLDWIDE.

GO TO HTTP://WWW-3.IBM.COM/SERVICES/LEARNING/, AND THEN SELECT YOUR COUNTRY FOR DETAILED DESCRIPTIONS AND SCHEDULES.

Classified data sources

CLASSIFIED DATA SOURCES REFER TO THE DATA THAT YOU ADD TO YOUR OWN OPERATIONAL DATA, ADDING ANALYTICAL VALUE TO YOUR WAREHOUSE. YOUR OPERATIONAL DATA SOURCES PROVIDE VALUABLE INFORMATION ABOUT THE VITALITY OF YOUR BUSINESS BUT DO LITTLE TO TRACK THE VITALITY OF YOUR COMPETITORS. HOWEVER, INFORMATION ABOUT YOUR COMPETITORS CAN BE PURCHASED FROM THIRD PARTIES, SUCH AS INDUSTRY WATCHDOGS. SEARCHING FOR THESE SOURCES OF EXTERNAL DATA IS LIKE SEARCHING THE CLASSIFIED SECTION OF YOUR NEWSPAPER, THUS THE MONIKER "CLASSIFIED SOURCES." DATA FROM THESE CLASSIFIED SOURCES ALLOW THE BUSINESS PROFESSIONAL TO ANALYZE INTERNAL CORPORATE DATA IN PERSPECTIVE WITH THAT OF THE ENTIRE RELATED INDUSTRY. THIS ENHANCEMENT TO TRADITIONAL CORPORATE DATA, IN ADDITION TO A COMPLETE TEMPORAL TRANSACTIONAL HISTORY, REPRESENTS THE MAJOR JUSTIFICATION FOR A DATA WAREHOUSE.

Installing the team repository server for VisualAge for Java on Windows NT

by Susan Yeshin

Deciding where and how to install the team repository server (EMSRV) on a Windows NT server can cause major headaches because of the many possible configurations. Breaking down the options by asking yourself the following questions can help you choose the configuration that works for your situation:

- Who will start the server?
- Should the server be started from a command line or as a Windows NT service?
- What kind of password validation, if any, is required?

After you can answer these questions, installing EMSRV is simple. This document explains some of the options you can choose for installing and starting EMSRV and provides instructions for using them. When you understand the options, follow the decision tree that leads to the steps you need to take before and during installation.

Installing EMSRV

In all cases, the first step is to copy the necessary files from the server CD to the proper directories.

Follow these steps to install EMSRV:

1. Copy the following three files from the server CD to a directory on the server:
   - EMSRV.exe
   - emadmin.exe
   - emsrvmsg.dll
2. Copy the ivj.dat repository file from the server CD to the server directory where you plan to store shared source code repositories. This directory should be specified as the EMSRV working directory, using the -W EMSRV startup parameter, when you start the server later.
3. Verify that TCP/IP is installed and correctly bound to a LAN adapter. You can verify the binding by using the ping utility to communicate with the server from a workstation on the LAN.

If you receive a response from the server, you're ready to choose and authorize the EMSRV user.

Authorizing the EMSRV user

When you start the server, you must provide the name and password of a user under whose authority the EMSRV program will run. This user is referred to as the EMSRV user. The EMSRV user is the only person who has the authority to start and stop the server.

To authorize the EMSRV user to start the repository server, do the following:

1. On the server, log onto Windows NT as an Administrator.
2. From the Start menu, select Programs – Administrative Tools (Common) – User Manager. The User Manager dialog box opens.
3. If desired, create a new user to be the EMSRV user, such as 'joe'.
4. Select User Rights from the Policies menu. The User Rights Policy dialog box opens.
5. Select the Show Advanced User Rights check box at the bottom of the dialog box, and then click the down arrow to see the Right list.
6. Select Act as part of the operating system from the list and click OK. The Grant To pane lists the users who currently have this privilege. Click Add. The Add Users and Groups dialog box opens.
7. Click Show Users. Scroll down the list of users in the Names pane and select the EMSRV user (joe) from the list. Click Add. Click OK to close all of the open dialog boxes.
8. Reboot the machine.

Deciding where and how to install EMSRV can be challenging...these configuration instructions can simplify EMSRV installation and help ensure that your efforts are successful.

User 'joe' now has the Windows NT operating system privileges needed to start the repository server.

After EMSRV is installed and the EMSRV user is authorized, you need to consider how EMSRV should start. You can start EMSRV from a command line, adding the parameters as necessary, or you can install EMSRV as a Windows NT service that starts automatically on startup. (You also can install EMSRV as a service that starts manually, but that option is not discussed in this article.)

Installing EMSRV as a service

There are two advantages to installing EMSRV as a service:

- You can specify automatic startup so that EMSRV starts whenever the repository server is rebooted.
- You can specify the default settings that you want EMSRV to use. For example, you might want to specify that password validation is always enabled.

To install EMSRV as a service, follow these steps:

1. Change to the directory where the EMSRV executable program is installed and issue the command

```
EMSRV -install [parameter2] [parameter3] ...
```

The first parameter must be

```
-install;
```

the others are the EMSRV startup parameters that you have chosen for your environment. Here is an example:

```
EMSRV -install -u joe -p donttell -W j:\javateam
```

This example installs EMSRV as a service in the Windows NT registry, with

```
joe
```

as the EMSRV user name and

```
donttell
```

as joe's password. By default, the EMSRV working directory will be

```
j:\javateam.
```

A message will confirm that EMSRV has been installed.

2. From the Windows NT Control Panel, double-click Services. The Services dialog box opens.
3. Select EMSRV from the list of services. In the Startup Parameters text box, type the EMSRV startup parameters that you want to use. If you are specifying the working directory for EMSRV to use, you must type an extra backslash for each backslash in the path. If you want your working directory to be e:\teamproject\emsrv\working type e:\\teamproject\\emsrv\\working
4. Click Start. A message appears, informing you that EMSRV is starting.

EMSRV is now installed as a service in the registry and the necessary DLLs have been copied to the system directory. The parameters that you provided will be used, by default, whenever EMSRV is started. You also can override or add to these parameters if you start EMSRV manually from the Services icon of the Windows NT Control Panel.

The next step is to decide what kind of password validation you need in order to ensure that the code on the server is secure. When password validation is enabled, team members must provide valid passwords to connect to a repository managed by that server. There are three options for password validation:

- No password validation required.
- Native operating system accounts and passwords are used.
- Passwords are stored in a file on the server.

If you decide not to enable password validation, any VisualAge for Java client can access the Java source code on the server and can make changes to that code.

Using native password validation

If you choose to use native operating system password validation, team members must enter their Windows NT logon user names and passwords to connect to any repository on that server.

To set up native password validation, do the following:

1. Create a Windows NT user ID for each user on the server.
2. In VisualAge for Java, add each user to the repository list –
   - In the User Administration dialog box (**Window > Repository Explorer > Admin > Users**), enter a unique name and full name, and provide the Windows NT ID as the Network Login Name.

When you start EMSRV using the password validation option (-rn), users are prompted for their system passwords when they connect to a shared repository on the server.

Using a password validation file

As an alternative to using native operating system passwords, a file containing a list of team members' user names and passwords can be maintained. This file is called passwd.dat.

The passwd.dat file resides in the EMSRV working directory. There is one passwd.dat file per server; that file is used for all shared repositories on the same server. The passwd.dat file contains one entry per team member, with one user name and password per line. The user name is specified first, separated from the password by a single space. Here is an example:

```
fred mypassword
wilma hello
betty ZXF65
```

The passwd.dat file is not encrypted. Passwords should not be the users' network login passwords.

To set up a password validation file, do the following:

1. Copy the sample password file, passwd.dat, from the server CD to the same directory where you copied ivj.dat.
2. Open the passwd.dat file in Notepad and create an entry for each user in the passwd.dat file on that server.
3. When this file is created, add each user to the repository user list in VisualAge for Java (**Window -- Repository Explorer -- Admin -- Users**), providing each name from the passwd.dat file as the Network Login Name in the User Administration dialog box.

When you start EMSRV using the password validation option (-rn), users are prompted for the password when they connect to a shared repository on the server.

The decision tree

Use the decision tree shown in ***Figure 1*** to find out which configuration of EMSRV best suits your needs. When you arrive at a number, follow the corresponding instructions for installing and starting EMSRV.

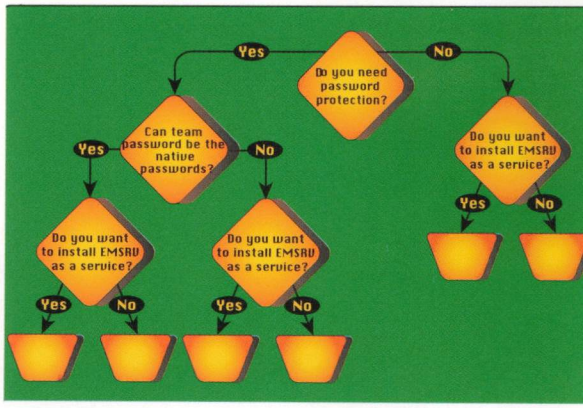


Figure 1. Decision tree to select EMSRV configuratio

Scenario 1: EMSRV starts from the command line. No passwords required.

1. Copy the necessary files from the server CD.
2. Authorize the EMSRV user (joe).
3. To start the server, change to the directory where the EMSRV executable program is installed and type:

```
EMSRV -u joe -p donttell -W d:\javateam
```

where the repository server is under the authority of a user called **joe** and joe's password is **donttell**. The working directory where EMSRV will write its log and where it will search for repositories is d:\javateam.

Scenario 2: EMSRV is added as a service. No passwords required.

1. Copy the required files from the server CD.
2. Authorize the EMSRV user (joe).
3. Change to the directory where the EMSRV executable program is installed and type:

```
EMSRV -install -u joe -p donttell -W j:\javateam
```

where the repository server is under the authority of a user called **joe** and joe's password is **donttell**. The working directory where EMSRV writes its log and where it searches for repositories is d:\javateam.

4. Reboot to start the server.

Scenario 3: EMSRV is added as a service. Passwords are the Windows NT logon passwords.

1. Copy the required files from the server CD.

CONTINUED ON PAGE 14

CONTINUED FROM PAGE 13

2. Authorize the EMSRV user.
3. Change to the directory where the EMSRV executable program is installed and type:

```
EMSRV -install -u joe -p donttell -W j:\javateam -rn
```

where the repository server is under the authority of a user called **joe** and joe's password is **donttell**. The working directory where EMSRV writes its log and where it searches for repositories is d:\javateam.

4. Create the necessary accounts to enable native password validation.
5. Reboot to start the server.

Scenario 4: EMSRV is added as a service. Passwords are stored in the passwd.dat file.

1. Copy the required files from the server CD.
2. Copy the passwd.dat file from the server CD to a directory on your server.
3. Edit the passwd.dat file and add each user to the repository user list.
4. Change to the directory where the EMSRV executable program is installed and type:

```
EMSRV -install -u joe -p donttell -W j:\javateam -rp
```

where the repository server is under the authority of a user called **joe** and joe's password is **donttell**. The working directory where EMSRV writes its log and where it searches for repositories is d:\javateam.

