In addition to just the software, the Transmitter sends a Properties file. Among other things, this file includes a recommended schedule for channel updates (frequently—less than five minutes—to daily to weekly).

Following the suggested schedule, the Tuner periodically polls the Transmitter for new information. The new information might include news headlines, which are displayed automatically to the user, or new software components, which add to the program's functionality (see Figure 15-4). Each update is performed efficiently and differentially.

15.2 Castanet Channels

So, you have a Tuner, a Transmitter, and a sophisticated update mechanism. But what exactly can you deliver? Castanet supports four main types of channels: HTML channels, applet channels, presentation channels, and application channels.

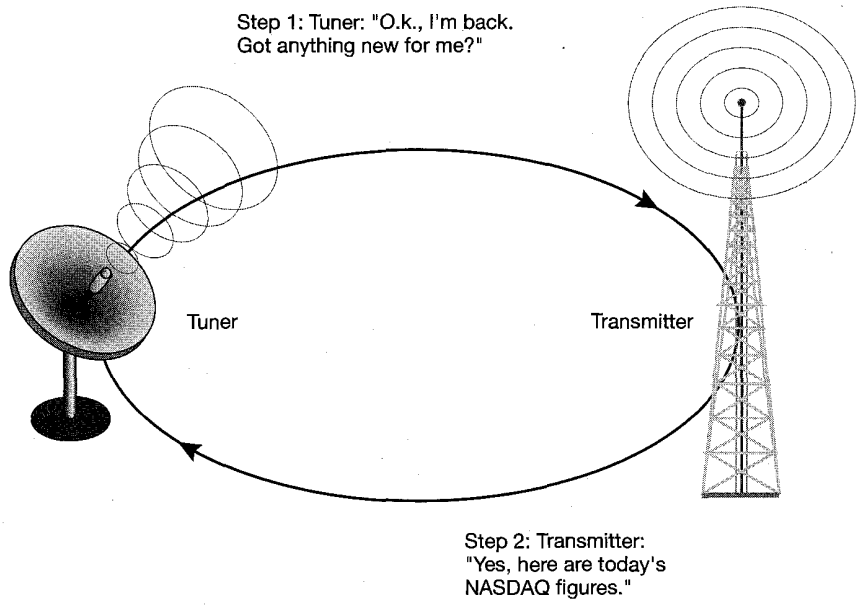


Figure 15-4
Channel updates. The Tuner follows a recommended schedule set by the channel developer and periodically polls the transmitter for new information. If new information is available, the data is differentially downloaded—that is, only files that have changed since the last update are downloaded.

15.2.1 HTML Channels

HTML channels represent an entire collection or directory of HTML pages. When a user first subscribes to the channel, the first page, usually `index.html`, is downloaded and passed to the user's default Web browser. If the user follows any links off the first page, those pages are downloaded.

After a few minutes, the Tuner downloads the entire HTML channel and stores a copy on the user's hard drive. The HTML documents therefore are always available, even when the user is not connected to the Internet. And, because HTML channel updates occur with Castanet's update technology, channel updates are both fast and efficient.

HTML channels are most similar to the Web-crawl capabilities provided by Netscape Netcaster and Microsoft Active Channels. Castanet uses a different approach, however, which enables you to transmit entire directories of HTML documents, circumventing the need to Web crawl any of these documents. Note that HTML channels do not follow the *Channel Definition Format* (CDF) officially endorsed by Microsoft.

15.2.2 Applet Channels

Applet channels are Java applets that are downloaded and updated automatically. In most cases, you can take any existing applet (Marimba actually calls these *legacy* applets!) and, with little or no changes to your source code, publish it as a Castanet channel. Because the applet is stored locally, users don't have to wait for lengthy downloads each time they want to work with your applet.

15.2.3 Presentation Channels

Presentation channels are user interfaces created with Marimba's Bongo software program. So what is Bongo? Bongo is a software application that enables programmers and nonprogrammers alike to easily create *graphical user interfaces* (GUIs) for Java applications and presentations. Developers simply can point-and-click to add a new button, for example. Or

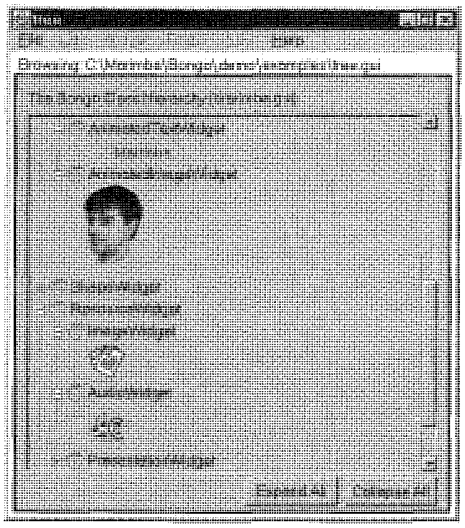


Figure 15-5
Marimba's Bongo software enables programmers and nonprogrammers to create Java GUI interfaces simply by pointing and clicking. Here is just a sampling of some of the widgets provided by Bongo.

they can choose from a large library of widgets, including scrollbars, drop-down menus, and animated text boxes (see Figure 15-5).

As you continue to develop your own Castanet channels, it is highly recommended that you check out Bongo. Many programmers have complained of Java's often complex *Abstract Windowing Toolkit* (AWT) for creating GUIs. In particular, many programmers have griped about Java's often confusing Layout Managers (anyone who has worked with Java's Grid Bag Layout Manager knows exactly what I'm talking about!). So check out Bongo and save yourself lots of programming aggravation! Evaluation copies and tutorials are available at the Marimba Web site at `http://www.marimba.com`.

15.2.4 Application Channels

To get the full effect of all of Castanet's many features, you will want to create an application channel. Application channels are secure Java programs that can receive automatic information updates or actually update themselves.

To ensure security, application channels are restricted to many of the same security restrictions as regular applets. But there is one exception

to these restrictions: Application channels are able to read from and write to one specific directory on the user's hard drive. This capability enables a new generation of Java applications that finally can circumvent the applet security "sandbox" and thereby create persistent data.

Examples of commercially available application channels follow:

CBS SportsLine Score Center	Displays constantly updated sports scores (see Figure 15-6)
COREL Office for Java	Provides a full suite of productivity applications written entirely in Java
go2vision	Displays constantly updated news headlines, sports scores, and stock quotes
HotWired talk.com channel	Provides real-time chat rooms for hundreds of simultaneous users
MapQuest Channel	Provides an interactive atlas and listings of millions of businesses and places of interest throughout the world
Net It Now!	Publishes desktop documents across an intranet without the use of browser plug-ins
Pencil Me In	Enables enterprise-wide calendaring and scheduling
Sesame Street Sticker Channel	From the Children's Television Workshop (see Figure 15-7)
Wall Street Web	Monitors stocks, bonds, and mutual funds
Web Interface for Telescience (WITs)	Program from the Jet Propulsion Laboratory (described earlier in this chapter)

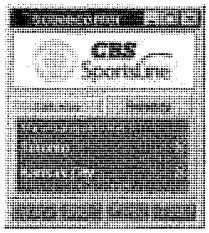


Figure 15-6
The CBS SportsLine Score Center provides a constantly updated list of sports scores from your favorite teams.

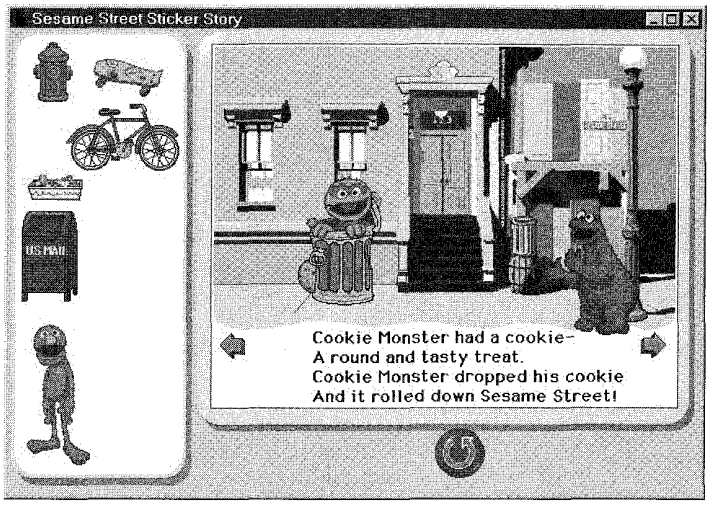


Figure 15-7
The Sesame Street Sticker Channel, from the Children's Television Workshop. Now where is that cookie, anyway?

15.3 Castanet Plug-Ins

Application channels can be customized and personalized for each subscriber through the use of Castanet plug-ins. Each channel can have its own plug-in, and the plug-in is housed on the Castanet Transmitter. You therefore can think of a Castanet plug-in as an extra component of software that plugs into the Transmitter to provide extra functionality, much like browser plug-ins provide added functionality for viewing different types of Web documents.

Each time a Tuner polls the Transmitter for new information, the tuner also sends a log file and profile data (see Figure 15-8). The log file could contain any record of user activity the channel developer wants to track. The application might track the specific advertisements that each user follows, for example, or it might compile a list of the most popular news categories or even store the results of an online survey.

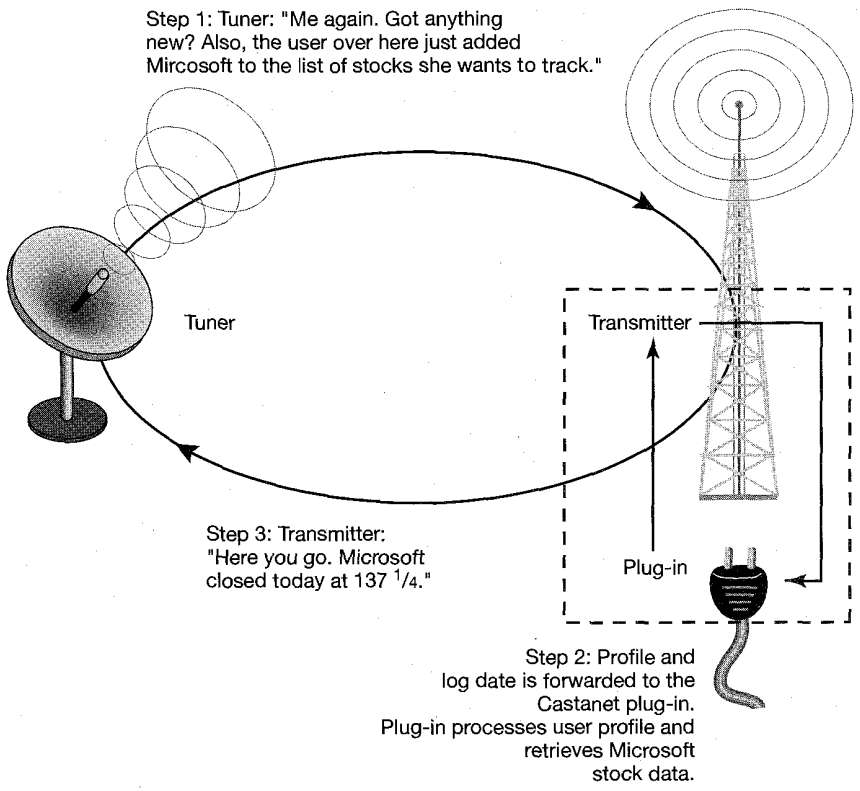


Figure 15-8
Castanet plug-ins. Each time a Tuner requests a channel update, it also sends its log and profile data. Plug-ins represent separate Java or C++ applications that can be used to record user feedback or customize the channel experience.

The profile data includes specific options and preferences set by the user. This might include demographic information, a list of news topics the user is interested in, or perhaps a list of stock symbols the user wants to track.

Plug-ins represent separate Java or C++ applications that can store and process both log and profile data. The plug-in might just save the logging information to the Transmitter hard drive, for example, or it might use the data to send back dynamic results. For an online survey, the results could be compiled automatically and sent back. Or, for advertisements, you could send back more detailed product information or information about similar products.

With profile data, you can personalize and customize both the content and the application. For a financial application, for example, you could send a bar chart program along with historical data for specific companies the user is tracking.

Processing user feedback is key to raising advertising revenue, and customized channels are key to retaining subscribers. Plug-ins give you the maximum flexibility to do both.

15.4 The Castanet Publisher

As a channel developer, you need one additional piece of software: Castanet Publisher. Castanet Publisher enables developers to efficiently copy over their Java applications from their local hard drives to the Castanet Transmitter. Much like Tuner updates, updates to the Transmitter via the Publisher are differential; only files that have changed since the last update are copied over. After channel contents are copied over, they are available immediately to all subscribers (see Figure 15-9).

The Publisher and Transmitter include several levels of security to ensure that only registered developers with security clearance can publish to the Transmitter. As is described in later chapters, you will need to use Castanet Publisher to test and debug your channels.

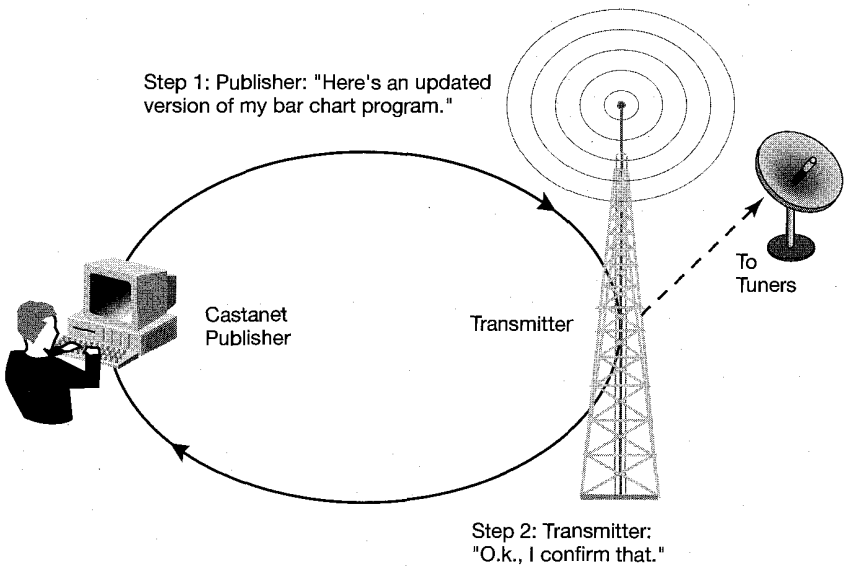


Figure 15-9
Castanet Publisher enables developers to efficiently copy their programs and data to the Castanet Transmitter. This enables many developers to publish to the same channel while ensuring an adequate level of security. After the updated channel is published, it is available immediately to all subscribers.

