Dr.Dobb's
THE WORLD OF SOFTWARE DEVELOPMENT

How Do I Create a Signed Castanet Channel?

Source Code Accompanies This Article. Download It Now.

- javaqa.txt

This month, Cliff addresses centralized systems administration using Marimba's Castanet 2.0 channel (and JavaSoft's HotJavaBean) for distributing secure, signed content to users.

January 01, 1998
URL:http://www.drdobbs.com/jvm/how-do-i-create-a-signed-castanet-channe/184410474

Cliff, vice president of technology at Digital Focus, can be contacted at cliffbdf@ digitalfocus.com. To submit questions, check out the Java Developer FAQ web site at http://www.digitalfocus.com/faq/.

For years, information systems managers and administrators have struggled to maintain software on client machines. The proliferation of applications and the use of computers to support every business task -- combined with the accelerated pace of updated software releases -- has made managing an organization's computing infrastructure a Herculean task. What is needed is centralized administration.

The idea of centralized administration is not new. Software has long been available in server-based or server-administered versions. However, most administration techniques used in the past have been proprietary, and not applicable as general solutions. Push technology, which is represented by a collection of products that automatically distribute arbitrary content to clients in a robust and application-independent way, has the potential to solve the client software administration problem. This month, I'll reexamine Castanet, the Java-centric (but not limited to Java) push technology from Marimba (http://www.marimba.com/), which I previously looked at in "How Can I Create a Push Java Channel" (*DDJ*, May 1997). This time, I'll examine how to create a signed Castanet channel for distributing signed and trustworthy content to users.

Distributing Real Applications

Since its release, Java's primary deployment platform has been browsers. Browsers, however, are best used for what their name suggests -- browsing. If you have a business need for an application, you don't need to browse -- you need an application that installs and deploys to run on demand. Browsers are not designed for this.

Recently released versions of browsers do incorporate installation features. Netscape, for example, has "Netcaster," which is actually Castanet repackaged inside of the Netscape browser. Thus, when you develop a Castanet channel, you are also developing a Netcaster channel. Internet Explorer has a similar technology for content subscription.

browsers and push software provide security features that limit what remotely obtained programs can do. Arguably, Java provides the most secure method for running remote applications -- a Java-enabled browser will not allow a Java applet to access the local disk drive, where sensitive information might be stored. More recent versions of browsers relax this restriction if the applet is digitally signed, providing traceability to the applet's source. Castanet now also allows you to digitally sign a channel and permit a signed channel to have full-featured access to the local system.

For business applications, the best way to view Castanet is as a technology for deploying highly scalable applications, with centralized update and administration capabilities. Using this technology, and its new content signing and SSL transmission features, you can now create full-featured client applications that are trustworthy, secure, and up to date -- regardless of whether or not they traverse the Internet. This is an ideal solution for intranet/extranet requirements, as well as public-access applications via the Internet.

(Castanet technology actually goes beyond this. You can, for example, use the UpdateNow API to embed Castanet into your software to create self-updating products.)

Castanet Overview

Castanet consists of a client called the "tuner" and a server called the "transmitter." Channels are published by an administrator on a transmitter. A workstation that has a tuner installed can subscribe to any number of channels on any number of transmitters. The tuner updates the channel's content, which may consist of Java code, media, and other files, including binary code (signed channels can load native methods supported by DLLs). The tuner presents a user interface for managing channels. The tuner interface is set apart from the interface constructed by any given channel (if the channel even has a user interface). This is unlike a browser, which imposes a user interface frame around any application that runs within it. Channels construct their own frames if and when they need them.

When a channel is published, the administrator specifies default update behavior for the channel. Depending on how the channel is programmed, it may or may not adhere to what the administrator specifies, since the update behavior can be controlled by the actual channel application code. Users also have control over the channel's update behavior. When the tuner determines that a channel needs to be updated, it automatically notifies the transmitter which publishes the channel and requests an update. The details of the protocol are completely hidden from the application code, and may occur even when the channel is not running. When the update process completes, the channel may be notified that there is new content, and it has the opportunity to specify how the content should be installed -- immediately, after a channel restart, and so on.

Castanet 2.0 Features

Release 2.0 of Castanet incorporates the RSA BSafe library for code signing and data encryption. Thus, Castanet now lets channel administrators digitally sign a channel. Administrators can also specify that the channel content be distributed via an encrypted SSL stream, providing a highly secure method of deployment that can easily traverse a public network without the risk of interception or forgery. To develop a channel, you don't have to do anything different than before -- the new functionality does not affect the development process or the channel API. It is all handled through the publish tool and transmitter protocol.