5. Reboot to start the server.

Scenario 5: EMSRV starts from the command line. Passwords are the Windows NT logon passwords.

1. Copy the required files from the server CD.
2. Create the necessary accounts to enable native password validation.
3. To start the server, change to the directory where the EMSRV executable program is installed and type:

```
EMSRV -u emsrvacc -p secret -W d:\javateam -rn
```

where the repository server is under the authority of a user called **joe** and joe's password is **donttell**. The working directory where EMSRV writes its log and where it searches for repositories is d:\javateam.

Scenario 6: EMSRV starts from the command line. Passwords are stored in the passwd.dat file.

1. Copy the required files from the server CD.
2. Copy the passwd.dat file from the server CD to a directory on your server.
3. Edit the passwd.dat file and add each user to the repository user list.
4. To start the server, change to the directory where the EMSRV executable program is installed and type:

```
EMSRV -u emsrvacc -p secret -W d:\javateam -rp
```

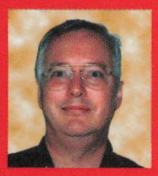
where the repository server is under the authority of a user called **joe** and joe's password is **donttell**. The working directory where EMSRV writes its log and where it searches for repositories is d:\javateam.

Summary

Deciding where and how to install EMSRV on a Windows NT server can be a challenging and at times confusing task, especially given the large number of possible configuration options. However, the configuration instructions presented in this article can simplify your EMSRV installation efforts and help ensure that your efforts are ultimately successful.

Susan Yeshin *has been an information developer on the VisualAge for Java team for the past two years. Currently, she writes online help for various VisualAge for Java components, including the IDE and Team Programing. Questions about this article, can be sent to Susan at syeshin@ca.ibm.com.*

DTDs in the realm of XML



by Don Day

Someone recently asked, *"I create my XML documents by converting them from clean legacy data. So, as long as they are well-formed and my tools are written to handle the tags that I know are in them, why do I need to bother with Document Type Definitions, or DTDs?"*

If your documents mainly consist of messages between applications or between servers, the right answer may be, "Don't bother with DTDs at all." Many programmers write processing applications that implicitly handle all expected deviations within their data. In effect, the program logic contains the "business rules" for the structure of your XML documents. But what if you could externalize those implicit rules so that you could use them as the basis for reconfiguring your tools, or so that others could interchange data more reliably with you? That principle of external declarations is what makes DTDs something you should know more about.

Why do I need a DTD?

DTDs are valuable to you because:

- DTDs specify your document model. A "Document Type" simply describes which elements are allowed and how they are organized in an XML document, much like a class in a Java program. A DTD can guide you as you develop new applications for processing your documents. For example, if you have an application development tool for Java classes and other developers want to modify their DB connectivity builder to integrate with your tool, they can use the DTD to determine the format requirements and to ensure validity, rather than having to interview you or try to read your source code. As another example, you might consult a DTD to ensure that you have developed all the required template rules for an XSLT stylesheet used to process your XML data. Finally, by using a DTD to manage the document structure, you make it easy to maintain your current and future document libraries. DTDs become schemas for setting up the storage structures for the documents they manage. In these ways, a DTD, along with explanatory documents about your design intents, is valuable to the tools developer and document owners.
- DTDs enable you to validate document content. Many XML documents are machine-generated from structured legacy data; as a result, you might expect the XML output to be inherently well structured. However, legacy formats are rarely "clean" to start with; in many cases, the corresponding XML-generated structures may be illogical. For example, bold, italic phrases in word processor documents may come through conversion tools looking like "<B><cite>Hello, world</CITE>," which fails the XML requirement for proper nesting and for matching case in the tag names. By validating documents against a DTD, you can locate errors in migrated content and correct either the legacy markup or the conversion tools.
- A DTD helps to keep the data and your processes consistent by:
  - Enabling you to determine quickly whether the document generators are generating data that is structured as you expected
  - Allowing the data generation and processing functions to be developed more independently with greater assurance that the two will work together when brought together
  - Letting you test everything you can in the incoming data – you can introduce additional checks when you need them for problem diagnosis in the production system
  - Giving you an easy way to check the validity of any test data you must create
- DTDs are required for editors that validate XML. Many free or low-cost XML editors attempt to check syntax but are unable to prevent writers from inserting illogical structures or unallowed attribute values as they draft their XML content. A validating editor relies on a DTD to check structural context at the insertion point. The editor then provides menus that show the elements allowed at the insertion point. DTDs enable writers to spend more time writing and less time checking their tags. They provide value to data owners, giving them assurance that the documents are valid as saved.
- DTDs are useful for validation checking. Validation helps assure that client-side tools have fail-safe data to process. Valid data should decrease the need for your applets to perform error-checking and recovery.

DTDs can guide you in the development of new applicaions, validate document content and assure that client-side tools have fail-safe data to process.

How can I create a DTD?

Your first concern may be, "Where can I get a good book about DTDs?" You wouldn't expect to learn the craft from a specification, any more than you would expect an automotive parts catalog to help you build a car from scratch. If you are new to DTD designing, you might want to work with someone who has had some previous SGML or XML experience and consult a good book. A good book to start with (albeit with an SGML emphasis) is *Developing SGML DTDs: From Text to Model to Markup*, by Eve Mahler and Jeanne El Andaloussi. More specific to XML is *Inside XML DTDs: Scientific and Technical*, by Simon St. Laurent.

If you are a programmer who prefers the school of raw experience, there is no better way to learn the ins and outs of XML DTD

CONTINUED ON PAGE 16

CONTINUED FROM PAGE 15

writing than by downloading a copy of the XML 1.0 Recommendation (http://www.w3.org/TR/1998/REC-xml-19980210/) and reading about XML DTDs from the syntax up. You still will need other books to help with design strategies.

Let's say you are more deductive...you would like to analyze your documents before you form your conclusions about the right DTD. Several vendors offer tools that take your source XML documents as input and produce a DTD that represents the composite structure of all the sample documents. You are then free to modify the DTD to more closely represent your larger body of documents. One of the first tools to accomplish this analysis was "Fred: The SGML Grammar Builder" from OCLC (http://www.oclc.org/fred/).

Another approach is to begin your design work with a tool that interactively leads you through the design process to create a DTD that suits the document model you have in mind. This might be an alternative if you have no prior document structures to work from (or if you just want to make a clean start on the design for future documents). Tools in this category typically have a visual design interface that allows you to build a DTD progressively. Design tools in this category include Near & Far Designer (http://www.microstar.com), XML Authority (http://www.extensibility.com/products/xml_authority.htm) and IBM's Visual DTD (http://alphaWorks.ibm.com/).

Finally, let's say you already have an SGML DTD for your data and you want to adapt it to XML form. This is less tedious than writing a DTD from scratch without the help of tools. However, you will need some help that the XML 1.0 Recommendation does not provide. Browse for an article called "Converting an SGML DTD to XML," by Norman Walsh at www.xml.com. That article leads you through a case study of a conversion and touches on all the syntax and alternative approaches you might consider as you migrate an existing SGML DTD.

Having done all this design work, you might want to test it. One particular tool, IBM's XML Generator (http://alphaworks.ibm.com/), generates sample document instances that conform to the design. You can use the instances as test cases for processing tools. This helps to create a tightly integrated processing environment.

How do XML DTDs differ from SGML DTDs?

A DTD is a "parser generator specification" that sets up a syntax and vocabularies that support parsing conforming classes of data. In other words, it specifies the allowed structure and content for an XML document instance. You might think of a DTD as a *map* that represents relationships within the data (useful in building tools that generate hypertext), as a *fence* that constrains how new documents should be generated and as a *filter* for validating the compliance of existing documents.

XML DTDs and SGML DTDs both provide these benefits. The main functional differences, then, are due to XML being seen more as a delivery language like HTML rather than an authoring language like SGML. Moreover, XML DTDs and documents are less complex; correspondingly XML documents are much easier to process and to write tools for.

So how are XML DTDs less functional or expressive than SGML DTDs?

- No conref support required of the parser. This is provided, instead, through XLink and XSL, so no net loss. (Conref stands for "content reference," an attribute that copies other content in place of an empty start tag.)
- No shortrefs, which basically allowed SGML to be used as Grep processor for weakly formed source (not an issue for generated XML or XML managed by a decent editor. XML parsing gives the information processing community long sought enforcement for well-formed content. So this is also implicitly on the "good" list!)
- No inclusions or exclusions in content models. (This is unfortunate; XML is weaker for enforcing business rules in validating editors. But authoring is not XML's primary domain; if it is generated from SGML where the business rules ARE applied, then you get the desired control of delivered content.)
- No '&' (and) connector, which requires explicit sequences of elements. This has been difficult for vendors to implement; few tears have been shed over this limitation.
- You can't write a single declaration for a set of elements that have the same content model; you have to repeat the declaration for each element. (This was a shortcut in SGML; in XML, you can assign the content model to a parameter entity and just reference the entity for each element declaration. Same maintenance benefit, different method.)
- Ditto for attributes. Same solution.
- Attribute typing is more limited in XML. (This should be rectified in the Schema work by W3C, but vendor support in tools will be months down the road.)

And what's uniquely good about XML DTDs?

- Same syntax and design as SGML; therefore, they present a good opportunity for you to reuse existing SGML skills.
- Generally easier to parse and apply than SGML shortrefs, '&' connector, omission rules, etc.; therefore more vendors have validating parsers.
- No shortrefs. (See comments for this same entry in the preceding list.)
- They still allow use of parameter entities, which is a bonus for managing DTDs.
- XML is short for "Extensible Markup Language," and extensibility means that you can create Web-based applications that can merge content from other XML providers. The World Wide Web Consortium has defined a namespace recommendation for applications as a roadmap for developing extensible applications. A namespace provides unique qualification of elements whose structure and content conforms to a particular DTD.
- All the other good things about SGML (XML rides on the shoulders of an enterprise-strength giant!)

What are the top ten stumbling blocks in constructing XML DTDs?

1. Defining an element but not referencing it in any content model -- in other words, like having dead code in a program.
2. Defining attributes for content that might be better cast as an element. Many early SGML DTDs used attributes to contain subtitle content, but such attribute content is not allowed to contain markup; therefore this limits the usefulness of that subtitle content.
3. Defining everything as an element. (OO programmers often think this way; inheritance comes by family relationship rather than by attribution.) But in our non-linear world, things don't always line up this way.
4. Mislabeling attribute types. ("Hmmm, should this be NAME or NAMES or NMTOKEN or NMTOKENS?") If it is not clear how to type an attribute, it's best to consult the XML recommendation, another

DTD guru or a newsgroup such as comp.text.xml.

5. Making all elements required. In authoring, this turns a document into one massive form or template -- no flexibility. On one hand, this may be a useful trait, such as ensuring that a form always expands to show all required sections, but for rich document content, this sets up limiting contexts for writing.
6. Some tools require putting a doctype and subset container around the DTD; other tools require NOT putting a doctype and subset container around the DTD (due to different interpretations about doctype in the XML Recommendation).
7. Not allowing multiple occurrences of things in an OR group (use one element and then you can't use any others in that context).
8. Using the "1 or more" occurrence indicator for an optional element. (This indicator might not cause a compile error.)
9. Failing to use a "0 or more" occurrence indicator on content models that contain PCDATA <!ELEMENT Para (#PCDATA, Emph, BulletList, Table)><!-- should be ..." Table)*>" -->
10. Creating inadvertent 'recursive' elements. Recursion is certainly allowed and you may have an intentional need for it. When it is not intended, it may occur because of the use of parameter entities or may occur when an element is allowed in lower levels of contained elements.