15.5 Scaling to Millions

With hundreds of Castanet Tuners inside your organization, you need a more efficient method of controlling tuner updates. And, with thousands of potential subscribers across the Internet, you need more than one Transmitter to handle the load. To solve both these problems, Castanet offers two additional software components: the Castanet Proxy Server and the Castanet Repeater. Together, these components can scale to reach thousands and even millions of subscribers.

15.5.1 The Castanet Proxy Server

The Castanet Proxy Server enables companies to efficiently manage Castanet traffic traveling through the corporate firewall. Without a Proxy Server, each Tuner makes requests directly to a Transmitter located somewhere on the Internet. Each Tuner can make the same request, and the updated information is relayed through the corporate firewall any number of times.

Tuner requests are sent to the Proxy Server, and the Proxy Server relays the request to the Transmitter. For each request, the Transmitter processes the channel plug-in to ensure personalization and customization; it returns a list of updated files but does not include the files themselves. If the Proxy Server already has the necessary files cached, they are sent immediately to the Tuner. Otherwise, the Proxy Server downloads and caches the new updated information and then forwards it to the Tuner (see Figure 15-10). This significantly reduces Castanet traffic through the firewall while ensuring the complete customizability of channels. The Proxy Server also maintains a single open connection to the Transmitter to eliminate the overhead associated with each TCP connection.

15.5.2 The Castanet Repeater

Castanet Repeaters enable organizations to distribute and balance requests for Tuner updates among several Transmitters. These Repeaters could be located in the same location as the original Transmitter or geographically dispersed throughout the world.

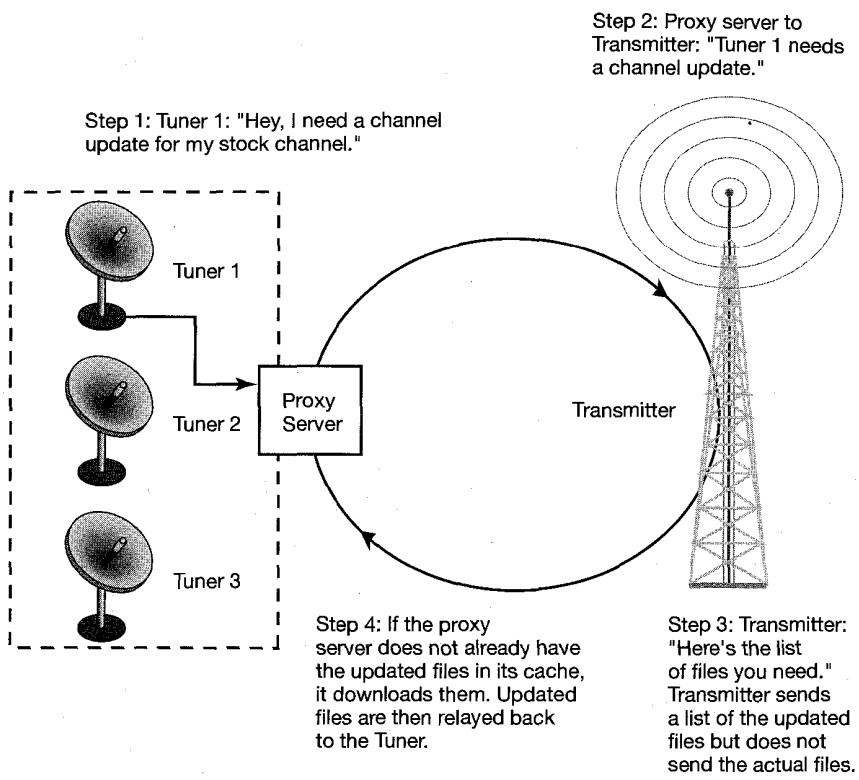


Figure 15-10
The Castanet Proxy Server handles Castanet traffic traveling though the corporate firewall. This makes for quicker updates and more efficient use of network bandwidth.

When a main Transmitter receives a Tuner request, the Transmitter can handle it or forward the request to a Repeater. Forwarding of Transmitter requests can be round-robin or based on time zone, sending the request to the Transmitter geographically closest to the requesting Tuner (see Figure 15-11). If the forwarding Repeater is down, the Tuner returns to the original Transmitter and requests the address of a substitute Repeater. When a developer updates or publishes a new channel, the updated information is propagated to the entire list of repeaters.

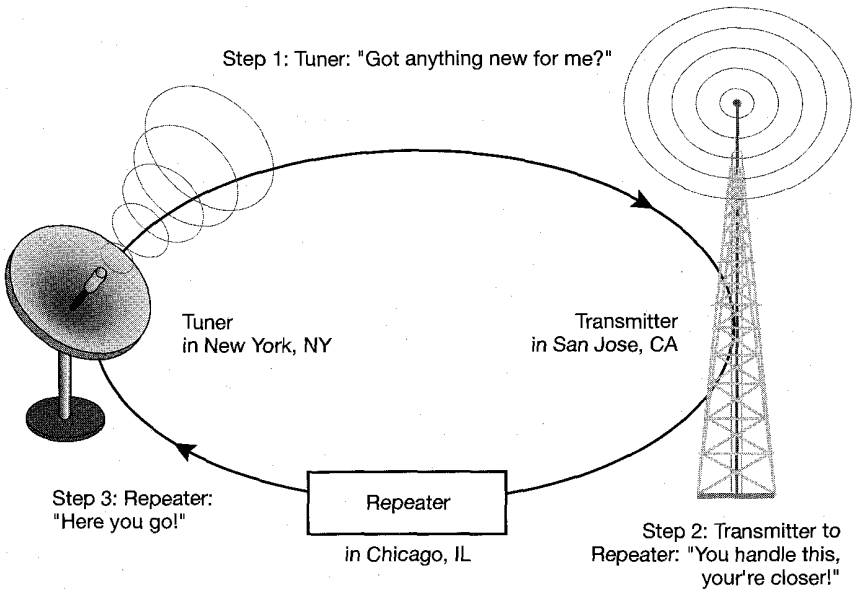


Figure 15-11
The Castanet Repeater enables an organization to distribute the load associated with multiple Tuner requests to multiple Transmitters. This enables the Castanet architecture to efficiently scale to an audience of millions. Repeaters remain entirely invisible to the end user.

Repeaters are completely invisible to the end user and enable the Castanet architecture to reach a potential audience of millions.

15.6 Assessing Castanet

15.6.1 Strengths

Castanet represents an entirely new infrastructure for software and content delivery. The infrastructure is cross-platform, efficient, completely customizable, secure, and scaleable. Here are Castanet's strengths:

Content and software. With Castanet, you can deliver news headlines or a customizable ticker for displaying those headlines. The application is merging with the content, and Castanet offers the greatest flexibility in delivering both.

Wide distribution through Netscape Netcaster. When Castanet was introduced, many skeptics wondered how many users would take the time to download and install a separate push application. Lots of people actually did, but the Castanet Tuner now is embedded directly into Netscape Netcaster, which widens the potential audience of Castanet subscribers to many millions.

Real Java applications. The current batch of Java applets suffers from two important drawbacks. First, the applet is downloaded each time the user visits the Web page. This often can be a lengthy process. Second, applets are restricted to a security "sandbox," which certainly is good for security but bad for creating full-blown Java applications. (Note: Netscape Communicator 4.0 and Internet Explorer 4.0 do allow a certain relaxation of security restrictions for signed applets.) Castanet solves both these problems: Applications are stored locally on the user's hard drive and are available immediately, whether online of offline. Also, application channels have the capability to read from and write to one directory on the user's hard drive, enabling Java applications to maintain persistent data.

Cross-platform. Castanet is built on Java. And Java is cross-platform, meaning that you can run Castanet just as easily on Windows 95/NT, Macintosh, or UNIX.

Customization. You can use Castanet plug-ins to customize each Castanet channel to record user feedback or personalize the channel content. Plug-ins are separate Java or C++ applications and therefore are completely customizable by the channel developer.

Efficient update mechanism. The entire Castanet architecture is designed to minimize bandwidth use as much as possible. Channel updates are performed within a single TCP connection and are performed differentially, ensuring that each Tuner downloads only what is new. The Castanet Proxy Server enables efficient bandwidth use within corporations, and the Castanet Repeaters can ensure rapid updates by distributing the network load among a number of Transmitters.

Scalability. With Castanet Proxy Servers and Repeaters, the Castanet architecture is capable of scaling to thousands and even millions of users.

Security. Each Castanet channel includes all the security measures of Java, including byte-code verification and a modified security sandbox similar to regular Java applets. The Castanet architecture also includes several built-in levels of security to ensure that only certain developers with security clearance can publish or modify channel contents.

15.6.2 Weaknesses

Castanet does have several disadvantages or weaknesses, however:

Development costs. Most push applications require developers to understand JavaScript, HTML, or the *Channel Definition Format* (CDF). Castanet requires a thorough understanding of Java and a much more extensive learning curve than other push platforms. The downsides associated with developing Castanet channels usually are longer development cycles and higher costs.

Java can be slow. Java sometimes can be twice as slow as native applications written in C or C++. The security and portability of Java therefore involve serious performance tradeoffs, even with the best Just-In-Time compilers.

15.7 Moving Beyond Java

In August, 1997, Castanet announced the availability of Castanet UpdateNow. UpdateNow is similar to Castanet, because it automatically distributes, installs, and updates software applications. But UpdateNow works with both Java applications and native code applications written in C, C++, and Microsoft Visual Basic.

One week after the release of UpdateNow, Marimba also reported that it was collaborating with Microsoft on Internet standards for describing software applications. The standard, the *Open Software Description* (OSD), currently is under review by the World Wide Web Consortium.

Much like CDF, OSD is an application language written in *Extensible Markup Language* (XML), but OSD is intended to enable developers to describe software components and their interdependencies. These software applications could be written in Java, C, C++, or Visual Basic. After the standardization process is complete, Marimba plans on including OSD within the Castanet architecture.

Both these initiatives represent a pragmatic strategy by Marimba to move away from a Java-only platform and to focus further on gaining corporate customers.

SUMMARY

Castanet enables the automatic delivery, installation, and updating of Java applications across the Internet and intranets. This capability enables channel providers to push both content and software applications.

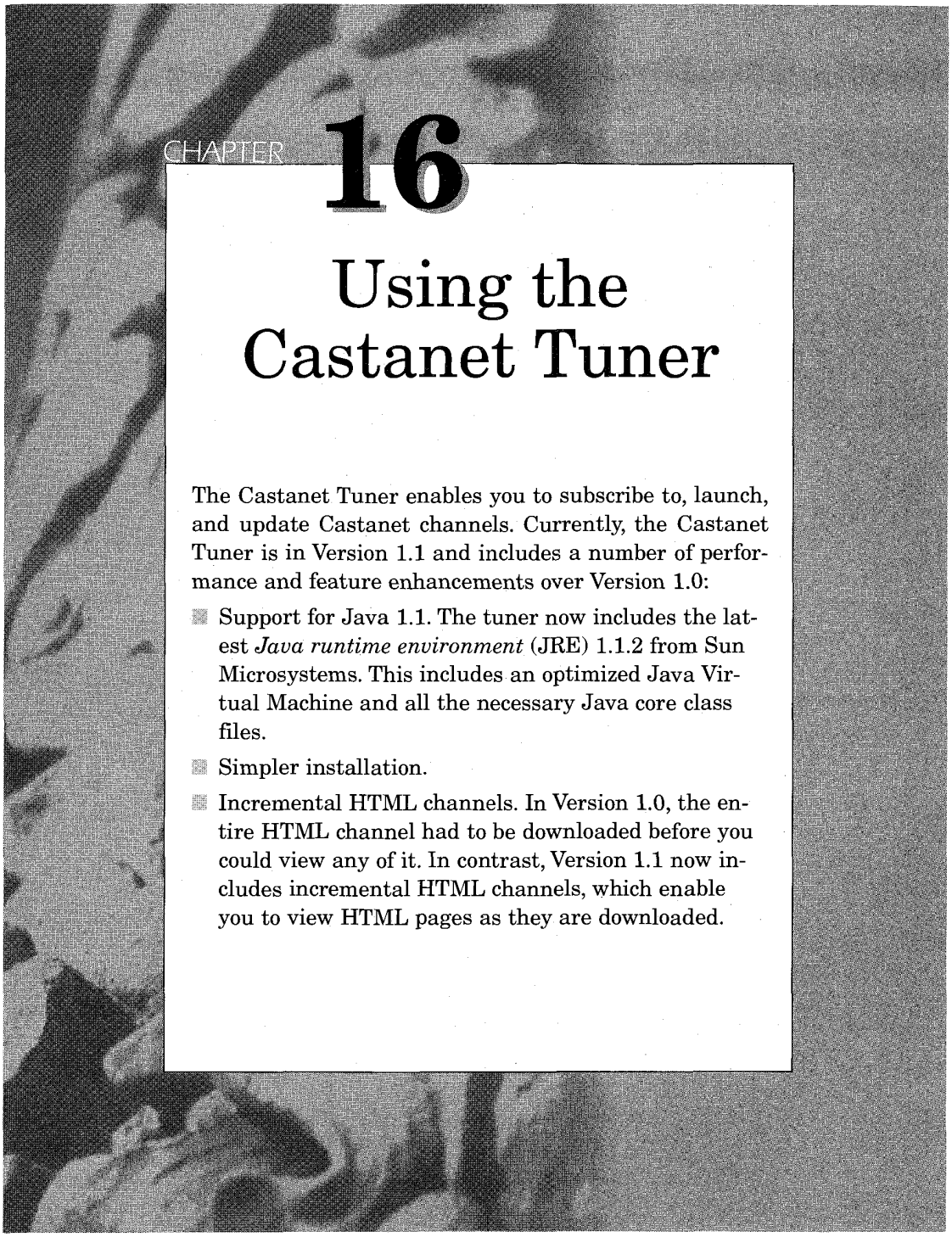
The Castanet architecture is composed primarily of Tuners and Transmitters. Tuners are client applications that manage, launch, and update channel contents. Transmitters are server applications that distribute content and software updates. These two core components are supplemented with Castanet Publisher for publishing channel contents to the Transmitter, Castanet plug-ins for customizing and personalizing channels, the Castanet Proxy Server for managing Castanet traffic in large organizations, and Castanet Repeaters for distributing Tuner updates among several Transmitters.