If you don't want to, you do not have to sign your channels or encrypt their downloads. The 2.0 features are optional capabilities. However, the advantages are enormous. If a channel is signed, Castanet allows an application to do all the things a traditional installed application would do, including reading from or writing to any part of the client's disk, and accessing any host on the network. This is because it is presumed that a signed channel can be trusted, because the creator of the channel can be identified via the digital certificate used to sign the channel, and the channel can be proven to have been unaltered since creation. At the present time, channel capabilities are either all or nothing -- if a channel is signed, it can do anything. A future release of Castanet will

To illustrate channel signing, I'll implement a signed channel that does something an unsigned channel cannot -- access arbitrary URLs. To do this, I will use Javasoft's HotJava HTML-rendering Bean.

Applications that Access the Web

Most of today's business applications must be able to access the company's web site. Since many companies now use the Web to publish critical business information, applications must be able to get to this information -- and be able to display it for users.

This generally means operating the application alongside or within a browser; hence, many applications are being developed as web applications, using web protocols (like CGI) or web-deployable application languages (like Java). In this mode, web data in HTML or any other web-enabled format can be displayed side-by-side with the application.

The disadvantage of this practice is that a browser is not a very good application platform. However, if you want to view web data, you are stuck with it -- or are you?

Many individuals and companies have attempted to create HTML-rendering Java components, which can bring the Web to an application. By embedding an HTML renderer in an application, the application can allow users to view company documents published on the company's web site, or even view data retrieved from a database, but rendered via HTML. Further, the application can then easily control the browsing session, and even provide users with a customized selection of URLs to view. The viewing window can be placed anywhere within the Java application, just like any other component.

Creating such a component is a big job. It is not too hard to write an HTML 2.0 renderer. However, writing a full-featured renderer that can display anything a webmaster might throw on a web site is another matter. The HotJavaBean developed by Javasoft is such a component -- it can handle full HTML 3.2, with style sheets, applets, and frames. Since it is the same component used in the HotJava browser, anything HotJava can do, the HotJavaBean can do. It is the perfect answer for an application that primarily needs to access business data, but also access a web site. HotJavaBean is only 600 KB -- practically nothing for this capability set -- and it's very manageable in the context of a Castanet channel.

HotJavaBean shows that by deploying an application via Castanet, you do not have to forego the web capabilities of a browser. Companies are generally afraid of the Internet for business applications -- the security risks are complex. However, they know that today they need web access, but in a controlled way. With Castanet and the HotJavaBean, you can create secure, web-enabled applications that primarily perform a business function, but can access web sites as required by the application -- and no more.

BigCorporateApp and *LittleBrowserWindow*

My demonstration program is a Castanet channel that opens a window on the screen, and allows a user to select from one of a small set of web locations to view. (These locations may be file URLs on a user's own machine, if desired.) Selection is made via a drop-down choice list, hardwired into the program. When the user makes a selection, the program fetches that content and displays it in a web viewing window, implemented with the HotJavaBean component.

I won't go into all the details of the HotJavaBean component, except to show you how to instantiate and do some very basic things with it. It is an interesting component in its own right, and I encourage you to study it.

The HotJavaBean is a JavaBean, which means that it can be added to the component library of a bean-capable development tool to build applications. In most cases, you would create an application using a visual builder tool, and place the HotJavaBean component using that tool. You would then hook its event sources to other

what to do when the user clicks on a hyperlinked URL in the HotJavaBean window -- ignore, allow, or disallow it.

You do not have to use the component as a Bean, however. Since I cannot publish a development environment with this article, I've decided to simply instantiate the component the way you would instantiate any component: I simply create one using the *new* operator, and call its methods directly. I could write event adapters, but that is getting out of the scope of this article.

My main class, *BigCorporateApp*, is a Castanet application that implements the *marimba.channel.Application* interface, and can be published as a Castanet channel (see Listing One). I want my application to come up in a Java Frame. I also want my application to use default Castanet policies for updating itself. The easiest way to accomplish these objectives is to extend the *marimba.channel.ApplicationFrame* class. This is a Java AWT frame that implements *marimba.channel.Application* in a standard way.

Castanet is JDK 1.1 compatible, so you can do anything you would do in a JDK 1.1 application. You are also subject to the same JDK 1.1 event model restrictions. The AWT decides which event model you are using, on a component-by-component basis, depending on whether the component has any listeners registered. Basically, if a component has at least one 1.1-style listener registered, the component is presumed to use the 1.1 event model; otherwise, the 1.0 model is used.