Above all, just be mindful not to dream up something so complex that nobody loves it or wants to use it. XML DTDs are as much for people as they are for the data that people are asked to author or write tools for!

What's on the horizon?

The World Wide Web Consortium (W3C) has chartered a working group to develop a schema mechanism to replace DTDs. Schemas express more information about the meanings of elements and relationships that cannot be expressed by the peculiar, Grep-like syntax of DTDs. Development on this standard has been slow because developers and vendors have to confront the difficult parts of implementing XML. The current draft may be obtained at http://www.w3.org/TR/xmlschema-1/. This article is worth reading to help you get ready for the schema-based tools that are expected to roll out in the coming year.

Once you have XML documents that are both valid and well formed, the next step is to use that data. Most of the browsers that customers use to read your data will not be natively able to read XML. Your next goal as a developer will be to learn more about the transformational component of the Extensible Stylesheet Language, or XSLT (http://www.w3.org/TR/xslt).

Don Day *is an advisory software engineer for IBM. For the past 15 years, he has designed and supported publishing tools for IBM's Information Development community. Don provides XML expertise for Information Design and Development in IBM's e-business Operating Systems Solutions area (located in Austin, Texas) and for IBM Corporate User Technology. He has represented IBM on the W3C XSL Working Group and is presently IBM's alternate rep for the W3C CSS Working Group. Don holds a dual-major Bachelor of Arts degree in English and Journalism and a Master of Arts degree in Technical and Professional Communication (with a minor in Computer Science) from New Mexico State University. You can contact Don at dond@us.ibm.com.*

IBM LEADS IN U.S. PATENTS FOR SEVENTH CONSECUTIVE YEAR

FOR THE SEVENTH CONSECUTIVE YEAR, IBM WAS AWARDED THE MOST U.S. PATENTS IN 1999. WITH A RECORD 2,756 PATENTS ISSUED BY THE U.S. PATENT AND TRADEMARK OFFICE, IBM TOPPED THE NEXT CLOSEST COMPANY BY MORE THAN 900 PATENTS.

THE COMPANY WAS AWARDED A TOTAL OF MORE THAN 15,000 PATENTS DURING THE 1990S, TRIPLING ITS OUTPUT OF THE PREVIOUS DECADE AND 2,300 MORE THAN THE NUMBER TWO PRODUCER OF PATENTS, CANON.

THE RESULTS WERE REPORTED TODAY BY IFI CLAIMS PATENT SERVICES, WHICH COMPILES THE CLAIMS PATENT DATABASE AND ANNUALLY REPORTS THE NUMBER OF PATENTS ISSUED TO COMPANIES.

A COMPANY SPOKESPERSON CITED THE ROLE OF IBM'S PATENT PORTFOLIO IN MORE THAN $30 BILLION WORTH OF OEM AGREEMENTS SIGNED BY THE IBM TECHNOLOGY GROUP IN 1999. IN ADDITION, PATENT AND INTELLECTUAL PROPERTY LICENSING EFFORTS GENERATE MORE THAN A BILLION DOLLARS IN REVENUE ANNUALLY.

IN ADDITION TO HARDWARE AND COMPONENTS, IBM'S 1999 PATENT PORTFOLIO INCLUDES MORE THAN 900 SOFTWARE-RELATED PATENTS THAT ARE FUNDAMENTAL TO THE COMPANY'S E-BUSINESS STRATEGY. BY COMPARISON, IFI CLAIMS RECORDS SHOW THAT MICROSOFT CORPORATION WAS AWARDED 353 PATENTS IN 1999, RANKING IT 38TH AMONG COMPANIES, AND ORACLE WAS NOT ISSUED ENOUGH PATENTS TO QUALIFY FOR THE TOP 50.

The Pervasive Computing Paradigm

by Bill Bodin

By now you may have heard the quote from our CEO, Lou Gerstner, "....Picture a day when a billion people will interact with a million e-businesses via a trillion interconnected devices...." The challenge this presents to the IBM Pervasive Computing Division is the creation of an architecture that provides an end-to-end capability to deliver content and function to and from these intelligent devices. This architecture must implement aspects of reliability, security and information scaling in such a way that the user doesn't think about how the data is delivered; rather, the user is simply enabled to make intelligent and informed decisions based on the "pervasive" availability of the data. This article will explore both the application development opportunities and the environments within which these applications exist and interact.

Pervasive applications

As you can imagine, the applications in support of this pervasive paradigm are seemingly infinite in number. While the main focus areas are Automobile Network Solutions, Networked Home, and Mobile e-business, there are many application and device support opportunities that comprise sub-elements of these categories. Let's take a closer look at these focus areas.

Automobile network solutions

For the most part, today's "in-vehicle connectivity" is achieved through a collection of pagers and cell phones. These devices allow us to stay in touch with people who require our interaction. This interaction is based largely on issues and events that arise spontaneously in our environment. As much as this allows us to "stay in touch," it also is largely responsible for the steady stream of non-productive interruptions that seem to predominate. The fundamental problem is that while these devices allow you to interact, they, historically, do not allow you to make informed decisions based on numerous pieces of related data.

Imagine it's 9 p.m. at night and you're driving on an extended trip. You're scheduled to stop for the night at 11 p.m. An hour ahead you'll run into a strong band of thunderstorms. With Pervasive technology, your "networked vehicle" would have communicated with Doppler weather servers to inform you of the severity of the impending weather condition and offer you a means to consider a number of different options including detours, hotel reservations changes and itinerary management. The services mentioned in this case, weather information, hotel reservations, and itinerary management, would all be examples of "vertical market" applications that are a natural fit into Pervasive Computing. The appliance, which surfaces the data in this scenario, will become as ubiquitous as your cell phone and as common as your car radio. In fact, it is a merger of these two current technologies, and many more, that will facilitate the entire pervasive information revolution. In-car radio appliances will allow custom and configurable access to news, stock quotes, streaming data, books on demand, ticket purchasing, food ordering, reservations and many more services. The IBM Pervasive Computing open architecture will support this "information revolution" from the client to the middleware applications through to the enterprise servers.

Networked home

The IBM Networked Home initiative includes the specification of a reference platform referred to as the "Service Gateway." This gateway device comprises a processor, memory, network connections, and more. The software stack includes an embedded Web server with Java servlet capability.

Pervasive Computing architecture must implement aspects of reliability, security and information scaling in such a way that the user doesn't think about how the data is delivered but enables the user to make intelligent and informed decisions.

The Service Gateway, like the automotive appliance mentioned above, is capable of managing numerous in-home or enterprise tasks. In the household environment, the Service Gateway, when equipped with a powerline protocol modem, is capable of managing your "enabled" appliances using no new wires. With a suite of powerline protocol vertical applications (Java servlets or native) implemented on the gateway, you could, for instance, adjust your thermostat from anywhere in the world or invoke a simulation of your living patterns to give your home a "lived in" look when you're away. In this case the gateway could serve HTML pages to a browser or present servlet entry points to an Internet-enabled cell phone. With the Pervasive application programming model accommodating both Java and native interfaces, the developer now has a wide array of possible implementation opportunities.

Enterprises can look forward to the Service Gateway coalescing data, managing inventory, and dispatching delivery personnel in vending-related services.

Other vertical opportunities include managing multiple PPP connections from a single ISP, Virtual Private Networking (VPN) Support, providing collateral support for thin client Screenphones, Intelligent Agents, maintaining an "always-on" Web presence, and much more.

Mobile e-business

Nowhere in the Pervasive Computing paradigm is information scaling more important than in the Mobile e-business arena. This focus area comprises devices such as pagers, conventional cell phones, Internet-enabled cell phones, personal digital assistants (PDAs), and other similar devices.

Enhancements in the use of these devices is due in part to the use of XML. XML provides an important advantage over HTML as the essential aspects of the data are actually an

integral part of the data. When XML is used as the mark-up language for server-side content, components, such as proxies or more specifically "transcoding" proxies, can be utilized to serve an appropriate view of the data to a particular class of devices. For instance, let's say that a real estate agency would like to make data available to their field agents using low-cost hand-held PDAs. Depending on the bandwidth and graphical display capabilities of these devices, the information that is actually delivered may differ from device to device. A typical IBM WorkPad* delivers content in a monochrome 160 x 160 pixel window, while an IBM ThinkPad* (representing a very robust client platform) presents data at SVGA resolutions and above. A transcoding proxy can make dynamic determinations, based on the clients' capabilities, and then converts large graphics and voluminous text into data appropriate for the receiving client. It is important to consider the device characteristics in the design phase. In a Pervasive Computing enterprise, these characteristics affect the architecture of the backend data store as well as the server-side intelligence (proxies) that may exist to serve a heterogeneous mix of clients.

Current solutions that exist for both Palm Computing Platforms and Windows CE are IBM Mobile Connect, IBM Mobile Net Connect, IBM NuOffice, and IBM SecureWay.* For more information on these products visit http://www.ibm.com/pvc. Look for more information on transcoding and the relevancy of XML to Pervasive Computing in future articles.

***Bill Bodin**, a Senior Technical Staff Member, leads the research and development efforts for the Pervasive Computing Division at IBM in Austin, Texas. He is currently involved in bringing key technologies to IBM's Pervasive Development Organizations. The Pervasive Advanced Technology Laboratory provides the conduit through which many of these technologies flow. Bill has been awarded several United States Ppatents across many disciplines related to computer operating systems and related platforms with several additional patents pending. He is a frequently requested speaker at many technical developer conferences. You can contact him at bbodin@us.ibm.com.*

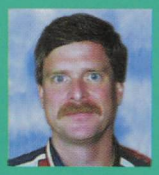
Pervasive Computing's Advanced Technology Laboratory

THE "PROVING GROUND" FOR MANY OF THE PERVASIVE TECHNOLOGIES MENTIONED IN THIS ARTICLE IS THE PERVASIVE COMPUTING ADVANCED TECHNOLOGY LABORATORY. THIS LAB, LOCATED IN AUSTIN, TEXAS, INTEGRATES AND INNOVATES LEADING LEADING-EDGE TECHNOLOGIES INTO BOTH RESIDENTIAL AND ENTERPRISE SCENARIOS. THIS LAB DEMONSTRATES TECHNOLOGIES SUCH AS HOME AUTOMATION POWERLINE PROTOCOL DEVICES, INFRARED HOME THEATER MANAGEMENT, WIRELESS DATA COMMUNICATIONS, SCREENPHONE, SCREENFRIDGE, NETWORKED VEHICLE, AND MORE. THESE TECHNOLOGIES SHARE A CROSS-PLATFORM ARCHITECTURE IMPLEMENTATION USING BOTH JAVA AND NATIVE COMPONENTS. MANY OF THESE TECHNOLOGIES REPRESENT VERY THIN CLIENT ARCHITECTURES USING JAVA SERVLETS. THESE JAVA SERVLETS PROVIDE DIRECT ENTRY POINTS TO ACCOMPLISH SPECIFIC TASKS. THIS IMPLEMENTATION ALLOWS ACCESS BY THE WIDEST POSSIBLE RANGE OF NETWORK-ENABLED DEVICES SUCH AS WIRELESS PDAS, INTERNET-ENABLED CELL PHONES, NETWORKED VEHICLES, AND OTHERS.