Currently, Castanet supports four main types of channels: HTML channels, which essentially are directories of HTML documents; applet channels, which are regular Java applets stored locally on the user's hard drive; presentation channels, which are short tutorials or animations created with Marimba's Bongo GUI development software; and application channels, which are Java programs that can receive automatic updates and read from and write to one specific directory on the user's hard drive.

The Castanet architecture is robust, scaleable, secure, and completely customizable. On the downside, it requires a long learning curve, and is dependent on the speed (often slow) of Java. Marimba currently is moving beyond its Java-only platform strategy and is moving toward collaboration with Microsoft on the *Open Software Description* (OSD).

Online Resources

- Castanet White Paper `http://www.marimba.com/datasheets/castanet-wp.html`. What does Castanet mean for you? Where is Castanet going? Check out this recently updated white paper on the Castanet architecture.
- The Open Software Description (OSD) `http://www.marimba.com/osd/`. Over the next few months, new information regarding OSD is sure to appear. Check here for the latest updates and FAQ.



CHAPTER 16

Using the Castanet Tuner

The Castanet Tuner enables you to subscribe to, launch, and update Castanet channels. Currently, the Castanet Tuner is in Version 1.1 and includes a number of performance and feature enhancements over Version 1.0:

- Support for Java 1.1. The tuner now includes the latest *Java runtime environment* (JRE) 1.1.2 from Sun Microsystems. This includes an optimized Java Virtual Machine and all the necessary Java core class files.
- Simpler installation.
- Incremental HTML channels. In Version 1.0, the entire HTML channel had to be downloaded before you could view any of it. In contrast, Version 1.1 now includes incremental HTML channels, which enable you to view HTML pages as they are downloaded.

- Online/Offline mode. The Tuner now includes a specific toggle switch for working online or offline so that you can more easily use your Castanet channels when not connected to the Internet.
- Feedback can be disabled. You now can disable three specific feedback options. This is more or less akin to disabling cookies in your Web browser; you can prevent the recording of any of your habits or personal preferences, but you lose out on the customization provided by Castanet plug-ins.

16.1 Installation

16.1.1 Supported Platforms

Currently, the Castanet Tuner is available for Windows 95, Windows NT 4.0, Macintosh, and Solaris 2.5 systems. The Castanet Tuner also is included in Netscape's Netcaster, making it available on all 17 platforms supported by Netscape.

16.1.2 Hardware Requirements for Windows 95/NT

The hardware requirements for installing the Tuner on Windows 95/NT 4.0 follow:

- At least 16M of RAM
- At least 10M free disk space
- An Internet connection (modem or direct)

16.1.3 Installing the Tuner

The CD-ROM that accompanies this book includes a Windows 95/NT version of the Castanet Tuner. If you have a different platform, free downloads are available from the Marimba Web site at `http:\\www.marimba.com`.

The Castanet Tuner is a self-extracting InstallShield file. Just double-click on the filename (`Tuner1_1.exe`) to begin installation and follow the onscreen instructions.

16.1.4 Adding the Tuner to the Windows Start Menu

To automatically start the Castanet Tuner each time you boot up your machine, you need to add the Tuner to the Windows Startup directory. Follow these steps:

1. Right-click on any empty area of the Windows taskbar.
2. Choose Properties.
3. Select the Start Menu Programs tab.
4. Under the Customize Start Menu options, click the Add button.
5. Click the Browse button to locate the Tuner executable. If you followed the typical installation, the file is located in `C:\Program Files\Marimba\Castanet Tuner\Tuner.exe`.
6. Click the Next button.
7. You have created a Windows shortcut to the Tuner. Now you need to place it in the Windows Startup directory. Double-click on the Startup directory.
8. Click Finish.

16.1.5 The nowin Option

To automatically launch the Tuner without actually opening its window, you need to set its command-line argument. When launched with the `nowin` option, the Tuner displays a quick splash screen and then runs in the background on the Windows taskbar, enabling the Tuner to receive automatic channel updates. Follow these steps:

1. Right-click on any empty area of the Windows taskbar.
2. Choose Properties.
3. Select the Start Menu Programs tab.

4. Under the Customize Start Menu options, click the Advanced button.
5. Open the Startup directory located under the Programs directory.
6. Right-click on the Tuner shortcut and choose Properties.
7. Click the Shortcut tab.
8. Under Target, you should see `C:\Program Files\Marimba\Castanet Tuner\Tuner.exe`. Append the target so that it now reads `C:\Program Files\Marimba\Castanet Tuner\Tuner.exe -nowin`.
9. Click OK.

16.1.6 Launching the Tuner

To manually launch the Castanet Tuner, click the Windows Start button and choose the Castanet Tuner.

16.2 The Tuner Interface

The Tuner interface is composed of five tabbed pages. Before we begin a discussion of subscribing to channels, let's take a quick tour of each Tuner page.

16.2.1 The Marimba Page

The Marimba page is an About page that describes the Castanet Tuner. The page lists the current version of software, the date the version was officially released, and the date the Tuner software was last updated (see Figure 16-1). The Tuner is itself a Castanet channel and checks the main Marimba Transmitter once a week to download any new software upgrades or new channel listings.

16.2.2 The Channels Page

The Channels page lists all your currently subscribed and unsubscribed channels (see Figure 16-2). Each channel includes four columns of information:

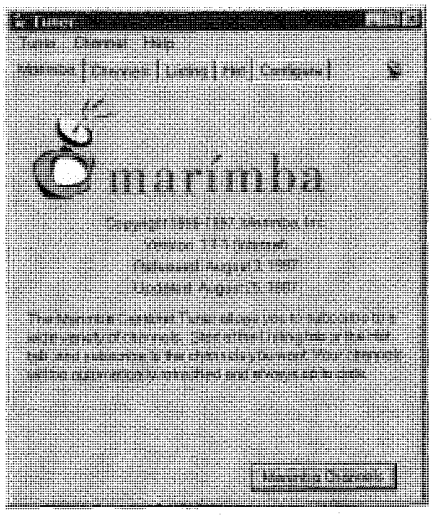


Figure 16-1
The Marimba page lists the current version of software, the date the version was officially released, and the date the Tuner software was last updated.

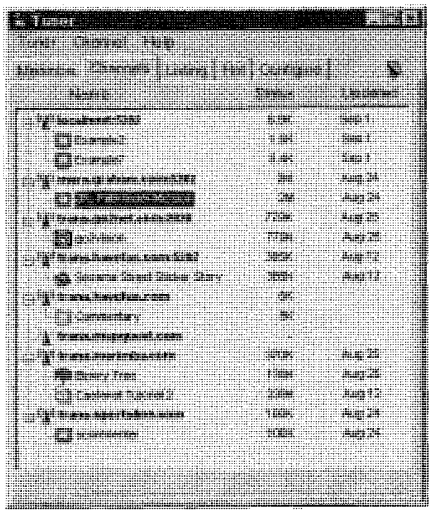


Figure 16-2
The Channels page lists all your currently subscribed and unsubscribed channels.

Column 1	Indicates the name of the Transmitter. Each Transmitter can host multiple channels.
Column 2	Indicates the name of the channel and whether you currently are subscribed to that channel. Unsubscribed channels are grayed out.

Column 3	Indicates the channel status. During channel updates, a small progress bar appears indicating the percentage of downloading completed. When downloading is complete, the column indicates the size (in kilobytes or megabytes) of the channel contents.
Column 4	Indicates when the channel was last updated. If any errors occur during updating, they are reported here.

16.2.3 The Listing Page

The Listing page includes a complete list of channels hosted by individual Castanet Transmitters on the Internet or your intranet (see Figure 16-3).

16.2.4 The Hot Page

The Hot page lists a half dozen premium Castanet channels (see Figure 16-4). Each time the Castanet Tuner is updated, the Hot page is updated with new listings.

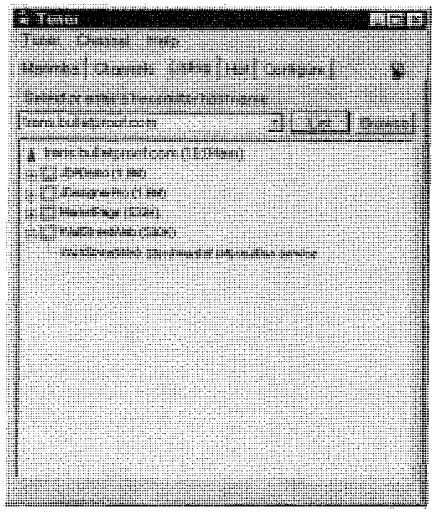


Figure 16-3
The Listing page includes a complete list of channels hosted by individual Castanet Transmitters. To get a short description of each channel, click on the plus (+) sign.

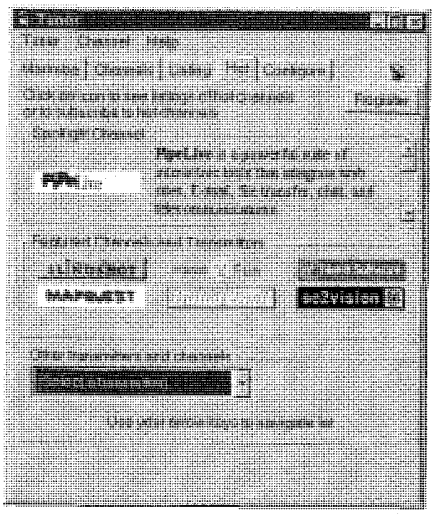


Figure 16-4
The Hot page includes a rotating list of premium Castanet channels. Check here first when looking for new channels.

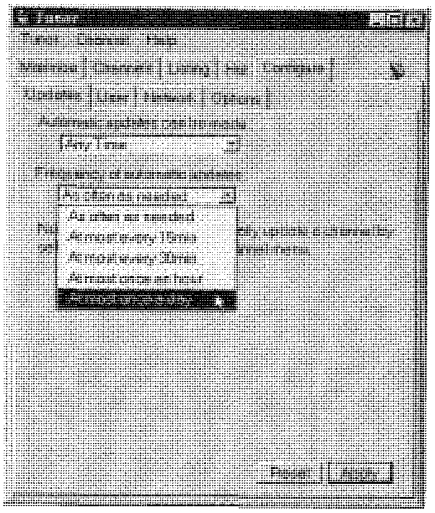


Figure 16-5
The Configure page. The Castanet Tuner is fully customizable for home users with modems and corporate users with direct Internet connections.

16.2.5 The Configure Page

The Configure page enables you to configure the network, update, and user options provided by the Tuner (see Figure 16-5).

16.3 Working with Channels

16.3.1 Subscribing to Channels

After you are oriented to the basic interface, you are ready to start subscribing to channels.

The best place to start is the Marimba site itself. The Marimba Transmitter hosts a number of small, simple channels that you can subscribe to just to make sure your Tuner is working properly. To subscribe to a Marimba channel, follow these steps:

1. Launch the Tuner and click on the Marimba page.
2. Click the Marimba Channels button.

The Tuner connects to the main Marimba Transmitter, downloads a listing of available channels, and switches to the Listing page (see Figure 16-6).

In the Listing page, you now see a complete list of available channels. Note that in the Host Name text box, you see **`trans.marimba.com`**. Most Castanet Transmitter names begin with the **`trans`** prefix instead of the regular **`www`** prefix normally associated with Web servers.

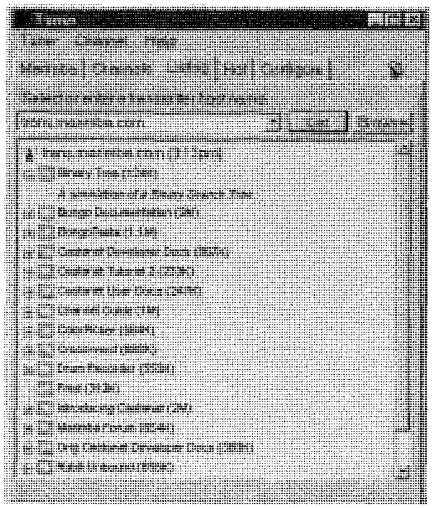


Figure 16-6
The Marimba channels. Marimba hosts more than a dozen of its own channels. Try subscribing to one of these to make sure your Tuner is set up correctly.

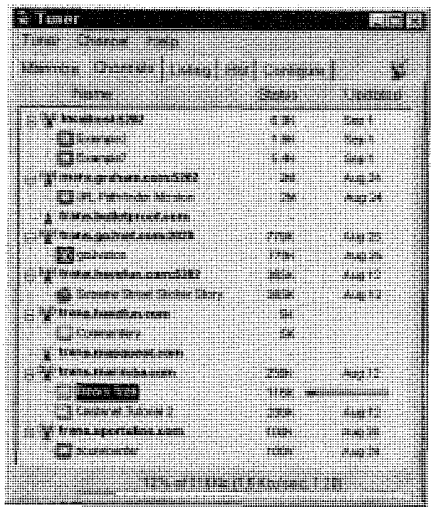


Figure 16-7
As soon as you subscribe to a channel, the Tuner downloads its contents. The progress bar to the right of the channel name indicates the percentage downloaded.

To get a description of a channel, just click on the plus (+) sign to the left of the channel name. To subscribe, just double-click on the channel name.

When you subscribe to a channel, the Tuner switches to the Channels page and begins downloading immediately. You should see a small progress bar to the right of the channel name (see Figure 16-7). This bar indicates the percentage of downloading complete. You also should see red beams streaming into the satellite dish icon at the upper right of the Tuner. A similar animation is displayed in the Windows taskbar so that you can monitor channel updates while working on other applications.

When the channel is fully downloaded, it is launched on your desktop (see Figure 16-8).

16.3.2 Launching Channels

After you download a channel, you can run it at any time, even when you are not connected to the Internet. Just click on the Channels page and double-click on the channel.

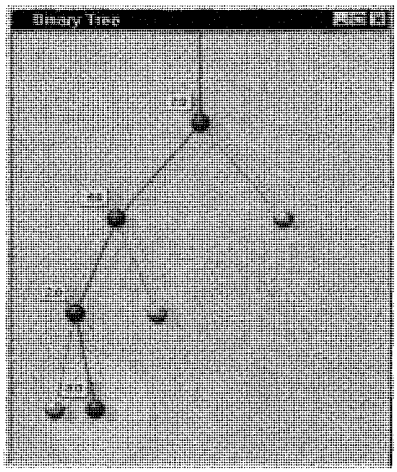


Figure 16-8
The Binary Tree channel. As soon as the channel is downloaded, the Tuner launches the application to the desktop.