The term "Castanet application" is a bit of a misnomer. In Java parlance, an application often connotes that the program has a main routine. (I use the term to represent a superset of programs that are standalone as well as applets.) However, a channel is actually an object that is instantiated by the tuner, with a preestablished environment. This environment includes a context called an "Application Context," which is the link between the channel program and tuner environment. It also includes a security manager which safeguards the local system. Thus, your channel cannot set its own security manager because the tuner has already set one.

A Castanet tuner calls an application's *start()* method after it constructs and initializes the application object. At this point, the application has an *ApplicationContext*. In the *start()* method for this demo, I first add a handler to handle the window-close event (triggered when the user clicks on the "X" of the frame to close the frame), and then I do two things:

- Obtain the URL that the user last selected during the previous session.
- Set up the GUI components of this application.

The only GUI control is a Choice selector, which lists the allowable URLs that the user can select from. When one of these is selected, the program saves it in a special data directory for the channel, then causes the HotJava browser Bean to fetch the content at that URL and display it. Note that the list of allowable URLs is hard-coded into the program. They could have been downloaded as part of the content of this channel, in which case the program would have to parse the file containing the list and construct the choice control from that. I chose to hard-code it for simplicity here.

I instantiate the HotJava bean and add it to the frame, saving its reference in the variable *littleBrowserWindow*. The window should be sized appropriately for its purpose. I have chosen to make use of the frame's default layout policy, and I placed descriptive text to the right of the browsing window. Imagine if instead this label area was occupied with data retrieved from a database, or other active application data. The browsing function would then be mostly ancillary. It is still an important part of the overall application -- but not so important that it justifies embedding the application itself inside of a browser!

Since users can select on any hyperlinks within the documents displayed within the browser component, webmasters should be careful. Of course, your firewall may restrict outside access. You might have to build proxy specification into the channel if you have a proxy (you can do this by giving end-users a control for setting the proxy properties of the HTML renderer).

Channel content can be transmitted encrypted using SSL. Current implementations of the SSL protocol require that servers, and optionally clients, possess a digital certificate of a type recognized by the server. Marimba has arranged with Verisign to provide its server certificates. When you start the transmitter, it displays a dialog labeled "Enable Secure Connections." On this dialog there is a button "Request Certificate." If you click this, the transmitter tool guides you through the steps of getting a certificate, including key generation and launching your browser to visit Verisign's web site. The dialog also has a space for entering a password for encrypting the private key returned by Verisign, so that someone cannot steal it from your computer.

One problematic item of information you will have to provide when getting a transmitter certificate is your host's domain name. In addition, you need to request the certificate from the same host that will be used to host the transmitter you are requesting the certificate for. If you later move the transmitter to another host, you will need to obtain a new certificate. (There is no technical reason for this -- it is merely a matter of certificate providers colluding to sell more certificates.)

The transmitter certificate lets you run SSL. To sign a channel, you need a signing certificate. This is another kind of certificate you need to obtain from Verisign. Logically, there is no reason why the same certificate could not be used. In fact, Marimba has implemented a test mode, in which you can use your transmitter certificate to sign your channels to see if they work, before getting a separate signing certificate (at a cost of about $400). Again, certificate providers have decided to make server certificates (such as the transmitter certificate) and signing certificates different, by choice, so that you have to buy more of them. Granted, there is some additional information in each kind of certificate, but there is no reason why one certificate could not in theory be used for both purposes.

Publishing the Application

You can publish either the entire channel as a signed channel, or you can sign just part of it. Signing part of it is accomplished by putting the portion to sign in a subdirectory named "signed" and selecting the "Only signed subdirectory" option in the Publish tool's Security tab; see Figure 1.

The reason you might want to publish only part of a channel is because some of the content of a channel might be dynamically generated (portions added by plug-ins, for instance), and this part cannot be signed because the signature needs to be calculated at the moment of publishing. When you publish the channel, you need to decide if the channel will run in the normal restricted mode on the client, according to the default restrictive security policy, or if it will be allowed to run unrestricted. (In the future more granularity may be added to this selection.) Selecting "ALL" chooses the latter, and the channel will then be able to read and write any files on the user's disk, access any network host, and so on -- provided the channel's digital signature can be authenticated.

The source code for the completed browser is available electronically from *DDJ* (see "Resource Center," page 3) and Digital Focus (http://www.digitalfocus .com/ddj/code).

Conclusion

At Digital Focus we have used Castanet to deploy mission-critical applications for clients, accessing CORBA, JDBC, object databases, and various other kinds of middleware. Castanet is the most robust and powerful way to deploy Java applications today, and we view it as a strategic component of the solutions we provide. The new features are a major step in addressing corporate security concerns, and as the Java security model matures, Castanet is expected to follow it. Full-powered applications, remotely administered, automatically deployed, secure, and always up to date -- what more could you ask for?

DDJ

Listing One

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.