VISIT IBM'S INTERNET RESOURCE AT HTTP://WWW.IBM.COM/PVC FOR UPDATES AND ANNOUNCEMENTS REGARDING THE PERVASIVE COMPUTING ADVANCED TECHNOLOGY LAB.

LOOK FOR MORE DETAIL ON DESIGNING FOR THE EMBEDDED ENVIRONMENT, XML AND PERVASIVE COMPUTING, THE IBM PERVASIVE CLIENT STACK, AND SERVER SIDE COMPONENTS IN FUTURE ARTICLES IN THIS PUBLICATION.

Java cryptography Part 2: Key generation and management



by Anthony Nadalin, Theodore Shrader and Bruce Rich

Beginning with Version 1.1 of Sun's® Java Development Kit (JDK), general purpose APIs for cryptographic functions, collectively known as the Java Cryptography Architecture (JCA), were provided along with their extension, the Java Cryptography Extension (JCE). The Java 2 Standard Development Kit (SDK) significantly enhances the Java Cryptography Architecture. Specifically, the Java 2 SDK augments the certificate management infrastructure to support X.509 V3 certificates.

Note: *The Java Development Kits (JDKs) are referred to as Standard Development Kits (SDKs) in Java 2.*

This article continues our discussion of Java Cryptography and describes some of the Key generation and Management classes used in advanced encryption and decryption and how to create a program exploiting these features. For more information, see "Java cryptography Part 1: Encryption and decryption," published in the January 2000 issue of the Developer Connection Technical Magazine. An HTML version of the magazine is available on the Developer Connection Web site at www.developer.ibm.com/devcon/mag.htm

Secret key interfaces and classes

JCE 1.2 offers a set of classes and interfaces to manage secret keys. Also known as symmetric or private keys, secret keys are used at both ends of the cryptographic process. The secret key that encrypts contents also can be used to decrypt the encrypted contents.

- The javax.crypto.SecretKey interface contains no methods or constants. Its only purpose is to group and provide type safety for secret keys. Provider implementations of this interface must overwrite the equals() and hashCode() methods inherited from java.lang.Object, so that secret keys are compared based on their underlying key material and not based on reference. Since it extends the Key interface, this interface is an opaque representation of a symmetric key.
- The javax.crypto.SecretKeyFactory class represents a factory for secret keys. Key factories are bidirectional, which means that they allow building of an opaque Key object from a given key specification (key material) or retrieving the underlying key material of a Key object in a suitable format.

In general, key factories are used to convert keys (opaque cryptographic keys of type Key) into key specifications (transparent representations of the underlying key material) and vice versa. In particular, secret key factories operate only on secret keys.

The javax.crypto.spec.SecretKeySpec class specifies a secret key in a provider-independent fashion. It can be used to construct a SecretKey from a byte array, without the need to go through a provider-based SecretKeyFactory. This class is useful only for raw secret keys that can be represented as a byte array and have no key parameters associated with them, for example DES or Triple DES keys. This class is a transparent representation of a secret key.

Java Cryptography describes Key generation and management classes used in advanced encryption and decryption and how to create a program exploiting these features.

The KeyGenerator class

The java.security.KeyPairGenerator class, which is part of the Java 2 SDK, Standard Edition, V1.2 APIs, is used to generate a pair of public and private keys. Key pairs differ from secret keys in that one key is used to decript the encrypted data stream or vice versa. JCE 1.2 provides for a KeyGenerator engine class which is used to generate secret keys for symmetric algorithms. KeyGenerator objects are created using the getInstance() factory method of the KeyGenerator class. Notice that a factory method is by definition static.

The getInstance() method takes as its argument the name of a symmetric algorithm for which a secret key is to be generated. Optionally, a provider name may be specified. If just an algorithm name is specified, the system will determine if there is an implementation of the requested key generator available in the environment and, if there is more than one, the preferred one will be selected. If both an algorithm name and a package provider are specified, the system will determine if there is an implementation of the requested key generator from the requested provider and throw an exception if there is not. A key generator for a particular symmetric-key algorithm creates a symmetric key that can be used with that algorithm. It also associates algorithm-specific parameters (if any) with the generated key.

The KeyAgreement class

Whenever two or more parties decide to initiate a secure conversation over a non-secure communication channel, they need to use the same secret key (which is called the session key), without transmitting it in the clear over the channel. To achieve this, public-key encryption could be used to transmit the session key securely.

Alternatively, another solution is to use key agreement. A key agreement is a protocol that allows two or more parties to calculate the same secret value without exchanging it

directly. Therefore, the parties share the same secret key and can encrypt the communication using symmetric encryption. The most famous of these protocols is the Diffie-Hellman (DH) algorithm, an implementation of which is provided by Sun in JCE 1.2 reference implementation.

The javax.crypto.KeyAgreement class provides the functionality of a key agreement protocol. The keys involved in establishing a shared secret key are created by one of the key generators (KeyPairGenerator or KeyGenerator), a key factory, or as a result from an intermediate phase of the key agreement protocol.

Each party involved in the key agreement has to create a KeyAgreement object. This can be done using the getInstance() factory method of the KeyAgreement class. This method accepts as its argument a string representing a key agreement algorithm as parameter. As with the KeyGenerator class, you can specify a provider as the second argument.

If the Diffie-Hellman algorithm is used, a Diffie-Hellman private key is used to initialize the KeyAgreement object. Additional initialization information may contain a source of randomness and/or a set of algorithm parameters.

Every key agreement protocol consists of a number of phases that need to be executed by each party involved in the key agreement. The doPhase() method is used to execute the next phase in the key agreement. This method takes two arguments: a Key and a boolean.

- The Key argument contains the key to be processed by that phase. In most cases, this is the public key of one of the other parties involved in the key agreement or an intermediate key that was generated by a previous phase. The doPhase() method may return an intermediate key that you may have to send to the other parties of this key agreement, so they can process it in a subsequent phase.
- The boolean parameter specifies whether or not the phase to be executed is the last one in the key agreement.
- A value of false indicates that this is not the last phase of the key agreement and there are more phases to follow.
- A value of true indicates that this is the last phase of the key agreement and the key agreement is completed.

After each party has executed all of the required key agreement phases, the secret key can be computed by calling the generateSecret() method.

Practical example of Java Cryptography

This section describes a couple of examples of the kinds of applications for which the JCA can be used.

In Part I of this Java Cryptography article series, we provided an example where a text file in the same class was encrypted and then decrypted. Real-life situations are more complex. A realistic situation would be when two persons are situated at two different locations and want to exchange data safely by encrypting the data during transmission. In this situation, there are two different programs for encryption and decryption.

Cryptography scenario

In the first example, consider the following scenario. Bob and Alice, want to exchange data. Bob wants to send data to Alice and wants to encrypt the data to maintain its safety in transit. He writes a program called Encrypt1.java which does the following:

1. Reads data from a text file JavaTeam.txt
2. Encrypts the data using a Cipher object initialized by the DES algorithm and a DES secret symmetric key
3. Stores the encrypted data in a file bob.enc
4. Stores the secret key in another file bob.key

The file JavaTeam.txt read and encrypted by the Encrypt1 Java program:

```
Anthony Nadalin
Ted Shrader
Bruce Rich
```

Bob runs his program and then sends Alice the two files bob.enc and bob.key. Alice, at the receiving end, writes a program named Decrypt1.java that does the following:

1. Reads the data from the encrypted file bob.enc
2. Reads the key from the bob.key file
3. Initiates a Cipher object using this key and decrypts the data from bob.enc
4. Writes the decrypted data into a file bob.dec

Bob's program

Bob's program, Encrypt1.java is shown in *Listing 1.*

Bob issues the javac command to compile this program.

```
java Encrypt1.java
```

```
import java.io.*;

import java.security.*;
import javax.crypto.*;

class Encrypt1
{
  public static void main(String args[])
{

    if (args.length != 3)
      System.out.println("Usage: java Encrypt1 inputFileName
        encryptedFilekeyFile");
    else
      try
      {
        // generate a Cipher object
        Cipher jcecipher = Cipher.getInstance("DES/ECB/NoPadding");

        // generate a KeyGenerator object
        KeyGenerator KG = KeyGenerator.getInstance("DES");

        // generate a DES key
        SecretKey mykey = KG.generateKey();

        // initialize the Cipher object to encrypt mode
        jcecipher.init(Cipher.ENCRYPT_MODE, mykey);

        // accessing the input file
        FileInputStream fis = new FileInputStream(args[0]);
        BufferedInputStream bis =new BufferedInputStream(fis);
        int len = bis.available();
        byte[] buff = new byte[len];
        byte[] encText = new byte[len];

        // update the cipher with the data to be encrypted
        while (bis.available() != 0)
        {
          len = bis.read(buff);
          int bytecount = jcecipher.update(buff, 0, len, encText);
        }
        bis.close();
        fis.close();
        jcecipher.doFinal();

        // write the output file containing the encrypted data
        FileOutputStream encfile = new FileOutputStream(args[1]);
        encfile.write(encText);
        encfile.close();

        // write the encoded key to a file
        FileOutputStream keyfile = new FileOutputStream(args[2]);
        keyfile.write(mykey.getEncoded());

        String s1;
        s1 = mykey.getFormat();
        keyfile.close();
        }
        catch (Exception e)
        {
          System.out.println("Caught Exception: " + e);
        }
  }
}
```

Listing 1.

CONTINUED ON PAGE 22

CONTINUED FROM PAGE 21

Next, Bob launches it by passing the names of the following files on the command line:

- The input file that he wants to encrypt
- The output file containing the encrypted data
- The output file containing the encoded key

This is the full command launched by Bob:

```
java Encrypt1 JavaTeam.txt bob.enc bob.key
```

The program runs successfully and two files are generated: the file bob.enc containing the encrypted data and the file bob.key containing the encoded key.

Alice's program

Alice receives the two files bob.enc and bob.key and wants to decrypt the encrypted data contained in bob.enc using the key contained in bob.key. For this reason, she writes a program that retrieves the key from the bob.key file, uses the key to initialize a Cipher object and uses the Cipher object to decrypt the message contained in the file bob.enc. The decrypted message is stored in a file called bob.dec.

Alice's program, called Decrypt1.java, is shown in ***Listing 2***.

Alice compiles the program above with javac.

```
java Decrypt1.java
```

Then she launches the program by passing the following file names on the command line:

- The encrypted file to be decrypted
- The file containing Bob's key
- The output file where the decrypted data must be saved

This is the full command launched by Alice:

```
java Decrypt1 bob.enc bob.key bob.dec
```

The program executes successfully and as a result it produces the bob.dec file, containing the decrypted text. On opening bob.dec with a text editor, it shows the same contents as the file JavaTeam.txt. Using Bob's key, Alice has successfully decrypted the message sent by Bob. This example showed the encrypted contents and secret key being accessible from the same location. Typically, the secret key is not sent with the encrypted contents. Instead, Alice and Bob would exchange the key through some secure means or Alice and Bob would use a key agreement protocol.