16.3.3 Stopping Channels

To stop a channel, just click on the regular Windows close icon in the upper right-hand corner of the application window. Or, click on the Channels page and double-click again on the channel name.

16.3.4 Finding Channels

So where are the rest of the Castanet channels? Check out the Hot page. Remember that this page is a rotating list of premium Castanet channels. Just click any of the channel icons to subscribe. Or, you can scroll through the list of Transmitters for additional channels.

If you already know the name of the Transmitter, you can connect directly to it. Just click on the Listing page. In the Host Name text box, type the name of the host (usually beginning with the `trans` prefix) and press Enter. Most Transmitters are served off of port 80, but some are not. To connect to a port other than port 80, you must enter the hostname, followed by a colon (`:`), followed by the port number—for example, `trans.mapquest.com:5282`.

To get a refreshed listing of channels available on the selected Transmitter, click the List button. To view an HTML listing of available chan-

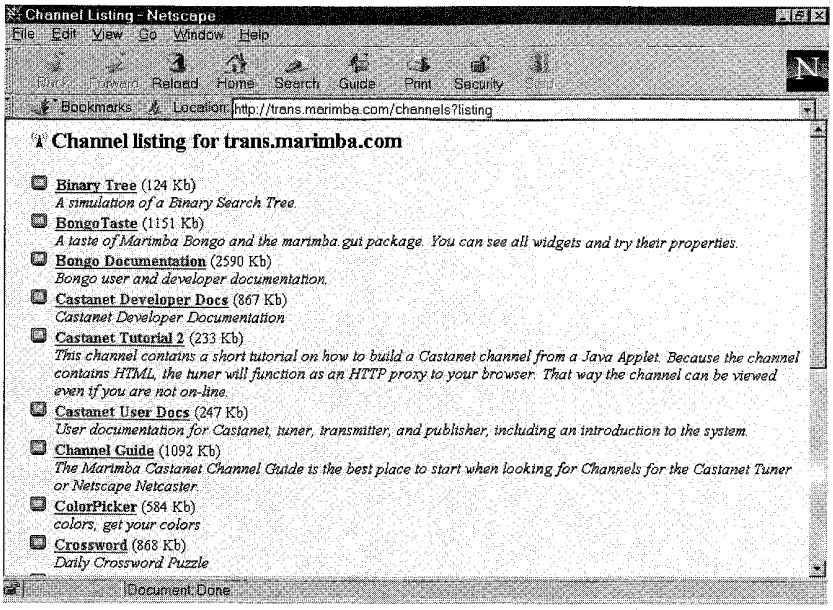


Figure 16-9
Browsing for channels. You can subscribe to new channels via the Tuner or your default Web browser.

nels, click the Browse button. This opens your default Web browser with a complete listing of channels and their descriptions (see Figure 16-9).

16.3.5 Subscribing to Channels via Your Web Browser

You also can find and subscribe to Castanet channels with your default Web browser. Another excellent starting point for finding channels is Marimba's channels directory at `http:\\www.marimba.com\channels` (see Figure 16-10).

If your browser is configured properly with the correct MIME type, you can just click on any of these channels. Your browser automatically launches the Castanet Tuner, and the Tuner immediately downloads the channel contents.

If you try to subscribe to a channel with your Web browser (particularly Internet Explorer), and the browser asks whether you want to save a file to the hard drive, you need to edit your MIME types to include Castanet

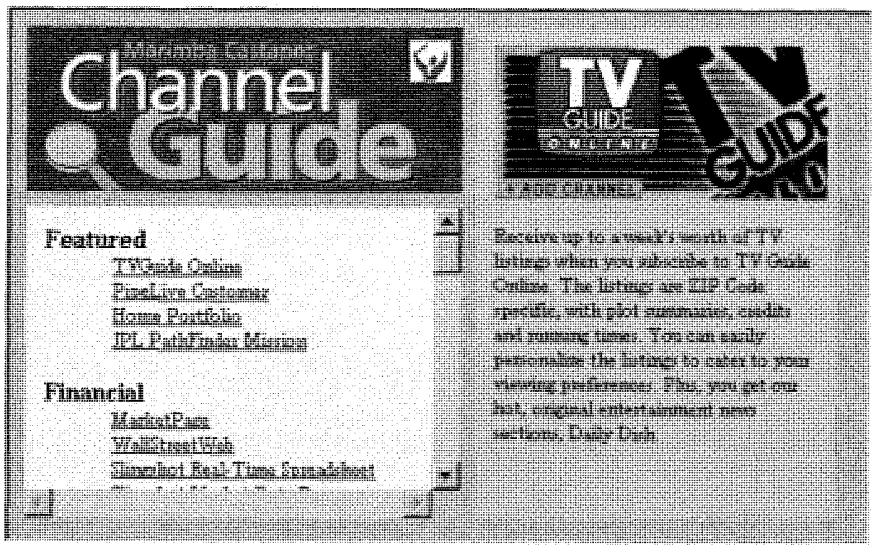


Figure 16-10
The Marimba channels directory. One-stop shopping for new channels.

files. The associated extension for channel files is `.chn`, the content type is `application/marimba`, and the path to the Tuner executable is, by default, `C:\Program Files\Marimba\Castanet Tuner\Tuner.exe`.

Note that if you are using the latest version of Netscape Navigator 4.0 with Netcaster, Netcaster is launched instead of your standalone Castanet Tuner.

16.3.6 Working with HTML Channels

HTML channels are entire collections or directories of HTML documents. In Version 1.0, users had to download the entire HTML channel before viewing any of its contents. In contrast, Version 1.1 now lets users view the HTML channel incrementally.

When you subscribe to an HTML channel, the Tuner downloads the first document (usually `index.html`) and passes this document to your default Web browser. If you click on any of the links in the first page, the Tuner downloads these links. But, after a few minutes, the Tuner proceeds to download the entire HTML channel. After the channel is down-

loaded, it is stored on your hard drive and therefore is always available —even when you are not connected to the Internet.

There is one catch, however: You must set up your Tuner as a Proxy Server for your Web browser. This enables the Tuner to intercept HTTP requests. If the request is related to an HTML channel, the Tuner retrieves the document from the user's hard drive and relays the document back to the browser. Otherwise, the Tuner simply forwards the request, and the Web browser operates as usual.

To set up your Tuner, you must set the Network Proxy options of your browser to point to 127.0.0.1 (local host), port 5283.

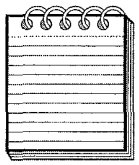
For Netscape Navigator 4.0, follow these steps:

1. Choose Preferences from the Edit menu.
2. Open the Advanced Category and select Proxies.
3. Select Manual Proxy Configuration and click View.
4. Under HTTP Server, enter `127.0.0.1`, port number `5283`.

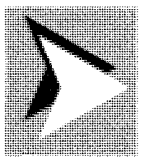
For Internet Explorer 4.0, follow these steps:

1. Choose Options from the View menu.
2. Select the Connection tab.
3. Select the Connect Using a Proxy Server option.
4. For the Proxy Server address, enter `127.0.0.1`, port number `5283`.
5. Click Apply and then click OK.

Even if your Web browser already points to a corporate Proxy Server, this will work, because the Web browser points to the Castanet Tuner, and the Tuner, in turn, points back to the corporate Proxy Server.



NOTE: *If you set your browser to handle HTML channels, the Castanet Tuner must be running* at all times. *If you try to browse and the Tuner is not running, your browser hangs while it looks for a nonexistent Proxy Server, and you cannot download any Web pages!*



TIP: *If you get impatient and want the full channel to download as soon as possible, return to the Tuner and choose Download Remainder from the Channels menu.*

16.3.7 Creating Desktop Shortcuts (Windows Only)

If you are running Castanet on Windows 95 or Windows NT 4.0, you can create desktop shortcuts for each channel application. Click on the Channels page, select your channel, and choose Create Shortcut from the Channels menu.

Each channel you subscribe to is included in the Channels directory of the Windows Start menu (see Figure 16-11). Channels are organized by Transmitter.

16.3.8 Unsubscribing from Channels

To cancel your subscription to a channel, click on the Channels page. Select the channel and then choose Unsubscribe from the Channel menu. The channel name and icon now appear grayed out.

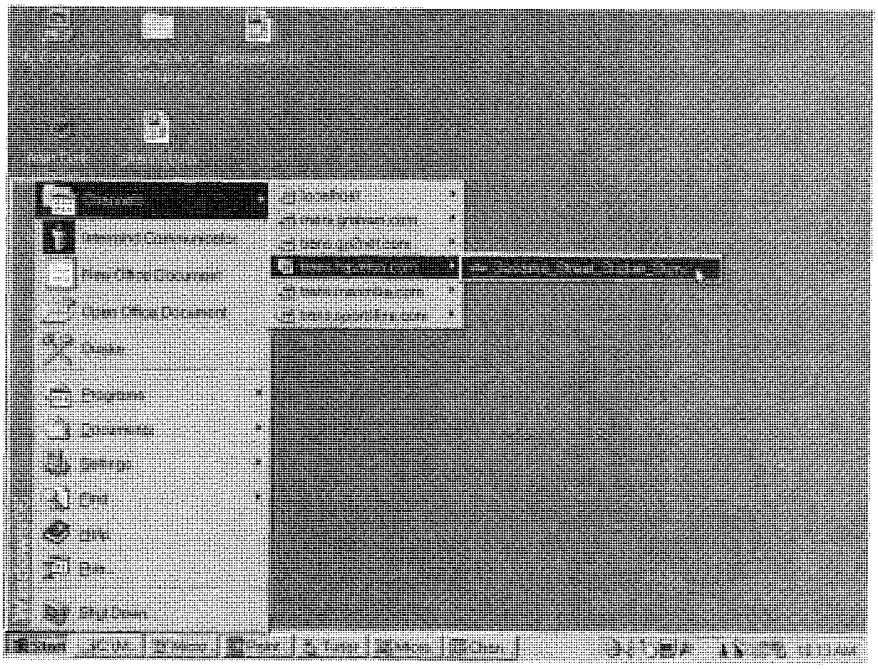


Figure 16-11
The Windows Start menu. Each Castanet channel is added to the Windows Start menu. You also have the option of creating desktop shortcuts for each channel.

When you unsubscribe from a channel, the Tuner deletes all the Java class files, including any Java resource files, such as images or sounds. The Tuner will not delete any preferences or persistent data you created with the channel application, though. When you unsubscribe from a calendar channel, for example, the Java files are deleted, but any persistent data, such as important appointments or engagements, are protected. That way, in case you resubscribe to the channel, your data still is available.

16.3.9 Removing Channels

To unsubscribe to a channel *and* delete all its persistent data, choose to remove it. Again, click on the Channels page, select the channel, and choose Remove from the Channels menu. This removes all Java files, plus any persistent data or user preferences, and removes the channel from the Channels page.

16.3.10 Displaying the Java Console

To display the Java console associated with each channel, choose Show Console from the Tuner menu. This is similar to the Java console provided by Web browsers and is absolutely essential for debugging purposes.

16.4 Updating Options

16.4.1 Automatic Channel Updates

When you subscribe to a channel, the Castanet Transmitter sends all the Java class files plus a Properties file. This file includes a recommended schedule for channel updates.

The Transmitter sends two schedules. The first tells the Tuner how often to check for updates while the channel is actually running on the desktop. This is called as the *Active Update schedule*.

The Active Update schedule usually is reserved for channels that receive and display constantly updated information. A sports ticker might have an Active Update schedule set to every 15 minutes, for example.

That way, when the ticker is running on your desktop, the Tuner checks for new scores fairly frequently and immediately displays them.

The second schedule tells the tuner how often to check for updates when the application channel is *not* running. This is called the *Inactive Update schedule*. The schedule generally is used to check for software upgrades or major new information. Depending on how the channel developer has set up the channel, the channel actually can launch itself if it receives new information or receives new information that the user may deem important. This varies from channel to channel.

This all sounds well and good, but how can you find the update schedule for a particular channel? The update schedule is included on the Channel Properties page. Click on the Channels page, select a channel, and choose Channel Properties from the Channel menu. You see detailed information on the channel, including the following:

- Hostname of the Transmitter
- Date the channel was first published
- Date the channel was last updated
- Date the Tuner last checked for an update
- Channel size
- Channel author and administrator

In the middle of the screen, you see the Active and Inactive Update schedules (see Figure 16-12). The schedules can vary from frequently (meaning less than every five minutes) all the way up to daily or weekly.

16.4.2 Limiting Automatic Updates

As much as possible, the Tuner attempts to follow the recommended schedule set by the channel provider. However, the user still has the last say in setting the frequency of updates. To set a customized schedule of updates, click the Configure page and then select the Updates tab.

On the Updates page, you have two settings (see Figure 16-13). The first setting lets you determine the general range of the update schedule. You can choose from Any Time, Between 9 a.m. – 5 p.m., Not Between 9 a.m. – 5 p.m., and Never (Manual Only).

The second setting lets you to determine the overall frequency of updates. You can choose from As Often As Needed, At Most Every 15 Minutes, At Most Every 30 Minutes, At Most Once an Hour, and At Most Once

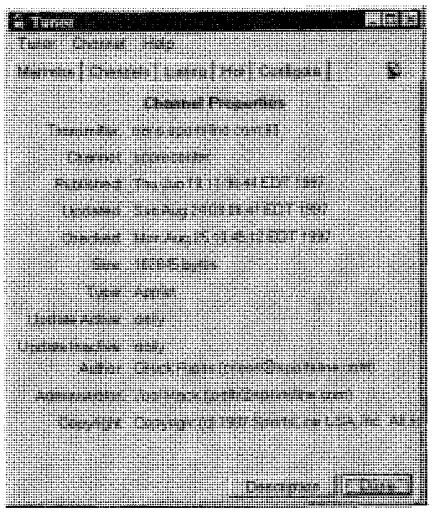


Figure 16-12
The Channel Properties page. Here, you can find the automatic Active and Inactive Update schedules.

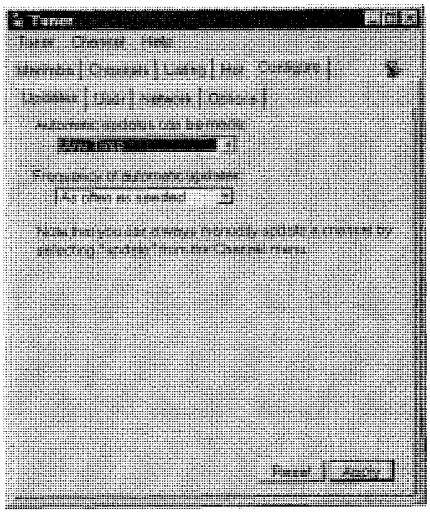


Figure 16-13
The Update Options page. As a user, you can limit the frequency of automatic updates.

a Day. Note that the channel cannot be updated more frequently than the schedule recommended by the channel provider.

After you set your update options, make sure that you click the Apply button.

If you are working from home and don't want any automatic updates, choose Never from the first drop-down menu.

16.4.3 Manually Updating Channels

You can choose to manually update a channel at any time. Just go to the Channels page, select the channel, and choose Update from the Channel menu.

16.4.4 Stopping Channel Updates

To cancel a channel update in progress, choose Cancel Transfers from the Tuner menu.

16.4.5 Working Offline

If you are working on a modem from home or a laptop on the road, you can elect to set the Tuner to offline mode. On the Tuner menu, you should see a checkmark next to the Online option. To go offline, just select the Online option (selecting the Online option toggles you back and forth between online and offline). When you are in offline mode, the Tuner does not attempt to make any network connections or update any channels unless you choose to do so manually.

16.4.6 Updating the Tuner

Once a week, the Tuner prompts you that it needs to be updated (see Figure 16-14). You can agree to the update or postpone it until a later time.

To manually update the Tuner, first quit any channel applications and then choose Update Tuner from the Tuner menu. If the Tuner downloads any updates, it automatically exits, installs the new components, and restarts itself.

16.5 Configuration Options

The Castanet Tuner is fully customizable to meet the needs of both home users with modem connections and corporate users with direct Internet

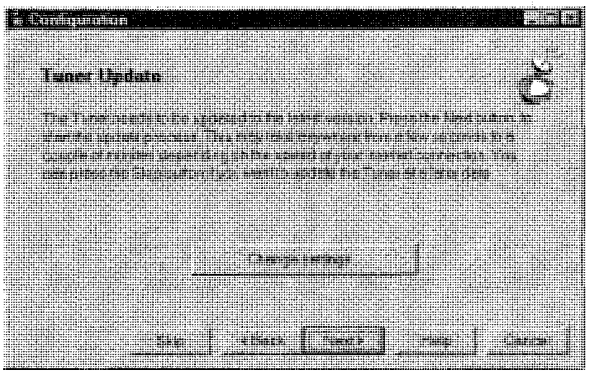
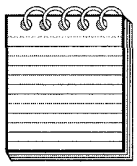


Figure 16-14
The Castanet Tuner is a Castanet channel and updates itself once a week.

connections. The Configure page has four sets of options, which are described in the following sections.



NOTE: *See Section 16.4.2, "Limiting Automatic Updates," for information about update options.*

USER OPTIONS This is optional user information that is sent to Marimba. Current fields include Name, E-mail Address, and Mailing Address. This information is sent only to Marimba and is not released to any channel providers.

NETWORK OPTIONS If your local area network has an HTTP Proxy Server or a Castanet Proxy Server, enter the information here. You must include the hostname and port number, as well as a username and password, if required. If you encounter any difficulties, contact your system administrator.

GENERAL OPTIONS

Connection options	You can set your Tuner to work with a modem or a direct Internet connection.
Windows options	You can elect to have new channels automatically added to the Windows Start menu.
CD-ROM options	Some channels are a hybrid CD-ROM/Web application. Bandwidth-intensive information,

	such as large images, are transferred from the CD-ROM, whereas dynamically updated data is transferred over the Web. To grant your channels access to the CD-ROM, toggle the CD-ROM option.
User Feedback options	There are three user feedback options: Tuner IDs, Log Files, and Profile Data. The Tuner ID uniquely identifies your Castanet Tuner and therefore is similar to a Web cookie. The log file includes a record of your online and offline activity within any given channel. Your profile data includes any preferences or options you set. You can disable any of these options, but this also disables the customization and personalization features provided by Castanet plug-ins.
Print options	Java 1.1 now supports printing. To grant your channels the right to print, toggle the Allow Printing option.

16.6 Exiting the Tuner

To close the onscreen Tuner window, click the close icon in the upper right-hand corner. Unlike in most applications, however, this does not really close the Tuner. It merely minimizes the application to the Windows taskbar so that the Tuner can continue any automatic updates in the background.

To really quit the Tuner, choose Quit from the Tuner menu.

SUMMARY

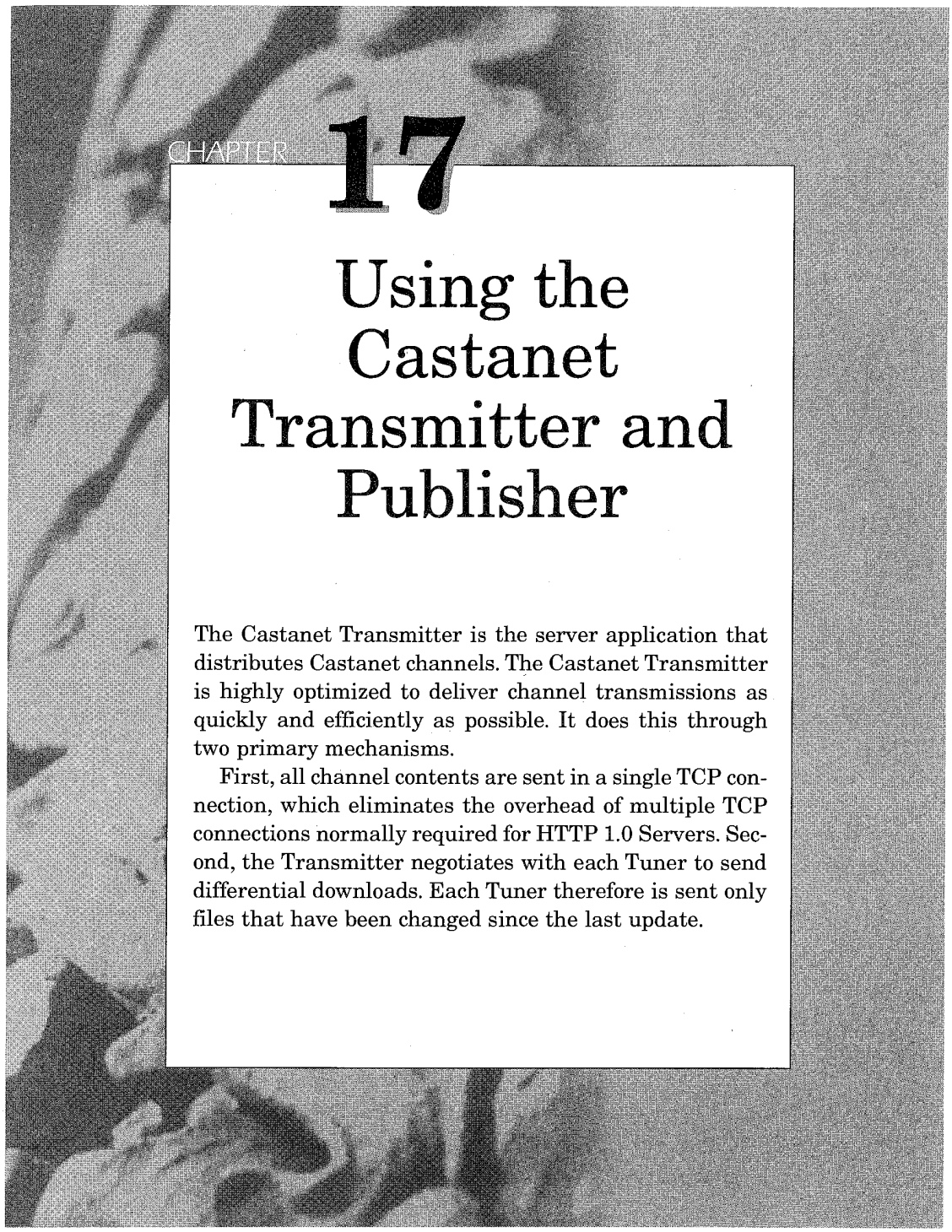
The Castanet Tuner enables you to subscribe to, launch, and automatically update Castanet channels. Version 1.1 has several performance and feature enhancements over Version 1.0, including support for Java 1.1, online and offline modes, incremental HTML channels, and the option to disable user feedback.

To get started with the Tuner (and to make sure everything is working), try subscribing to one of the demonstration channels hosted on the Marimba Transmitter (`trans.marimba.com`). Just open the Marimba page, click the Marimba Channels button, and double-click on any of the channels. The channel is automatically downloaded and launched.

Each Castanet channel follows its own update schedule, which is set by the channel developer. But, as the end user, you have the last word on the frequency of channel updates. These and other customization options are available on the Tuner's Configure page.

Online Resources

- Quick Start Tuner Guide `http://www.marimba.com/doc/Castanet_User_Docs/tuner/1.1/quickstart.html`. The essentials to using the Castanet Tuner.
- Castanet Tuner 1.1 FAQ `http://www.marimba.com/doc/Castanet_User_Docs/tuner/1.1/faq.html`
- Castanet Tuner 1.1 Release Notes `http://www.marimba.com/doc/Castanet_User_Docs/tuner/1.1/notes.html`. Is this a bug or a feature?



CHAPTER 17

Using the Castanet Transmitter and Publisher

The Castanet Transmitter is the server application that distributes Castanet channels. The Castanet Transmitter is highly optimized to deliver channel transmissions as quickly and efficiently as possible. It does this through two primary mechanisms.

First, all channel contents are sent in a single TCP connection, which eliminates the overhead of multiple TCP connections normally required for HTTP 1.0 Servers. Second, the Transmitter negotiates with each Tuner to send differential downloads. Each Tuner therefore is sent only files that have been changed since the last update.

Castanet Publisher, on the other hand, enables developers to quickly transfer Java class and data files from their local hard drives to the Castanet Transmitter. This enables multiple developers to publish to the same Transmitter while maintaining the necessary security precautions. Much like Tuner updates, publishing to the Transmitter is differential and therefore highly efficient. After the channel contents are published, they are available immediately to all subscribers.

Currently, both the Transmitter and Publisher are in Version 1.1. And, Version 1.1 of the Publisher now includes specific support for publishing to the Castanet Tuner built into Netscape Netcaster.

If you are planning on developing your own channels, it is essential to understand both the Transmitter and Publisher. Therefore, before moving on to the next few chapters on channel development, make sure that you read this chapter first.

17.1 Installation

17.1.1 Platforms Supported

Currently, Version 1.1 of the Castanet Transmitter and Publisher are available for Windows 95, Windows NT 4.0, and Solaris 2.5 systems.

17.1.2 Hardware Requirements

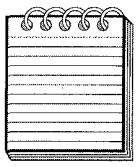
To run the Castanet Transmitter and Publisher on Windows 95/NT, you must have the following:

- At least 16M of RAM
- At least 10M of free disk space; additional disk space is needed for channels as they are published

17.1.3 Installing the Transmitter/Publisher

The Castanet Transmitter and Publisher come bundled together in a single, self-extracting InstallShield archive. An evaluation copy of the Win-

dows 95/NT version is available on the CD-ROM that accompanies this book. Evaluation copies for other platforms are available at the Marimba Web site at `http:\\www.marimba.com`.



NOTE: *The evaluation copy of the Transmitter/Publisher enables you publish up to three channels to three individual subscribers and is for development purposes only. Check the Marimba site for the latest price quotes on purchasing full versions of the Transmitter.*

Before attempting to install the Transmitter/Publisher, make sure that you do the following:

- Uninstall any previous versions of the Castanet Transmitter.
- Exit any running Castanet applications. Make sure that you check the Windows taskbar to see whether the Tuner is running in the background.

Then double-click on the self-extracting file `Trans.exe` and follow the onscreen instructions.

17.2 Launching the Transmitter

You can launch the Transmitter from the Windows Start menu or from the command line (`C:\Program Files\Marimba\Castanet Transmitter\Transmitter.exe`).

17.3 Transmitter Configuration

When you first launch the Castanet Transmitter, you need to set the proper configuration parameters. You can set these parameters by flipping through the configuration pages provided by the Transmitter user interface (see Figure 17-1).

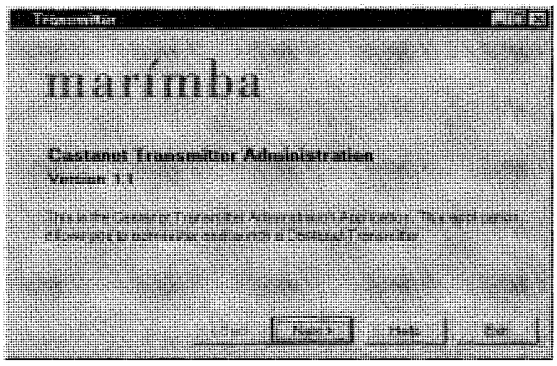
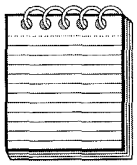


Figure 17-1
The Castanet Transmitter opening page. Keep clicking the Next button to fully configure your Transmitter.

17.3.1 Transmitter Channel Directory

First up, the Transmitter Channel directory (see Figure 17-2). This directory contains all the files pertaining to your individual channels, including the Java class files, plug-in programs, and any logging or profile data sent back from your subscribers. Unfortunately, there is no Browse button here, so you must enter the full pathname of your Channel directory. The Channel directory could be located in the **Marimba** directory or anywhere on your hard drive. If the directory does not exist, you are prompted to confirm its creation.



NOTE: *The Channel directory also can be located on a remote machine on your local network, but Marimba recommends a local directory for optimum performance.*

If you choose to uninstall the Transmitter at a later date, the Channel directory is not deleted. This enables you to easily upgrade to new versions of Transmitter software without losing your existing channels or settings.

17.3.2 Transmitter Hostname and Port

The next page asks you for the full domain name and port number of the Transmitter (see Figure 17-3). If you are planning to use the Transmitter for real delivery of applications across the Internet, you must include the

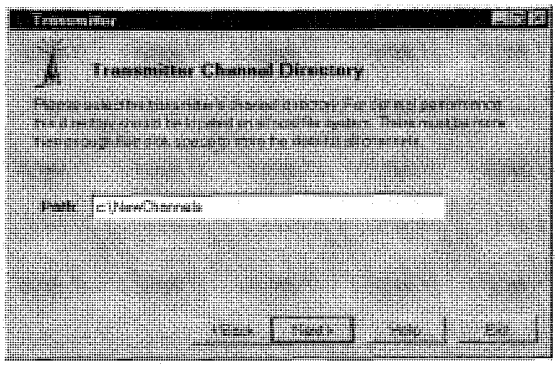


Figure 17-2
Transmitter configuration, step 1: The Transmitter Channel directory.