Conclusion

The JCE includes classes that extend the Java Cryptography Architecture and allows developers to design and implement advanced cryptography techniques. Developers can choose from a variety of key types, including secret keys, key pairs and keys generated through key agreement protocols.

For more information on Java Security, consult the JavaSoft Web site at http://java.sun.com/products/jdk/1.2/docs/guide/security/index.html.

Anthony Nadalin *is the lead architect for the IBM Java Security project. As the senior architect, he has responsibility for infrastructure design and development across IBM. He serves as the primary security liaison to JavaSoft for security design and development collaboration. You can contact him at drsecure@us.ibm.com.*

Theodore Shrader *is a feature lead in the IBM Java Security project. He has written numerous patents and articles dealing with Internet and Java development, distributed computing, object-oriented design and database architecture and programming. He also is a co-author of an operating systems programming guide published by John Wiley and Sons. You can contact him at tshrader@us.ibm.com.*

Bruce Rich *is the team lead of the IBM Java Security project. He has been involved in software for 21 years, first in operating systems development, then in secure distributed file systems, more recently in secure Web server applications and Java. He has filed a number of patents and contributed to a book on distributed computing. You can contact him at rbruce@us.ibm.com.*

```
import java.io.*;
import java.security.*;
import javax.crypto.*;
import javax.crypto.spec.*;

class Decrypt1
{
  public static void main(String args[])
{
  if (args.length != 3)
    System.out.println("Usage: java Decrypt1 inputEncryptedFile keyFile
        outputFile");
  else
    try
    {
      // get the key to decrypt
      FileInputStream kfis = new FileInputStream(args[1]);
      byte[] encKey = new byte[kfis.available()];
      kfis.read(encKey);
      kfis.close();
      SecretKeyFactory mykeyfac1=SecretKeyFactory.getInstance("DES");
      DESKeySpec dk = new DESKeySpec(encKey);
      SecretKey mykey1 = mykeyfac1.generateSecret(dk);

      // generate a Cipher object
      Cipher jceCipher = Cipher.getInstance("DES/ECB/NoPadding");

      // initialize the Cipher object to decrypt mode
      jceCipher.init(Cipher.DECRYPT_MODE, mykey1);

      // access the file to be decrypted
      FileInputStream data = new FileInputStream(args[0]);
      BufferedInputStream bis = new BufferedInputStream(data);

      int len1 = bis.available();

      byte[] encText1 = new byte[len1];
      byte[] buff = new byte[len1];

      // update the cipher with the data to be decrypted
      while (bis.available() != 0)
      {
        len1 = bis.read(buff);
        int countJlt = jceCipher.update(buff, 0, len1, encText1);
      }

      bis.close();
      data.close();
      ceCipher.doFinal();

      // write the output file containing the decrypted data
      FileOutputStream encfile = new FileOutputStream(args[2]);
      encfile.write(encText1);
      encfile.close();

    }
    catch (Exception e)
    {
      System.out.println("Caught Exception: " + e);
    }
  }
}
```

Listing 2.

Sash Weblications for Windows:
An overview for programmers

Edited by Sean Martin

Editor's note: *This article is intended for programmers who would like an answer to the question: Should I seriously consider Sash for my next programming project? This article is neither definitive nor comprehensive for any aspect of Sash, including technical, business case or philosophy. Other documents exist describing those aspects as does software that you can try. (Refer to the IBM Alphaworks site at http://sash.alphaworks.ibm.com for more information.)* But should you bother to delve into those more time-consuming resources? *The author is a programmer who needed an answer to that question. Read on for his conclusions.*

Sash description

Sash is a programming environment and delivery platform for weblications. A weblication is a program that is:

- Easy to develop, even for people who are familiar only with Web page creation.
- Small, but leverages existing large components and operating system services.
- Easy to install and distribute updates.
- Easy to integrate into the native desktop/work-area seamlessly.
- An encapsulation of operating system functionality, which provides ease of use and higher-level access to features that previously only Windows programming experts could understand well enough to employ. Examples of such features include:
  - Win32
  - COM programming
  - C++
  - Obscure and complex GUI API's
  - Creating installation or e-installation programs

Note: *Sash also is being developed for other platforms including Windows CE, Palm Pilot and Linux.*

Sash contains SashScript, an extension to JavaScript that permits the program to run in non-browser contexts (known as "Locations" such as the Windows desktop tool bars or in the Start toolbar) as well as being able to look like a regular Windows application.

The platform

A Sash weblication currently can run on a machine that is running Windows 95▲, Windows 98▲, Windows NT▲ and Windows 2000▲. Work to support other operating systems is underway. It requires the Microsoft▲ WebBrowser control version 4.71.1712. An easy way to do this is to install Microsoft Internet Explorer▲ 4.0 or above. Most systems shipped in the last two years already have all the pre-requisite software.

The client must have the Sash Weblication Manager installed on it. This software supports the SashScript interpreter and provides functionality common to many weblications. Using the Weblication Manager's capabilities, mini-browser windows are created and embedded in the familiar controls (interface widgets, the start bar, the icon tray, etc.) of the Microsoft desktop. In each of these "Locations," a weblication creates and manipulates one or more of these mini-browsers and its containing control.

The Weblication Manager is about 1.6 MB of DLL and EXE files. This engine can be easily installed directly from the Web. It must be downloaded only once and is done so automatically with the first weblication that a user installs. Weblications are essentially Web pages that have behavior coded in SashScript and are typically several to tens of kilobytes.

The Weblication Manager provides:

- Integration of weblications with a Microsoft Windows desktop by simplifying and abstracting the interfaces.
- Support for inter-weblication communication.
- Simple methods for securely reading and writing of the Windows registry and file system and access system services.
- Secure network communications, such as getting Web page contents and manipulating their Document Object Models (DOM).
- Creation of new types of windows -- including irregular shapes and sprite animations.
- Automatic updating of the engine itself as well as installed weblications.
- A caching system to allow fast execution and disconnected use of weblications.

Sash permits rapid development by enabling the programmer to leverage existing complex, hard-to-write components that already are on the user's system.

The strategy

More than 90 percent of personal computers in today's business environments are running Win32 operating systems. Sash supports Win32. It enables much simpler programming and tighter integration with the Win32 platforms than would otherwise be possible using only a browser. Sash permits rapid development by enabling the programmer to leverage existing complex, hard-to-write components that already are on the user's system. Sash weblications strive to free the user interface from the confines of traditional windows and browsers by supporting the development of user interfaces that integrate into the desktop while still utilizing browser technology. Miniature, unobtrusive user interface components can be tucked into the borders of the desktop, either permanently visible or "show on mouse-over" just as the Windows Start toolbar is commonly configured to do.

By providing easy access to Windows common UI components, weblications tend to adopt a similar look and feel to the Windows UI, making the user experience uniform and achieving tight visual integration with the Windows desktop. Weblications can look like Web pages in browsers or like applications that have been written in full programming environments such as C++. Weblications can take

CONTINUED ON PAGE 24

CONTINUED FROM PAGE 23

advantage of style sheets just like ordinary Web pages.

A weblication may trigger an event in another weblication by writing a message to a channel that the target weblication is listening to. These messages are only strings; other data structures cannot be sent or shared. However, because SashScript is a superset of JavaScript it has eval, so that a message may contain source code that is evaluated in the target weblication.

Programming with Sash

A Sash weblication is one or more Web pages that contain HTML and SashScript. SashScript is JavaScript augmented with a few hundred additional Sash functions, properties and events for:

- Access to Windows guts such as the registry, file system, custom dialogs, networking and if necessary any API function.
- Inter-weblication communications.
- Control of the Windows desktop environment surrounding a weblication.

For instance, a weblication embedded in the Windows Start toolbar can react to resizing events from the operating system (such as when the user stretches the Start bar).

In general, weblications cannot be browsed in an ordinary browser if they utilize Sash's JavaScript extensions because most of the Sash objects will not be available. However certain Sash objects in the "Common Sash API" set, which were deemed to have no security risks associated with them, and a couple of objects that can manipulate Internet Explorer can be accessed in Web pages that are specially designed to be browsed with Internet Explorer on a machine that has the Sash Weblication Manager installed. Work is underway to provide a secure way of giving Web pages the same security access context that a weblication has.

A weblication is made up of one or more Actions. Each Action lives in a Location that determines where on the desktop the Action will appear and which environment it will run in. The available Locations are:

- ***Desktop Band.*** A pane in the Windows Start Button bar.
- ***Explorer Bars.*** The vertical Explorer bar is situated in the left pane in theWindows/Internet Explorer window. Usually a list of directory folders appears in this position. The vertical Explorer bar location can be a substitute for the Explorer Folders view. The horizontal Explorer bar is also a location in a Windows/Internet Explorer window usually splitting the right pane in two. Both are enabled when the user selects the menu options "Views" and then the submenu option "Explorer Bar."
- ***Desktop Toolbar.*** The pane found on the Windows desktop that resides along the borders of the screen. This pane is sometimes hidden until you move the mouse. The pane can be dragged from one edge of the screen to another. It is a custom version of the Start bar. It is invoked by double-clicking on the .dtb file in the Windows Explorer.
- ***Property Sheet.*** You can embed Actions in property sheets, which are usually tabbed dialogs. Property sheets open when you right-click a file and select the properties context menu item.
- ***Screen Saver.*** An Action can be the screen saver on your computer that appears after your computer has been idle for several minutes.
- ***Windows Explorer Shell Namespace.*** The namespace provides ownership of a Folder tree under the Windows Folder hierarchy (such as My Computer and Network Neighborhood). It also provides control of the right-hand Explorer pane, when the "owned" folder hierarchy has focus.
- ***WinDHTML.*** This Location creates a stand-alone application. Its main window can be invoked by double-clicking on the associated .dex file in the Windows Explorer.

Building weblications

The Weblication Development Kit [WDK] contains tools to help develop weblications including:

- Searchable API reference documentation [in MS HTML help format].
- Example code.
- The Weblication Maker for creating weblications -- this includes a wizard to get you started quickly.
- The Weblication Publisher for preparing your weblication for distribution.
- Weblication testing tools.
- Debugging tools.
- Templates for development environments.

To develop a weblication, you first use the Weblication Maker wizard to create the weblication and its associated Actions. Then, you write files containing HTML and SashScript to supply the behavior for each Action. The HTML and SashScript may be created in any text editor, although Microsoft's Visual Interdev 6.0 is currently the easiest environment to use because Sash augments Visual Studio's "Intellisense" function with the Sash objects and properties as well as supports its step through debugger for all weblications.

You then use tools in the WDK to test and debug your weblication.