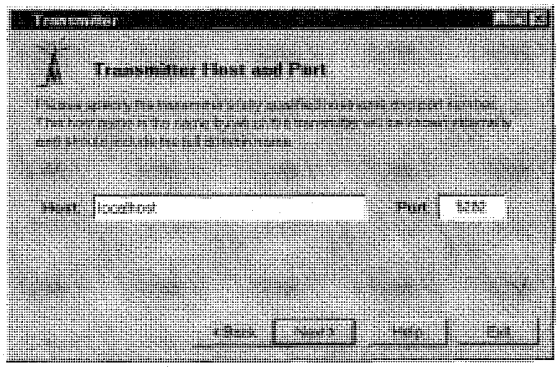


Figure 17-3
Transmitter configuration, step 2: Hostname and port number.

full domain name of your machine here. If you are planning to use the Transmitter for development purposes only, set the hostname to **`localhost`**. This enables you to download and debug channel applications entirely on your local machine.

If your current machine already is running a Web server, make sure to pick a port number other than 80 (port 80 is reserved for HTTP Servers).

17.3.3 Transmitter Access

The Transmitter Access page enables Castanet's built-in security mechanism (see Figure 17-4). To password protect your Transmitter, enter the password here. Each time you use Castanet Publisher to publish new chan-

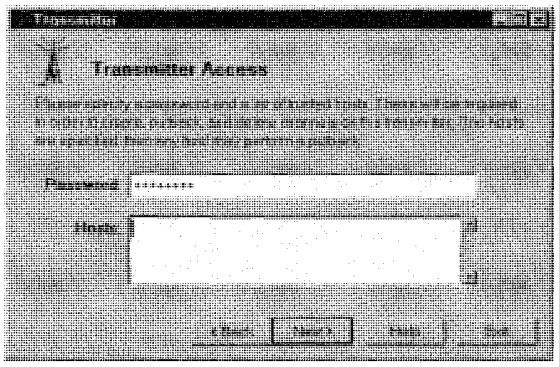


Figure 17-4
Transmitter configuration, step 3: Transmitter access and password protection.

nels or to modify old channels, you are asked to enter this password. If you do not want to password protect your channel, leave this field blank.

As an extra measure of security, you can enter a list of trusted hosts. Enter the full domain names of trusted hosts here, or leave this section blank to accept requests from any host.

17.3.4 Publish Notifications

Each time a new channel is published or updated, you can direct the Transmitter to send e-mail to the specified e-mail address (see Figure 17-5). As a default, the Transmitter sends these updates to Marimba so that it can be aware of new channels. If you do not want e-mail notification, leave the E-Mail Address field blank.

17.3.5 Advanced Settings

After you complete the Transmitter configuration, you can launch it by clicking the Launch button (see Figure 17-6).

Or, you can opt to configure the Transmitter's Advanced settings:

Transmitter Concurrency: Each time a Transmitter receives a Tuner request, the request is assigned to a thread process. This enables the Transmitter to handle multiple Tuner requests simultaneously. To optimize the performance of the Transmitter, you can set the maximum number of available threads. This can be a tricky issue, though.

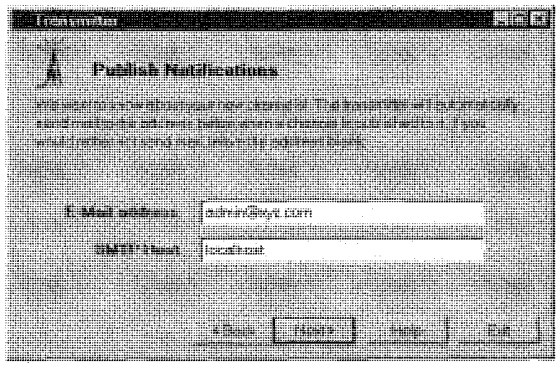


Figure 17-5
Transmitter configuration, step 4: E-mail notification.

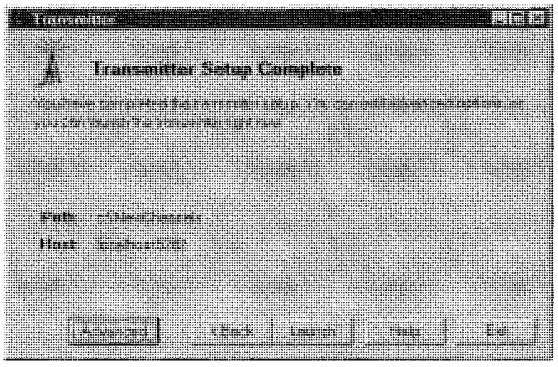


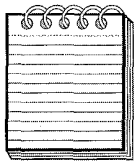
Figure 17-6
Transmitter configuration, step 5: Launch. After you complete the standard configuration, you can launch the Transmitter. Or, click the Advanced button to set the Threads and Cache options.

If you have too few threads, you cannot handle many requests at once. And, if you have too many threads, they may swamp your system and actually slow performance. The optimum number of threads depends on both your hardware and overall traffic patterns, and you have to be willing to experiment a bit. When in doubt, keep the default settings, especially if you are using the Transmitter only for development purposes.

Transmitter Cache: Each Transmitter has a memory cache and a disk cache. If you want to further optimize the performance of your Transmitter, consider increasing these cache limits. But make sure to test the performance. And, again, when in doubt, keep the default values, especially if you are using the Transmitter only for development purposes.

17.3.6 The Properties File

After you complete the Transmitter configuration, the Transmitter stores your settings in the `properties.txt` file. This file is located in the Channels directory you set up in section 17.3.1. If you want to change these settings at any time, you can use the Transmitter interface, or your can manually edit the `properties.txt` file. Listing 17-1 shows a sample `properties.txt` file.



NOTE: Do not attempt to manually modify the Transmitter password. This is an encrypted string that must be set via the Transmitter GUI interface.

17.3.7 The License Agreement

As noted earlier in this chapter, the evaluation copy of the Castanet Transmitter enables you to publish only three channels to three unique subscribers. If you try to publish a new channel and the Transmitter refuses, check the license agreement. Just click the View License button on the last page of the Transmitter setup. The license tells you the maximum number of channels, the number of actual channels, the maximum number of users, and the number of actual users (see Figure 17-7).

17.4 Using the Castanet Publisher

You now have your Transmitter fully configured and running. But is it really working? To test the Transmitter, you need to give it a real channel.

Listing 17-1
A sample `properties.txt` file.

```
#Transmitter Properties
server.cachesize=32
server.host=localhost
server.maxthreads=32
server.memdisk=4
server.password=V3lvlWiuZyE=
server.port=5282
server.putback.mailto=
server.smtphost=localhost
server.trustedhosts=localhost/127.0.0.1
```

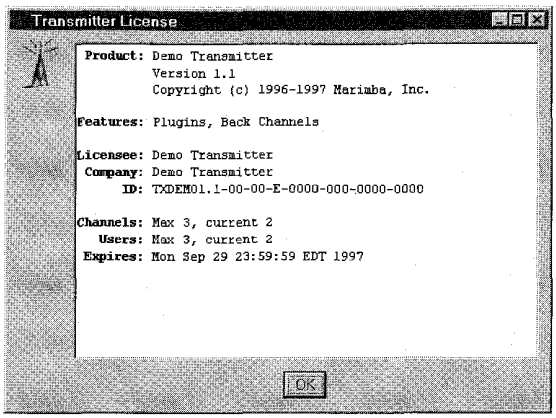
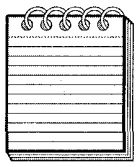


Figure 17-7
The Transmitter license agreement.

In other words, you need to publish a new channel with Castanet Publisher. After the Transmitter has a channel, you can use your Tuner to test the Transmitter. Hang in there. We are very close!



NOTE: *The next section is your introduction to Castanet Publisher. More details on Publisher are provided in Chapters 18 and 19.*

17.4.1 Launching Publisher

First, launch Castanet Publisher. Publisher is included in the **`Castanet Transmitter`** directory on the Windows Start menu.

17.4.2 Publishing a Sample Channel

Fortunately, the evaluation copy of the Castanet Transmitter includes several sample channels. Let's publish one of these channels just to make sure everything is working.

When you launch Publisher, you see the introductory screen (see Figure 17-8). Click the Channels button to get started. You now should see a blank screen called Channels under Development (see Figure 17-9). As you continue to add new channels, you will see them listed here.

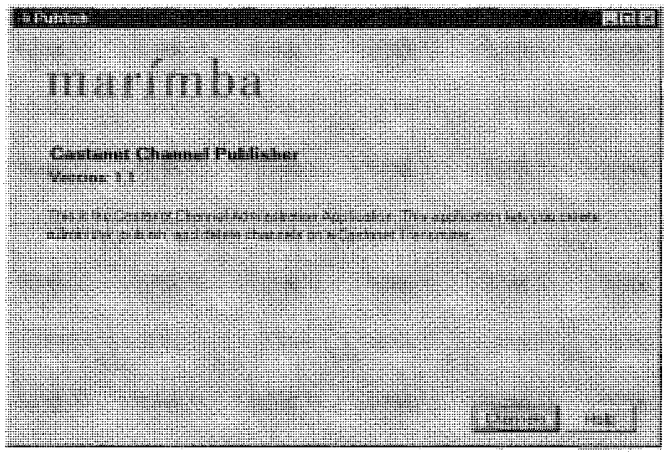


Figure 17-8
The Castanet Publisher, the official gateway to your Transmitter.

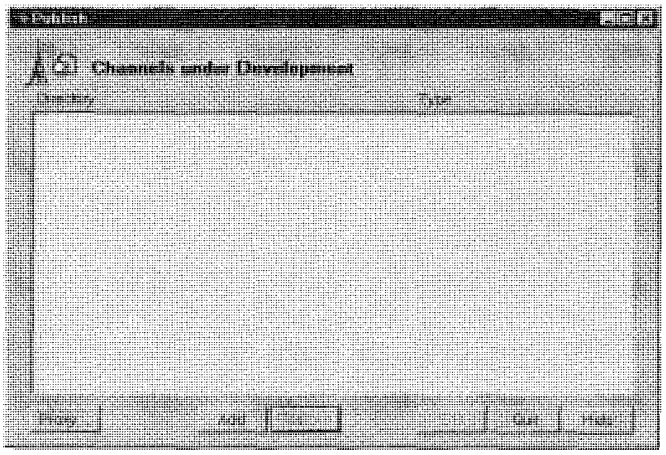


Figure 17-9
The Channels under Development screen.

Click the Add button to add a new channel. Then enter the full pathname of your channel's directory. Again, unfortunately, there is no Browse button, so you have to enter the entire pathname. In the latest Transmitter release, the sample channels are included in the **`Publisher`** subdirectory under the **`developers`** subdirectory—**`c:\Program Files\Marimba\Castanet Publisher\developers\channels\`**. If you cannot find

the channels here, look around (previous releases install the channels in the `Castanet Transmitter` directory).

The sample channels directory includes six sample channels. For now, we will pick the Crossword channel, so enter the full pathname of the crossword directory: `c:\Program Files\Marimba\Castanet Publisher\developers \channels\Crossword`. Then click the Add button.

You now should see the Channels under Development page again with your channel listed at the top (see Figure 17-10). Now click the Edit button to set the channel properties. You will see the main interface for configuring all of the many parameters associated with channels (see Figure 17-11).

Each sample channel includes its own `properties.txt` file (not to be confused with the Transmitter's `properties.txt` file). This file includes parameters for the channel, and you will notice that each of the configuration pages already is populated with settings. To publish the channel correctly, you want to keep most of these settings. In Chapters 18 and 19, we will return to the exact meaning of each of these parameters.

For now, just make sure that the Transmitter page options are correct. In particular, be sure to enter the right hostname (usually, `localhost`) and the right port number. Also, make sure that you have entered the correct password. Leave all the other settings as they are.

When you are finished, click the Apply button. Then click the Publish button. The files are copied over to the Transmitter, and you should re-

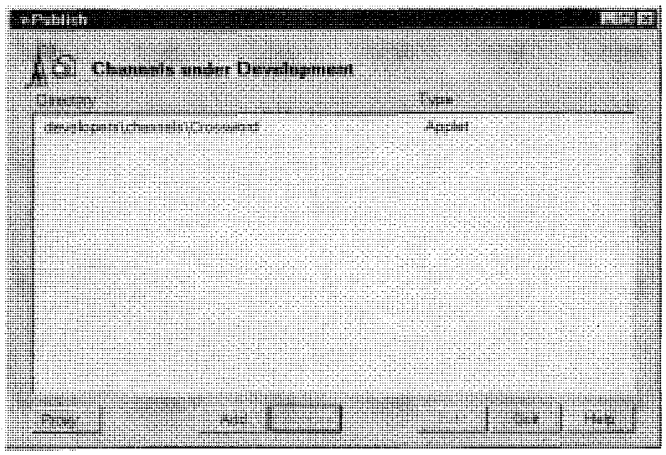


Figure 17-10
Viewing your sample channel name on the Channels under Development screen.

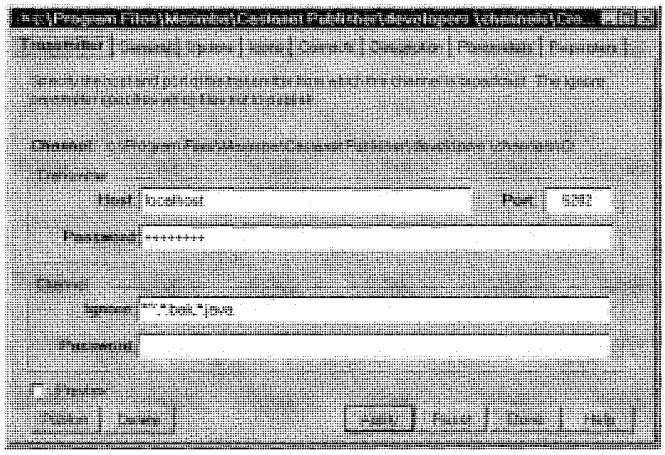


Figure 17-11 The Castanet Publisher configuration options. You can set a wide range of properties for each channel. For now, just make sure that the Transmitter options are correct.