When the weblication is complete and tested, you package it for distribution by running the Sash Weblication Publisher. This prompts you to:

- Give a human-readable name for the weblication.
- Give a textual description of the weblication.
- Connect the Action HTML files to the appropriate desktop Locations where they will implement the weblication behavior.
- Give a graphic file to use as a "logo" or a Web page for use during the weblication installation.
- Indicate additional source files [html, gif, cab, etc.] and where they should be installed.
- Indicate which files should be placed in the end-user's cache.
- Indicate the level of system access required by the weblication. These will include things like reading and writing to the user disk that are often considered to be security violations.
- Indicate licensing information.
- Indicate instructions for use.
- Indicate what to do in the event of failed installation.
- Digitally sign the weblication for guaranteeing code and author validity and identity.

When you have specified the meta-data for your weblication, the Weblication Publisher generates a unique identifier (a GUID) for the weblication and performs code-signing. Based on the above information, the Weblication Publisher produces a Weblication Definition Format (WDF) file in XML syntax. It also might compress the files into a single/multiple archive file. Finally, the Weblication Publisher creates a "weblication home page" HTML file from a template (which may be altered). The developer then uploads the finished weblication to a normal Web server using the Weblication Publisher's "upload" facility.

Although the weblication code may be entirely local to the end-user's machine, server-only or server-client code storage are normal, too.

The Sash end-user experience

To install a weblication, the user browses the newly created home page of the weblication and clicks on the install button. The user will be informed about the digital signing, system requirements and licensing/payment characteristics of the weblication as defined by the developer. If the user accepts these terms, the weblication will be installed in just a few seconds and is immediately usable without rebooting. If the subscription terms require immediate payment, a normal Web commerce server is employed.

If the Weblication Manager has not yet been installed on the user's machine, it is automatically installed. If the Weblication Manager is installed already but is an obsolete version, then it is updated to the current level.

The user can now operate the weblication in the same manner as any other Windows application. Depending on the developer's specification, the weblication may be updated automatically at any time. The user can use the control panel to remove the weblication just like any other Windows program.

Limitations in Sash

Sash is brand new and is not yet extensively documented or tested. If you choose to use Sash, keep the following limitations in mind:

- Sash doesn't yet address server-side processing.
- Currently only the Windows platforms are supported.

Conclusion

Sash is a programming environment and delivery platform for weblications – applications that are easy to program even by programmers who have limited experience. If you are interested in using Sash, keep the following in mind:

Don't consider Sash if:

- Your application requires a large amount of non-user interface code that is computationally intensive (although you can use Java and native controls with and in Sash weblications to overcome this limitation).
- The downloading and installation of a 1.6 MB runtime engine is too much of a burden for the end users.

Do consider Sash if:

- Your application doesn't require a lot of complex "business logic" and if you:
- Want to use your Web technology tools and skills to build Windows programs
- Want Desktop and desktop application integration (including matching look and feel)
- Want Integration with other weblications
- Are short on "hard-core" programmer time or talent
- Need easy delivery and updating to end-user machines
- Want "instant on" applications over the network.

Sean Martin *works in IBM's Advanced Internet Technology Group in Cambridge, MA. He is one of the lead architects and implementers of IBM's Sash Weblications for Windows technology (http://sash.alphaworks.ibm.com/). In a previous life he architected and built much of the technology behind IBM's high volume Web sites including the Atlanta Olympics, Wimbledon, the US Open and the Kasparov Chess matches. He was the inventor of IBM's prototype Web Object Manager (WOM) technology and has written various patents. Now days he spends all his time developing and evangelizing Sash. He can be contact at sean@gemini.ibm.com*

Understanding cryptographic **messages in e-business**

by Theodore Shrader, Anthony Nadalin and Bruce Rich

This article takes the building blocks of certificates and signatures – discussed in "Signing and verifying with certificates in Java," published in the January 2000 issue of the Developer Connection Technical Magazine – and demonstrates how they are used in today's e-business transactions. In particular, this article introduces the Public Key Cryptography Standards (PKCS) and the use of the signed data in Secure/Multipurpose Internet Mail Extensions (S/MIME) transactions.

Certificates are a unique and standard way in which users and entities can publicly represent themselves in electronic commerce. When users have certificates, they have placed information about themselves, such as their name and e-mail address, in the public domain with a way for that information to be verified or revoked. A certificate also can contain the user's or entity's public key, and this key plays an integral role in the signature process.

The signature process utilizes the user's public key and associated private key to provide for the verification of data. A key pair algorithm uses a random value or other seed to generate a public and private key pair. A user can sign data, such as a message, by running the data through a signature algorithm that also takes the associated private key. The recipient of the data and signature can verify the signature value by running the signature, data and the user's public key through the verification algorithm, which is the counterpart to the signature algorithm. If the algorithm returns true for verification, the recipient knows that the data originated from the sender and that it was not modified in transit.

Public-Key Cryptography Standards (PKCS)

RSA Security Inc. and a consortium of companies developed the first pieces of PKCS in 1991. The set of PKCS standards has expanded and matured to encompass everything from defining encryption techniques (PKCS #1 and #5) to the use of smartcards and electronic tokens (PKCS #11 and #15). The standards include PKCS #7, which describes how signed and encrypted data should be presented, and PKCS #8, which defines the format for private keys, including encrypted private keys. PKCS #9 defines a set of attributes used by many of the PKCS standards. When users request a certificate from a Certificate Authority (CA), they send their public key in a PKCS #10 object to the CA and, once approved, the CA issues a certificate that is wrapped in a PKCS #7 object.

PKCS #12 defines a format for packaging various objects together in a single file. For example, when users export a certificate from a Web browser's certificate database, the Web browser typically creates a password-protected PKCS #12 file containing the exported certificate and accompanying private key. PKCS #12 files are meant to be interchangeable, allowing them to be imported into another Web browser or system certificate database. The full set of PKCS standards is available for review at http://www.rsasecurity.com/rsalabs/pkcs/.

Most Internet and intranet transactions are conducted using one or more of the PKCS standards. A good user interface shields users from needing to know that PKCS standards and objects are being used under the covers, but be assured, they play a key roll in ensuring that transactions can be verified and kept secret. In particular, most e-business transactions exploit objects defined by PKCS #7.

PKCS #7: Cryptographic message syntax standard

The PKCS #7 standard includes a host of widely used objects. The most popular objects include EnvelopedData and SignedData. The EnvelopedData object allows senders to encrypt data with a secret key. (Typically, the application that creates the EnvelopedData object automatically creates the secret key so that the caller does not need to generate it.) The secret key also is encrypted for each recipient, using the recipient's public key. Once the EnvelopedData object arrives, each recipient can use the private key to decrypt the secret key, which in turn can be used to decrypt the data. Popular encryption algorithms include RC2, DES and Triple DES.

Mastery of these underlying standards help customers focus on their business objectives without having to know the infrastructure details that keep their transactions safe and secure.

The SignedData object allows senders to package and sign data and recipients to verify the signed data. This object contains a number of attributes and subobjects. The top layer is composed of the digest algorithm, encapsulated contents, a set of certificates and a set of Certificate Revocation Lists (CRLs). Applications can use CRLs to determine that the certificates are still valid and have not been revoked by the issuing CA. A set of SignerInfo objects forms the second layer. Each SignerInfo object includes the issuer and serial number corresponding to a certificate in the parent SignedData certificate set, the digest and signature algorithms, signed and unsigned attributes, and the signature value itself. Popular digest and signature algorithms include SHA1withDSA and MD5withRSA.

Note that not all fields are required. For example, CRLs are rarely packaged with SignedData objects. Additionally, the SignedData object is structured to allow more than one user or entity to sign the data. Each signer is represented by a SignerInfo object. No matter how many users sign the data, the SignedData object does not replicate the data to be signed. It is enclosed once at the top layer in the encapsulated contents.

Different types of signed data

The fact that not all fields are required means that applications can use the SignedData object in a variety of forms. The first form a SignedData object can take packages the signed contents and at least one signature. When populated with contents and one or more SignerInfo objects, recipients of

the SignedData object can verify that the content was signed by the certificate corresponding to the SignerInfo object.

Frequently, senders construct the SignedData object without contents and wind up sending the contents along with the separate SignedData object to recipients. This second form is known as a signature-only SignedData object. The signature-only object contains the signature value and information about the signer to allow recipients to take these values along with the detached message and verify the signature normally. Recipients cannot use the SignedData object itself to verify the signature on each SignerInfo object since the verification algorithm requires the detached contents to be included as part of its parameters.

The third most popular form of the SignedData object is known as certificate-only. Earlier, we discussed how users could send a PKCS #10 message to a CA to request a certificate. In response, the CA can send back a certificate-only SignedData object. This object only contains a certificate; it does not contain any contents or SignerInfo objects, since there were no contents to sign.

The PKCS #7 standard allows objects to be wrapped within other objects. For example, a user can seal a message by first signing the message by creating a SignedData object. Next, the user can encrypt the SignedData object by creating an EnvelopedData object that takes the SignedData object as its contents to encrypt. To unseal, recipients would decrypt the sealed object to gain access to the SignedData and verify the signature for each of the SignerInfo objects contained in the SignedData object before extracting the message enclosed within.

The PKCS #7 standard has continued to evolve. The most recent version of the standard is reflected in RFC 2630, located at http://www.imc.org/rfc2630. This version upgrades the objects and attributes to support additional functionality.

Sending signed data in e-business

Let's say that Ben wants to use his e-mail application to send a signed message to his business colleague, Patrick, who lives near Antietam Creek. All Ben needs to do is compose the message and select the signed message checkbox before pressing the send button. When Patrick receives the message, his e-mail application displays the message along with an icon indicating that the message was signed and verified. Patrick can rest assured that Ben sent the message and that the contents were not tampered with in transit.

Sending signed data is a simple process for the user, but this process is more complicated under the graphical interface. Developers of e-mail programs and any application that sends and receives signed data needs to be involved in the intricacies of the signing and verification process.

Before delving into the details of the transaction, let's first examine the use of S/MIME. To send secure messages between parties, developers have created the S/MIME standard, which extends the MIME standard. The S/MIME standard builds upon the PKCS standards to allow applications to send secure data through such public mail protocols as the Simple Mail Transfer Protocol (SMTP). Today's applications implement version 2 of the S/MIME specifications. Work on the version 3 specifications currently is underway.

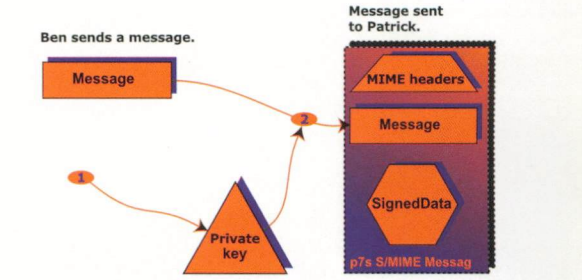
S/MIME Version 2 builds upon the PKCS #7 and #10 objects. The MIME file extensions convey the types of supported objects:

- p7m – EnvelopedData or SignedData with contents
- p7s – SignedData with signature(s) only
- p7c – SignedData with certificate(s) only
- p10 – CertificationRequest object

S/MIME further wraps the SignedData and EnvelopedData objects in a PKCS #7 ContentInfo object that consists of the content type and content fields. The ContentInfo object helps senders package the different PKCS #7 objects in a common object. The content type field helps recipients to identify the transmitted object and use the content field to decode it.