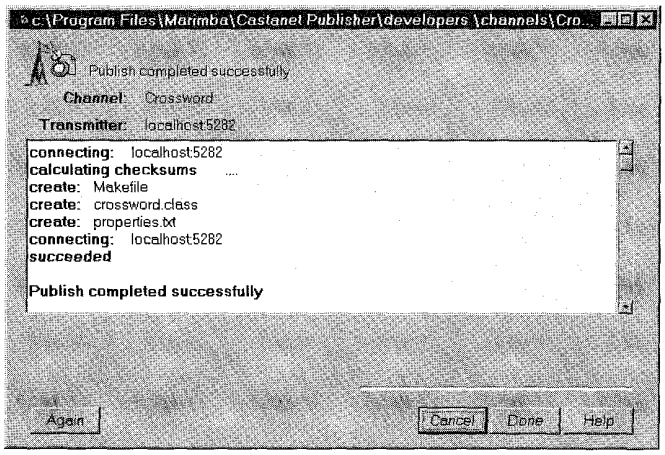


Figure 17-12 The Castanet Publisher. Success!

ceive a confirmation of success (see Figure 17-12). If any errors have occurred, they are reported here.

If you have any errors in the publication of your channel, make sure that the Transmitter is running. And, make sure that you have entered the correct hostname, port number, and password.

To exit the Publisher, click the Done button (this button is a bit of a misnomer and actually should be labeled **Back**) until you return to the starting page; then click Quit.

17.4.3 Testing the Transmitter

Your transmitter now is ready to serve a single Castanet channel. To test the entire system, launch your Castanet Tuner and click the Listing page.

In the Hostname field, enter the name of your Castanet Transmitter. If your Transmitter is not running on port 80, you must remember to specify the correct port—for example, `localhost:5282`. Then click List. The Crossword channel now appears (see Figure 17-13). Just double-click to subscribe.

If the channel downloads and executes properly, everything is set up correctly and you are ready to start developing your own channels (see Figure 17-14).

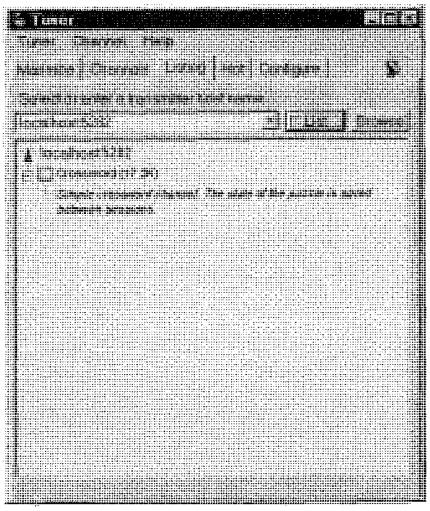


Figure 17-13
The Castanet Tuner. Use the Tuner to test your Transmitter. You should see at least one channel listed here.

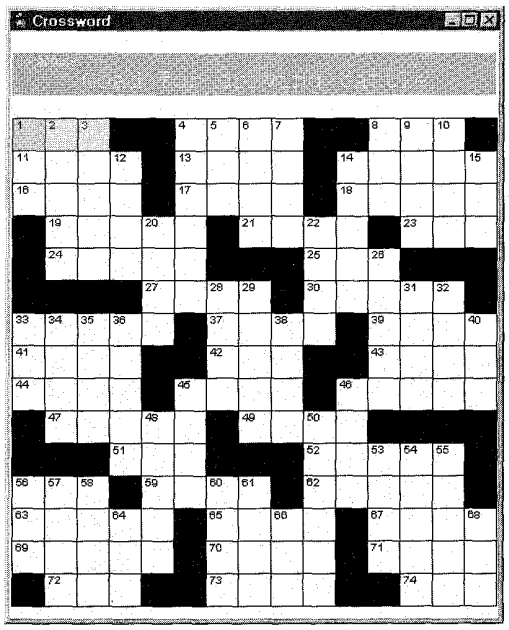


Figure 17-14
The Crossword channel. *If you see this, everything is working properly.*

SUMMARY

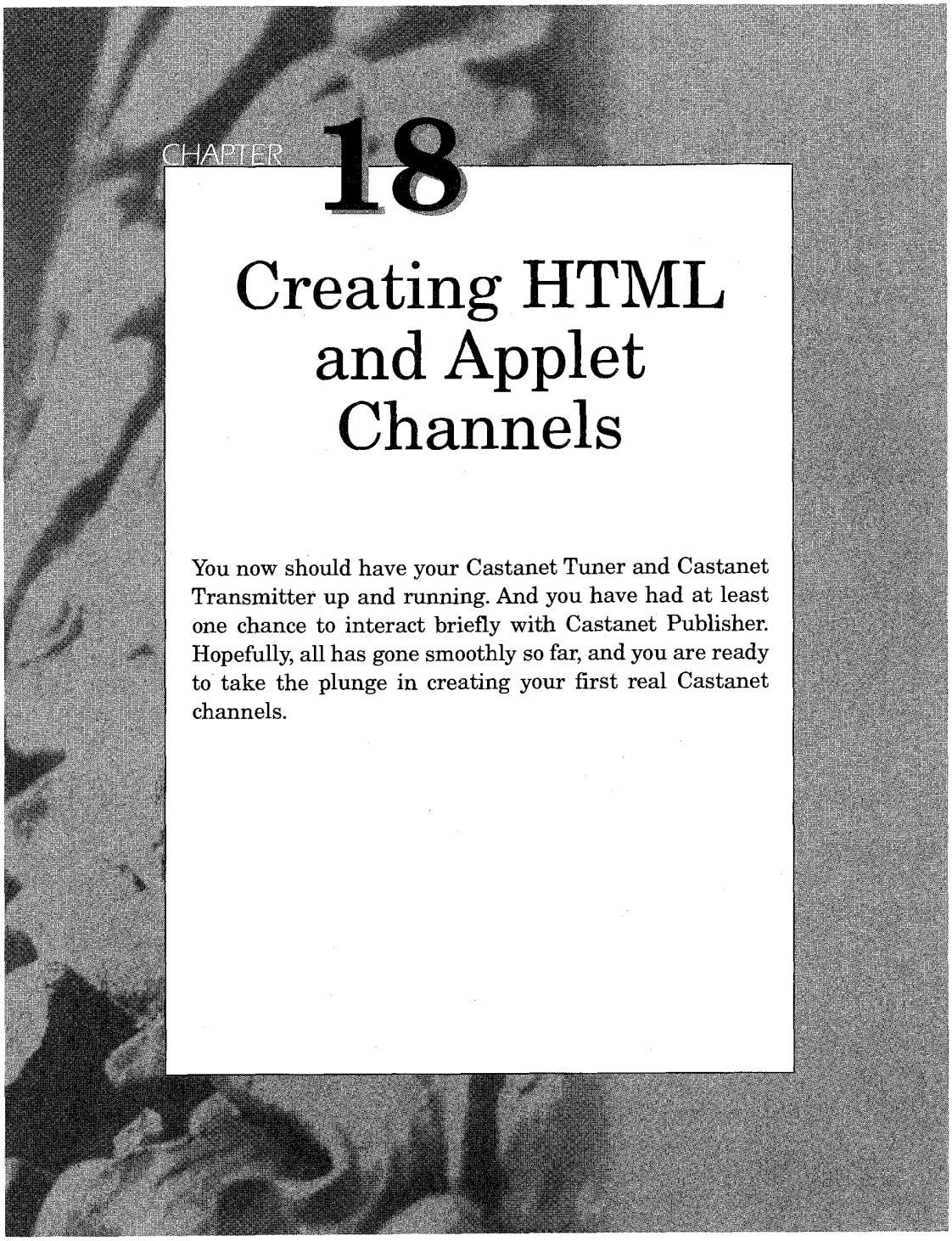
If you are serious about developing your own Castanet channels, you have to start with a thorough understanding of the Castanet Transmitter and Publisher. Working with these two programs is essential to developing and debugging your own channels. So take the time now to work with these applications.

The Castanet Transmitter distributes channels and channel updates to Castanet Tuners. The Transmitter is relatively easy to set up and can be set up on your local machine so that you can develop and debug channels entirely on one machine.

Castanet Publisher enables developers to copy over their Java class files and other data from their local hard drives to Castanet Publisher. Even if you have your Tuner, Publisher, and Transmitter on the same machine (for development purposes, of course), there is no other way to publish your channel. This chapter covered just the basics of using the Publisher—just enough to enable you to test the Transmitter. The next chapter covers the Publisher in much greater detail.

Online Resources

- The Castanet Transmitter, The Big Picture `http://www.marimba.com/doc/Castanet_User_Docs/transmitter/1.1/bigpicture.html`
- The Castanet Transmitter, Quick Start Guide `http://www.marimba.com/doc/Castanet_User_Docs/transmitter/1.1/quickstart.html#arguments`
- Castanet Transmitter 1.1 Release Notes `http://www.marimba.com/doc/Castanet_User_Docs/transmitter/1.1/notes.html`
- The Castanet Publisher, Quick Start Guide `http://www.marimba.com/doc/Castanet_User_Docs/publisher/1.1/quickstart.html`



CHAPTER 18

Creating HTML and Applet Channels

You now should have your Castanet Tuner and Castanet Transmitter up and running. And you have had at least one chance to interact briefly with Castanet Publisher. Hopefully, all has gone smoothly so far, and you are ready to take the plunge in creating your first real Castanet channels.

HTML channels are the easiest to build, so we will begin our discussion there. After that, we will progress to applet channels. The next chapter will move on to full-blown application channels, which take full advantage of the Castanet architecture. Along the way, we will explore all the publishing options supported by Castanet Publisher.

18.1 HTML Channels

18.1.1 HTML Channels Defined

HTML channels are collections or directories of HTML pages transferred via the Castanet architecture. Like other Castanet channels, HTML channels are updated periodically on a schedule set by the channel developer. The update schedule can be very frequent (every hour, for example), daily, or weekly. This ensures that end users always have the most up-to-date information.

HTML channels are stored on the user's hard drive and therefore are always available, whether online or offline. And, because HTML channels take advantage of Castanet's efficient automatic updating mechanism, channels can be updated frequently, quickly, and efficiently. The update mechanism is considerably faster and more efficient than the Web-crawling technique used by Netscape Netcaster.

The Castanet architecture is designed mainly for the efficient transfer of files, but it is not designed for displaying HTML pages. To view HTML channels, end users therefore must use the Castanet Tuner with their default Web browser. Users must set their Castanet Tuner to act as a Proxy Server for their Web browser. This enables the Web browser to access the files previously stored by the Castanet Tuner.

Unfortunately, this might be a little too much to ask for the basic end user, and it adds a layer of complexity to the viewing of HTML channels. The user also must remember to leave the Tuner running at all times in order to browse the Web. But remember that most users will not be using the standalone Castanet Tuner. Most users will be using the Castanet Tuner embedded inside Netscape Netcaster, and the Tuner inside Netcaster is integrated seamlessly with the Netscape Navigator browser. No special configuration is required, and viewing HTML channels is as easy

as viewing an HTML page. This makes things considerably easier and significantly expands the potential audience for HTML channels.

In addition to the efficient updating mechanism provided by Castanet, HTML channels provide an efficient mechanism for logging user activity. By including a few short HTML-like tags in your documents, you can direct the Castanet Tuner to record specific user activity and to relay this information back to the Castanet Transmitter.

18.1.2 Incremental HTML Channels Defined

In the original Castanet Tuner, the entire HTML channel had to be downloaded before you could view any of it. With Castanet Tuner 1.1, and the Castanet Tuner embedded in Netcaster, you now can view your HTML channels incrementally. This gives the impression of a much faster response time.

If the first page of your HTML channel is `index.html`, your channel is considered an incremental HTML channel. Otherwise, it's a regular HTML channel that is downloaded in its entirety. If you are delivering an incremental channel, the Tuner first retrieves the `index.html` page. If the user then clicks on any subsequent links, the Tuner downloads these pages. After a few minutes, the Tuner proceeds to download the entire channel contents and stores these files on the user's hard drive.

18.1.3 Creating Your HTML Documents

The first step in creating your HTML channel is to create your HTML documents. You could reuse existing content from your current Web site or create new channel-specific content. Either way, this is just regular HTML. Nothing new here.

Because your HTML content will be parsed by a separate browser (or, in the case of Netcaster, your content will be parsed by Netscape Navigator), your HTML channel can include anything that might appear on a regular Web page. You therefore can take full advantage of frames, images, JavaScript, and dynamic HTML. You also can embed your own Java applets, or if you are writing specifically for Internet Explorer, you even can include your own ActiveX controls.

18.1.4 Creating Your HTML Channel Directory

With regular Web sites, you simply take your HTML documents and place them somewhere in the root directory of your Web server. With HTML channels, however, you first create a base channel directory on your local hard drive. Anything in this directory, including any subdirectories, is considered part of the channel. You then transfer this directory to the Castanet Transmitter via Castanet Publisher. Any time you want to make changes to your HTML documents, you modify them in your local directory and republish them with Publisher.

18.2 A Sample HTML Channel

18.2.1 The Castanet Tuner Help Channel

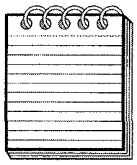
With this background information in hand, let's try creating a sample HTML channel. And, instead of creating an entirely new set of HTML documents, let's just use the HTML documents provided by the Castanet Tuner. The goal is to take the existing help documents concerning the Castanet Tuner and repackage them as an HTML channel. That way, users always can stay on top of the latest developments and receive the most up-to-date help documentation.

If you followed the standard Tuner installation, the Tuner help documents are installed in `C:\Program Files\Marimba\Castanet Tuner\lib\doc\tuner`. If you cannot find these documents, try taking another set of HTML documents—perhaps from your existing Web site.

18.2.2 Publishing HTML Channels

After you identify the base directory for your HTML channel, you are ready to transfer the files over via Castanet Publisher. To get started, make sure that the Transmitter is running, and then launch Castanet Publisher.

On the opening screen, click the Channels button. Then click the Add button to add your HTML channel. For the directory, enter the full pathname of the Tuner help documentation: `c:\Program Files\Marimba\Castanet Tuner\lib\docs\tuner\`. Then click the Add button.



NOTE: *The Add and Create buttons on the Directory page can be a bit misleading. Because you are creating a new channel, you might think to click the Create button. The Create button is reserved for creating new directories on your hard drive, though.*

Now that you have added your channel to the list in the Channels under Development screen, you are ready to set the channel properties. Click the Edit button to proceed. Each of the channel properties is explained in the following sections.