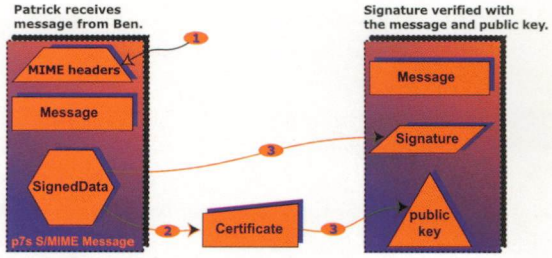
Figure 1 shows how an application, such as an e-mail program, signs and sends Ben's message to Patrick. Once Ben composes his message and presses the send button, the application extracts Ben's private key from the application's keystore or certificate database. The application feeds the private key and message into the signing algorithm, such as MD5withRSA, and generates a signature. This signature, along with other information, is bundled into a signature-only SignedData object. Before sending the message, the application constructs a p7s S/MIME message with MIME header information, original message and SignedData object. MIME header information includes such information as the content transfer encoding, which is base64 for PKCS objects. Finally, the application sends the S/MIME message to Patrick via SMTP.

Figure 2 shows how Patrick's application receives and verifies Ben's message. The application first verifies the MIME headers in the S/MIME message. If the headers indicate a pkcs7-signature type, the application extracts the certificate corresponding to each SignerInfo from the SignedData object and verifies it by tracing the certificate chain back to a trusted root CA. If the certificate is verified, the application extracts the public key from the certificate and extracts the signature(s) from the SignerInfo objects in the SignedData object. The application verifies each signature with the message and corresponding public key, returning true or displaying the signed and verified graphic if the verification was successful.



1. Application extracts Ben's private key from the certificate or keystore database.
2. Application signs the message with Ben's private key, creates a p7s multipart S/MIME message, and sends it to Patrick.

Figure 1.Sending a Signed Message



1. Application verifies the MIME headers.
2. Application extracts and verifies the certificate.
3. Application extracts the signature and verifies it with the message and the public key from the certificate.

Figure 2. Receiving and Verifying a Signed Message

Conclusion

This article discussed the popular cryptographic message standards, PKCS and S/MIME, that are in use today. Using PKCS objects to sign data is not limited to e-mail. Applications can sign and encrypt various formats of binary data in all types of e-business transactions, particularly those that travel across public networks. A developer's mastery of these underlying standards helps customers focus on their business objectives without having to know the infrastructure details that keep their transactions safe and secure.

For additional information:

- On PKCS standards, visit http://www.rsasecurity.com/rsalabs/pkcs/.
- On the S/MIME working group, consult http://www.imc.org/ietf-smime/.

All that JAAS: An overview of the Java Authentication *and Authorization Services*

by Bruce Rich, Anthony Nadalin and Theodore Shrader

On April 4, 1999, JavaSoft published the specification for a new standard extension at http://java.sun.com/security/jaas. The acronym for the extension is JAAS (pronounced "jazz"), which stands for "Java Authentication and Authorization Services." This article presents an overview of JAAS. What it is; why it was created, and some of the rationale behind the current design.

JAAS extends the Java 2 security model

The current Java 2 security model provides fine-grained, policy-based access control for both applets and applications. The permission model takes into account the physical origin (the directory or URL) of the classes that currently are active, as well as their logical origin, the identity of the organization that produced the classes, as proved by digital signature. This model serves well the browsers that first popularized Java, as it deals effectively with the issues of mobile code.

JAAS augments the current Java 2 runtime to add an awareness of the user trying to run the applet or application and augments the Java 2 security model to allow both the specification of permissions that take into account a user's identity and to enforce these permissions at runtime. The first augmentation is what would be referred to as "Authentication." The latter two additions would be referred to as "Authorization."

JAAS extends Java's reach

Java already enjoys success on the desktop and in the browser. For Java to capture the enterprise backend environment, it needs to interact with secure operating systems, resource managers and applications that today are concerned about the identity of the person or computer attempting to execute the code. JAAS allows Java to interact with underlying security applications, learn what the current identities are, possibly modulate these identities (for example, "log in"), reflect these identities to the Java runtime and enforce access controls specified in terms of these identities.

Although it is expected that JAAS capabilities will be rolled into the Java base, it is important to get feedback on what the right set of capabilities should be, and it is much easier to get feedback and make changes if JAAS is first available as an extension. When interfaces stabilize, JAAS functionality then could be incorporated in the Java base.

The final release of JAAS 1.0 should be available sometime in the first quarter of year 2000. It will be available only from Sun Microsystems on Java 2 SDK, Standard Edition, V 1.3 and Java 2 Runtime Environment, V 1.3. IBM will provide JAAS 1.0 for IBM platforms and Linux with the IBM Developer Kit, Java Technology Edition, V 1.3, as well as a slightly-modified version of JAAS for IBM platforms with the IBM Developer Kit, Java Technology Edition, V 1.2.2.

JAAS goals

JAAS is designed with several goals in mind:

1. JAAS allows "simple," pluggable authentication, which means that the authentication interfaces are designed to hide as much complexity as possible. "Pluggable authentication" means that the interfaces are abstract enough that alternate authentication mechanisms should be able to be substituted without security-casual applications needing to know or care.
2. JAAS allows "policy-based authentication," which means that security-casual applications need not concern themselves with the exact authentication mechanisms currently in use. The default login configuration mechanism for JAAS is a configuration file, so that applications need neither know nor care what authentication is happening.
3. JAAS allows "stackable authentication," which means that multiple authentication mechanisms might need to be successful before an authorized context is established. For example, to run the "testDB2" application, a Kerberos login and a DB2 login are required.
4. JAAS allows user-based permissions to be a "simple extension" of the current permission model, which means that the current Java 2 permission model is preserved and that the additional capability is provided in an extensible, easily understood way.

JAAS allows simple, pluggable, policy-based and stackable authentication, and allows user-based permissions be a simple extension of the current permission model in an easily understood way.

Fundamental abstractions in JAAS

Now that the overview-level features and goals of JAAS have been covered, it's time to "open the hood" and examine JAAS in somewhat greater detail. JAAS is contained in several packages rooted at javax.security.auth, with some implementation classes in vendor-specific packages.

One of the first issues that the JAAS designers wrestled with was how to represent the "user," "machine" or the authenticated identity with which we wish to associate permissions. Earlier versions of Java moved in the direction of using java.security.Identity, which is a class (now deprecated) that said that users had a name and java.security.Certificates with which to prove identity. Identity's biggest drawback was its insistence that Certificates were an essential part of an authenticated identity. Since Certificates are obviously not part of all authentication mechanisms, Identity was rejected by the JAAS designers on the grounds of insufficient generality. Another strong contender was java.security.Principal, which is a simple interface whose main method is simply a getName() method that returns a java.lang.String. The

JAAS designers opted to go with java.security.Principal, because it seemed that all authentication mechanisms (to be usable) represent users as having a printable name.

Having decided that Principal would be a key abstraction, the next issue was how to aggregate them to represent an authenticated context. Since one of the JAAS goals was to allow stackable authentication, it was obvious that one could easily end up with multiple Principals in one authenticated context. So, without much discussion, javax.security.auth.Subject was chosen to represent this collection of Principals.

With somewhat more controversy, the JAAS designers concluded that Principals may have some sort of proof of identity that they need to be able to provide at a moment's notice, and these proofs of identity may include sensitive information, so a set of public credentials and a set of private credentials were also added to Subject. Since the content of a credential may vary widely across authentication mechanisms, from a simple password to a fingerprint (to infinity and beyond!), the type of a credential was simply left as java.lang.Object. Relationships between Principals and credentials, if any, were left as an exercise for the implementor of the particular Principal class (or more likely, the particular LoginModule class). From a JAAS perspective, the only difference between private and public credentials is that a particular javax.security.auth.AuthPermission is required for access to the set of private credentials.

Enabling a Java program for JAAS

An important feature in the JAAS design is the mechanism by which an authenticated context is set up. Given a multitude of situations in which authentication is unnecessary, the JAAS designers did not want to make it part of the normal dispatch path for a class or instance method. Thus it was decided that if a Java program wishes to request whatever authentication has been requested for it, it should construct a javax.security.auth.LoginContext and call its login() method. This annotates a Subject with appropriate authenticated identities. To assume the identity of that Subject, the program calls Subject.doAs(Subject,java.security.PrivilegedAction), which runs the specified PrivilegedAction as that Subject, then returns to the original security state. When the program has no further need of the authenticated identities, it can simply call the LoginContext's logout() method. In keeping with the JAAS goal of simplicity, there are no parameters on either of these calls. There are parameters on the LoginContext constructor, however. The first of these is simply a java.lang.String. This is used as an index into a JAAS configuration file, which locates the appropriate LoginModule(s) to be used to authenticate the current user of the program. The usage of this index is entirely up to the Java program, and could be a mechanism to deliberately select a particular type of authentication to be used. If the program is security-casual, it might simply pass in its own class name as the String and pick up whatever authentication had been configured for it. This would look something like:

```
LoginContext lc = new LoginContext(this.getClass().
    getName());

lc.login();
...
Subject.doAs(lc.getSubject(),somePrivilegedAction
    Instance);
...
lc.logout();
lc = null;
```

The additional constructors for LoginContext deal with some increasingly complex situations and are covered briefly with an example in the next section.

Dealing with all possible authentication mechanisms

Another of the issues that the JAAS designers had to wrestle with was the wide variety of authentication mechanisms and the various arguments that they might take. For example, authentication could be as simple as an account name (or user name) and a password or could require a distinguished name and the ability to prove identity through digital signature or could require a name and a fingerprint, or a retinal scan, or ad infinitum. Since one of the goals of JAAS was to have a pluggable authentication mechanism, the framework methods had to be generic enough to allow all authentication mechanisms to work and simple enough so that complexity need not be a hindrance to authentication mechanism providers. This issue is adroitly handled by having the four authentication methods visible at the javax.security.auth.spi.LoginModule API, login(), commit(), abort(), and logout() take absolutely no parameters, and return nothing (in programming, this is a return type of void). This approach certainly succeeds in not requiring authentication providers to add anything artificial to their current interfaces, but still leaves the issue of how to provide the additional information needed.

The mechanism for providing additional information is to have an initialize() method in each LoginModule, where one may or may not pass in a javax.security.auth.callback.CallbackHandler. These callback routines, if provided, can be used in an implementation- and environment-specific manner to gather additional information to satisfy the needs of the particular LoginModule. Translated, this means that if a LoginModule needs some information to authenticate the user, it can examine the array of Callbacks that it was initialized with to see if it has a mechanism to derive the necessary data. For example, if a userid and password are needed, the LoginModule can examine the Callback array to see if it contains a javax.security.auth.callback.NameCallback and a javax.security.auth.callback.PasswordCallback. If so, it can call them to attempt to get the necessary information.

Developers may wonder "Just how did the Callback array get passed in to the initialize() method?" The answer is that one of the more complicated LoginContext constructors (that were glossed over earlier) takes a CallbackHandler and uses it when connecting to LoginModules.

Authorization

Now that the Authentication framework has been discussed, it is time to move on to a brief overview of the Authorization piece of JAAS. JAAS extends the Java 2 permission model to incorporate the idea of authenticated identities in the permission itself. This extension narrows the scope of the permission being granted to only the Principal specified in the permission. To illustrate, the following entry in a normal Java 2 policy file would grant READ access to the file "foo" to all classes in the codebase "http://gordo.austin.ibm.com/barnone":

```
grant codeBase http://gordo.austin.ibm.com/barnone/* {
    permission java.io.FilePermission "c:\\foo", "read";
    };
```

In a JAAS-extended policy file, the following entry would grant READ access to the file "foo" to all classes in the codebase "http://gordo.austin.ibm.com/barsome," but only if the current Subject contained a principal (in this case, a

CONTINUED ON PAGE 30

CONTINUED FROM PAGE 29

com.ibm.security.auth.NTUserPrincipal) named "bob:"

```
grant codeBase http://gordo.austin.ibm.com/barsome/*,
    principal com.ibm.security.auth.NTUserPrincipal "bob"{
    permission java.io.FilePermission "c:\\foo", "read";
};
```

For a more in-depth discussion of JAAS authentication and authorization, consult the whitepaper entitled "User Authentication and Authorization in the Java Platform" (currently at http://java.sun.com/security/jaas/doc/jaas.html).

Additional information

There are a few other points that need to be made regarding JAAS. First, JAAS is intended to be the underpinning for the Java 2 Extended Edition (J2EE) Enterprise JavaBeans (EJB) security model. Given the importance of EJB in e-business, one may expect to see rapid adoption of JAAS.

Second, in part due to the relationship between JAAS and EJBs, IBM plans to offer a version of JAAS bundled with the IBM Developer Kit, Java Technology Edition, V1.2.2. For the most part, this earlier version of JAAS is compatible with the Kestrel version (V1.3.0). The biggest noticeable difference is that the pre-Kestrel JAAS requires the use of a custom security manager, javax.security.auth.SecurityManager, rather than the system default of java.lang.SecurityManager. For additional details, one should consult the programming information included with the IBM Developer Kit.

Lastly, JAAS is expected to be used as the underpinning for Remote Method Invocation (RMI) security enhancements. For more details on this, please refer to http://java.sun.com/products/jdk/rmi/.

Conclusion

The Java 2 platform security framework is an impressive base for secure computing in Java. JAAS builds on this foundation to integrate more closely with secure operating systems, resource managers and applications on enterprise servers.

Additional reading

For the most recent, up-to-date information on the JAAS API, see http://java.sun.com/products/jaas/ (until early availability is over, JAAS information can be found at http://developer.java.sun.com/developer/earlyAccess/jaas/).

IBM MAINTAINS COMMITMENT TO OPEN STANDARDS AND J2EE TECHNOLOGY

RECENTLY, THERE WAS SIGNIFICANT PRESS COVERAGE OF SUN'S DECISION NOT TO SUBMIT JAVA TO THE EUROPEAN COMPUTER MANUFACTURING ASSOCIATION (ECMA), THE INDUSTRY STANDARDS ORGANIZATION SUN AND OTHER COMPANIES (INCLUDING IBM) AGREED SHOULD MANAGE THE STANDARDIZATION OF JAVA. AS A RESULT, THE ECMA IS CONSIDERING PLANS TO PROCEED WITH THE STANDARDS EFFORT FOR JAVA, A DECISION IBM SUPPORTS AS A COMPANY DEEPLY COMMITTED TO MULTI-PLATFORM, MULTI-VENDOR STANDARDS. ALONG THESE LINES, SUN ALSO HAS ANNOUNCED "BRANDING" OF JAVA 2 ENTERPRISE EDITION (J2EE®) AS A SUN PRODUCT INSTEAD OF AN OPEN STANDARD. ALTHOUGH IBM HAS MADE NO COMMITMENT TO SUN'S RECENTLY ANNOUNCED BRANDING CONCEPT FOR J2EE, THIS IN NO WAY NEGATES IBM'S COMMITMENT TO J2EE TECHNOLOGY. WE CONTINUE TO WORK WITH SUN ON THE DEVELOPMENT OF JAVA AND SUPPORT THE ENTERPRISE JAVA TECHNOLOGY THAT COMPRISES SUN'S J2EE.

EVIDENCE? IBM CONTRIBUTED TO THE DEFINITION OF MORE THAN 80 PERCENT OF THE J2EE APIS, INCLUDING ENTERPRISE JAVABEANS (EJB), JAVASERVER PAGES (JSP), JAVA SERVLET, JAVA INTERFACE DEFINITION LANGUAGE (IDL), JAVA DATABASE CONNECTIVITY API (JDBC), JAVA MESSAGE SERVICE (JMS), JAVA NAMING AND DIRECTORY INTERFACE (JNDI), JAVA TRANSACTION API (JTA), JAVA TRANSACTION SERVICES (JTS) AND RMI-IIOP. IBM LED AND TOOK THE INITIATIVE WITH SUN IN DEFINING EJB, JTS AND RMI-IIOP AND DEVELOPED JOINTLY WITH SUN THE REFERENCE IMPLEMENTATIONS OF JTS AND RMI-IIOP.

MORE PROOF? THE DELIVERY OF J2EE TECHNOLOGY, INCLUDING EJB 1.0, SERVLET 2.1, JSP 1.0, JTS/JTA 1.0, A SUBSET OF JDBC 2.0, JNDI 1.1, RMI-IIOP 1.0, SECURITY 1.0 AND XML IN IBM'S WEBSPHERE APPLICATION SERVER FAMILY (STANDARD EDITION, ADVANCED EDITION, ENTERPRISE EDITION) V3.0 TODAY — WITH PLANS TO DELIVER FULL SUPPORT FOR THE J2EE APIS BY THE END OF 2000 — AND MQSERIES SUPPORT OF JMS BY THE END OF THIS YEAR, IS FURTHER EVIDENCE OF THE COMPANY'S COMMITMENT TO THE IMPORTANCE OF JAVA AND J2EE IN OUR MIDDLEWARE PORTFOLIO.

AND FINALLY, IBM'S APPLICATION FRAMEWORK FOR E-BUSINESS, WHICH ILLUSTRATES IBM'S COMMITMENT TO MULTIPLATFORM, MULTIVENDOR STANDARDS BY OFFERING AN IMPLEMENTATION APPROACH FOR E-BUSINESS AND DEFINING HOW TO DESIGN AND BUILD FLEXIBILITY AND OPENNESS INTO APPLICATIONS, INCORPORATES AND WILL CONTINUE TO PROVIDE SUPPORT FOR THE J2EE APIS AS DEFINED IN THE J2EE STANDARD EXTENSION.

IBM CONTINUES TO EVALUATE THE IMPLICATIONS OF SUN'S J2EE BRANDING STRATEGY, AND ITS IMPACT ON OUR CUSTOMERS, BUSINESS PARTNERS AND THE 3500+ IBM DEVELOPERS WHO ARE COMMITTED TO JAVA. PLEASE BE ASSURED THAT IBM CONTINUES ITS COMMITMENT TO OPEN INDUSTRY STANDARDS AND TO J2EE TECHNOLOGY.

IBM Developer Connection Technical Magazine

DIRECTORY

Order numbers in the following countries:

Austria	0660 8705	Mexico (D.F.)	5 270 5990
Canada	800-561-5293	Mexico (Interior)	01-800-006-3900
Germany	0 130 828041	United States	800-6DEVCON(633-8266)

Latin and South America order numbers:

Argentina	0-800-44-426 92Interno 2987	El Salvador	02-98 5011
Bolivia	02-35 1840	Guatemala	02-31 5859
Brazil	0800-111426 r.1351	Honduras	32-2319
Chile	800 218 218 anexo 6970	Panama	02-639 977
Colombia (Nacional)	9800-17555	Paraguay	444-094
Colombia (Bogota)	616-7555	Peru	349-0040
Costa Rica	223-6222	Uruguay	0-800-A-IBM (0-800-2-426)
Dominican Rep.	566-5161	Venezuela	800-1-5000
Ecuador	(02) 565-090		

Asia/Pacific order numbers:

The Developer Connection can be ordered in Asia/Pacific countries from IBM in Australia (61 is the country code) and Japan. Please ensure that you dial the international access code applicable to your country before the listed phone or fax number.

Australia:	Phone	+61 2-9354 7684
	Fax	+61 2-9354 7766
	E-mail	pwdap@au1.ibm.com
Japan:	Fax	+81 3-5200 6310
	E-mail	os2pid@jp.ibm.com

Europe and other countries not listed:

The Developer Connection can be ordered directly from IBM in Denmark (45 is the country code). Please ensure that you dial the international access code applicable to your country before dialing the appropriate phone or fax number. Operators speaking the following languages are available:

Danish	+45 48 101300	German	+45 48 101000
Dutch	+45 48 101400	Italian	+45 48 101600
English	+45 48 101500	Norwegian	+45 48 101250
French	+45 48 101200	Spanish	+45 48 101100
Finnish	+45 48 101650	Swedish	+45 48 101150
		Fax	+45 48 142207

Note: To subscribe to the Developer Connection in South Africa, contact Claudia Wissler, IBM South Africa, at 27-113029111, ext. 6960.

Electronic Support for the Developer Connection Release 2 Program

Electronic support is provided through the Internet and OS/2 BBS. Obtain technical support or use the forums to exchange messages, ideas, or comments with the Developer Connection team or other subscribers. Note: Refer to the specific product information for the operating system technical support methods.

Internet users may address their questions or comments to devcon@us.ibm.com.The DEVCON CFORUM is on the OS/2 BBS under TalkLink, which is a feature under the IBMLink Commercial Services. For TalkLink access, U.S. customers can call 1-800-426-5465; customers outside of the U.S. should contact their local IBM Marketing Representative. Note: Commercial members of PartnerWorld for Developers (formerly IBM Solution Developer Program) can access the DEVCON CFORUM on the Web from the PartnerWorld for Developers Web page (www.developer.ibm.com) without specifically signing up for IBMLINK.

PartnerWorld for Developers

The Developer Connection is one of a wide range of offerings that are available as part of PartnerWorld for Developers. In one single resource, PartnerWorld for Developers provides technical, business, marketing, and information services that can help you to reduce your development costs, accelerate your development efforts and showcase and sell your products. Members receive, at no charge, a wealth of information about IBM products and technology that can be searched on the PartnerWorld for Developers Web site 24 hours a day, seven days a week, at www.developer.ibm.com/

If you are not a member of PartnerWorld for Developers, why not join now – registration is free! It takes only a few minutes to register on the Internet, and in return, you'll receive a membership ID and password to access our services. Welcome aboard!

PartnerWorld for Developers Hotline:

United States/Canada	1-800-627-8363
Worldwide	1-770-835-9902
Worldwide Fax	1-770-835-9444

Global Solutions Directory:

The Global Solutions Directory is IBM's comprehensive, online source for over 30,000 partner solutions. The guide is a worldwide resource for applications, tools and services. Visit the Guide today at www.software.ibm.com/solutions/isv/

Staff

Publisher	Karen Foley
Managing Editor	Jean Swanson
Assistant Editors	Tracey Marcelo and Loree Eubank
Graphic Design	Stil Point Images, Michael Rainer