18.2.3 Transmitter Properties

The Transmitter Properties page tells Publisher where to publish your channel (see Figure 18-1). You can publish to multiple Transmitters, so make sure these settings are correct:

Host	The full Internet domain name of your Transmitter. If you are publishing to a Transmitter on the same machine, enter `localhost` here.
Port	The Transmitter listens for Publisher and Tuner requests on this TCP port. Make sure that you have the right port number, and make sure that you are not accidentally connecting to a Web server (port 80) running on the same machine as the Transmitter.
Password	This is the same password you set during the initial configuration of your Transmitter (see Chapter 17).

In addition to these options, you will notice two more settings in the Channel options:

Ignore	If you have specific directories or files that you do *not* want to publish to the Transmitter, you can enter them here. Most of the time, this is reserved for publishing

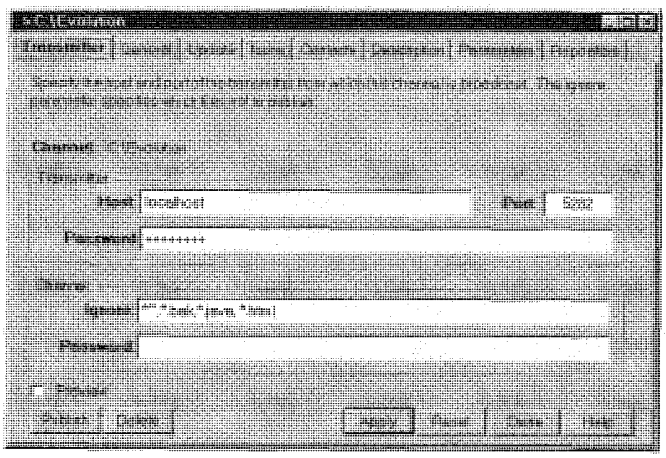


Figure 18-1
The Transmitter Properties page.

applet and application channels. You can choose not to publish any files ending in `.java` or `.bak`, for example, to prevent the wholesale distribution of your source files (a good idea!). If you want to block certain directories from being published, enter the relative pathname followed by a slash. To prevent the distribution of the image directory of our sample channel, for example, enter `image\`. This blocks any files in the `image` directory, as well as any subdirectories. Make sure that you do not block the transfer of `*.txt` files, however. To successfully publish your channel, Publisher must transfer the `properties.txt` file (which contains all the properties you now are setting!).

Password — Another password? Yes, each Transmitter has a unique password. And each Channel also can have a unique password. If you are publishing your channel for the first time and want to add an extra layer of security, enter a password here. Then, each time you try to update the channel, this password is required.

18.2.4 General Properties

The General Properties page describes several of the most important characteristics of your channel (see Figure 18-2). Depending on what type

of channel you are creating (HTML, applet, presentation, or application), the settings can vary considerably.

For now, we will focus only on generic properties common to all channels and the specific properties associated with HTML channels:

Name	All channels must have a name. The name appears on the list of available channels distributed by the Transmitter. It also appears on the Channels page of the Castanet Tuner.
Type	You can choose from HTML, Applet, Presentation, or Application. For now, choose HTML.
Classpath	If your HTML channel includes any Java applets, and those applets require access to special Java packages (such as the Bongo.zip package for GUIs), enter them here. For most HTML channels, you can ignore this setting.
Index Page	The Index page is the first page retrieved by the Tuner when the user initially subscribes to the channel. If the Index page is **`index.html`**, this is considered an incremental HTML channel.

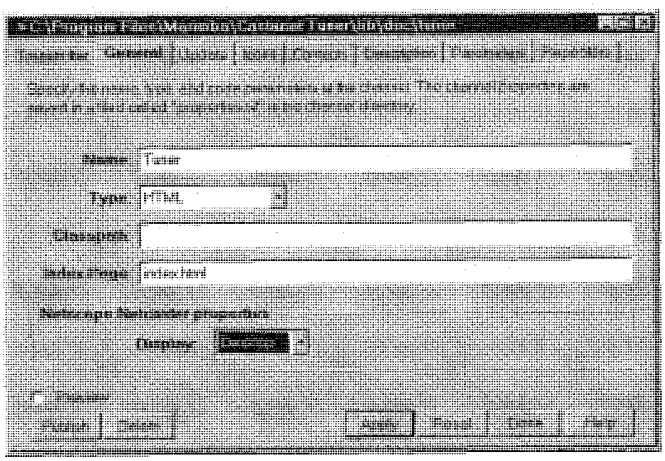


Figure 18-2
The General Properties page. HTML channel settings.

Netscape Netcaster Properties

If you are planning on distributing HTML channels to Netcaster subscribers (and you should be), there are several options to support Netscape Navigator Webtops. Webtops are full-screen Web pages attached to the bottom application layer of the user's operating system. This provides an always visible and fully immersive Web experience. There are three options for displaying your HTML channels in Netcaster:

Desktop	A Netscape Webtop.
Fullscreen	A full-screen Navigator window.
Window	A regular Navigator window.

18.2.5 Update Properties

Each channel follows its own update schedule, as shown on the Update Properties page (see Figure 18-3). And, as detailed in Chapter 16, section 16.4.1, "Automatic Channel Updates," each channel actually follows two update schedules. The *Active Update schedule* determines the frequency of updates while the channel is active or running on the user's desktop. The *Inactive Update schedule* determines the frequency of updates while the channel is *not* running.

HTML channels are never really considered active in the same sense as applet, presentation, and application channels. If you are creating an HTML channel, you therefore only need to set the inactive update frequency. This property determines the overall frequency of updates. The options are Never, Hourly, Daily, and Weekly. The user always can override this setting. (For now, disregard the Data Available Action option; we will return to it in section 18.4.4.)

18.2.6 Icon Properties

Each channel can have its own set of icons. Four types of icons are available on the Icon Properties page (see Figure 18-4):

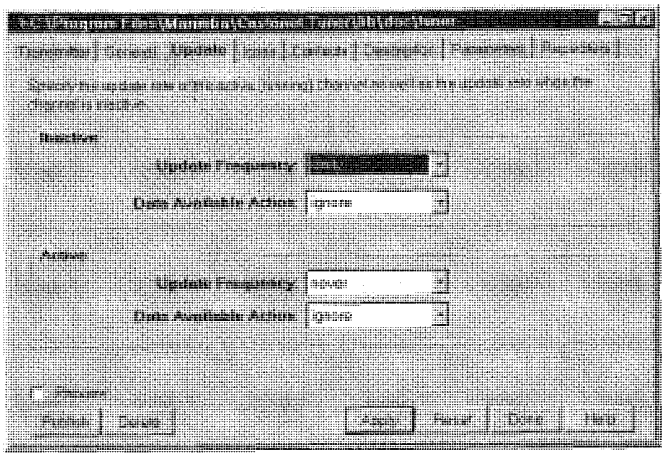


Figure 18-3
The Update Properties page.

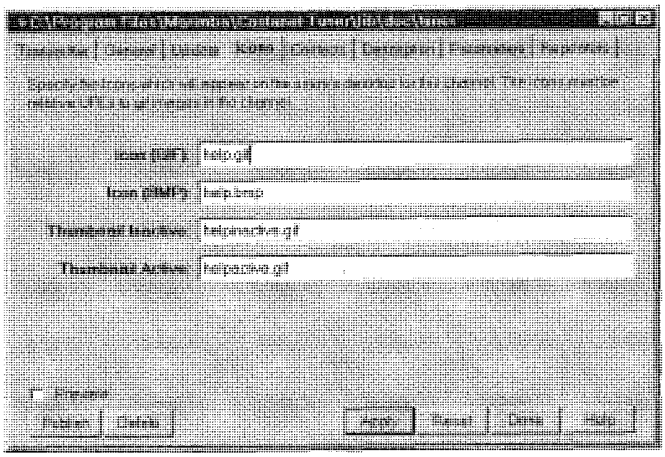


Figure 18-4
The Icon Properties page.

Icon (GIF)	A 64×64-bit desktop icon in GIF format. This is used on all platforms except Windows.
Icon (BMP)	A 64×64-bit desktop icon in Windows BMP format. This icon is used for desktop shortcuts in Windows 95/NT 4.0 (described in section 16.3.7, "Creating Desktop Shortcuts").

Thumbnail Inactive	A 16×16 GIF or JPEG image displayed on the Channels page of the Castanet Tuner. The Inactive icon appears if the user is subscribed to the channel but the channel currently is not running.
Thumbnail Active	A 16×16 GIF or JPEG image that indicates the channel is running.

Each of these icons must be placed inside your channel's base directory. Pathnames are relative to this base directory. If no icons are specified, the Tuner uses the default icons.

18.2.7 Contacts

If you want to specify an author or administrative contact for your channel, you can enter it here. This can be a full name or e-mail address. This information appears in the Channel Properties page of the Castanet Tuner.

18.2.8 Description

Your channel description goes here, as does any copyright information you want to enter. The channel description appears along with the Transmitter's list of available channels and on the Channel Description page of the Castanet Tuner.

18.2.9 Testing Your Channel

After you complete all your settings, click the Publish button to transfer your files to the Transmitter. You receive a confirmation of the process or a listing of specific errors. Then fire up your Tuner and try subscribing to your channel.

18.3 HTML Logging Capabilities

With the inclusion of a few simple HTML-like tags, you easily can add sophisticated logging capabilities to your HTML channels. These logging capabilities can track the links each user follows or even keep a running tab on which advertisements are displayed. Logging data is stored temporarily on the user's hard drive, and each time the channel is scheduled for its automatic update, the Tuner transfers the data directly to the Transmitter. This enables channel developers to track user behavior whether they are online or offline.

After data is transferred to the Transmitter, it is appended to the channel's log file or processed by a customized Castanet plug-in, if one exists.

Note that the end user always can opt to disable the logging and feedback functions through the Options page of the Castanet Tuner (or the Options dialog box provided by Netcaster).

18.3.1 The Log Tag

To track user activity, you simply add the Castanet `<?log=>` tag to your existing HTML documents. The Log tag always is preceded by a question mark (`?`) and usually is embedded inside an HTML Anchor `<A>` tag. To track user interest in your new products, for example, you could use this code:

```
<A HREF="?log=products/widget.htm">Check out our latest
      widget</A>
```

When the user follows the `widget` link, the event is recorded in the user's log file.

If you want to include more descriptive log entries, you can precede the URL address with a description. For example, you could write this:

```
<A HREF="?log=ClickedOnWidgets,products/widget.htm"><Check
      out our latest widget</A>
```

Don't forget the comma!

Finally, you also can include Log tags within Image tags. For example, the following line records that the widget ad was displayed to the user:

```
<IMG SRC="?log=WidgetAdDisplayed,images/widgetAd.gif">
```

18.3.2 The Channel Log File

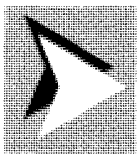
Each channel gets its own log file (by default, each log file is named after the channel and ends with a `.log` extension). All log files are located in the `logs` subdirectory of the Transmitter Channel directory (this is the directory you identified when you configured the Transmitter).

The following is the log file generated from the previous tags:

```
127.0.0.1 - w94vahgps34n [28/Aug/1997:15:05:42 -0400] "GET
     WidgetAdDisplayed HTTP/1.0" 200 0
127.0.0.1 - w94vahgps34n [28/Aug/1997:15:06:10 -0400] "GET
     WidgetAdDisplayed HTTP/1.0" 200 0
127.0.0.1 - w94vahgps34n [28/Aug/1997:15:06:16 -0400] "GET
     ClickedOnWidgets HTTP/1.0" 200 0
127.0.0.1 - w94vahgps34n [28/Aug/1997:15:06:18 -0400] "GET
     WidgetAdDisplayed @cx:HTTP/1.0" 200 0
```

The first column of the log file displays the IP address of the subscribed Tuner, and the second column identifies the unique Tuner ID. The third column records the date and time, and the last column records the actual event and protocol.

From this data, we can see that we had only one subscriber. But the subscriber saw our widget ad a total of three times and actually followed the link for additional information on the product.



TIP: *If you are just interested in processing user feedback like this, you might want to consider writing a simple PERL script that periodically parses the log file and dumps new data into a database. If you are a PERL wizard, this can be substantially easier than creating your own Castanet plug-in.*

18.4 Applet Channels

18.4.1 Sample Applet Channel

If you already have an existing Java applet, you easily can repackage it as a Castanet applet channel. And, 95 percent of the time, you can repackage your applet with little or no change to your existing source code.

To create an applet channel, first identify the Java applet you want to deliver. If you don't have one, you can use my Evolution applet on the CD-ROM accompanying this book. This is a relatively simple applet that demonstrates some of the very basic principles of genetic algorithms. It also demonstrates some very basic principles of Java GUI and layout development.

Before we publish the Evolution applet, let's consider changes to the source code.

18.4.2 Modifying Your Applet Source Code

There are only two instances when you must modify your applet source code. First, check to see whether your applet makes any calls to the browser's `showStatus` method. If your applet is running on a Web browser, the `showStatus` method displays a message in the browser's message or status line. But, if you run your applet on the Castanet Tuner, the message actually is displayed on the Tuner's Java console. Most users do not have this window open, though. Therefore, if the message is critical to the functioning of your program, consider delivering the message directly to the applet (perhaps use the `drawString` method or add your own Status text field). Otherwise, you safely can ignore the problem.

Second, check to see whether your applet makes any calls to the `getCodeBase()` or `getDocumentBase()` method. These calls generally are used for retrieving images or sounds or making live connections back to the server. If you are running your applet on a Web browser, either of these methods returns a URL of this form: `http://host:port/directory`. If you run the same applet on the Castanet Tuner, though, the URL is in this form: `tuner://host:port/directory`.

Each time a user subscribes to your channel, all the associated resource files, including images and sounds, are transferred over and stored on the user's hard drive. In fact, everything in your base channel directory is transferred over. The `tuner://` prefix therefore tells the Tuner to immediately retrieve the associated files from the hard drive instead of making additional network connections back to the Transmitter.

If your applet is using `getCodeBase` or `getDocumentBase` to subsequently retrieve files that exist in your base channel directory, no changes to the source code are necessary. But, if you are using either of these methods to retrieve other files or make live connections back to the Transmitter, you must modify your code. Specifically, you have to parse out the `tuner:\\` string and replace it with the `http:\\` string so that the Tuner knows to forward your request back to the Transmitter.

Fortunately, my Evolution applet does not make any calls to `showStatus`, and it doesn't make any live connections back to the server. So it's all ready to go and can be used without any modifications to the source code. To publish the channel, launch Castanet Publisher and follow the property guidelines in the next sections.

18.4.3 General Properties

To execute your Castanet applet, you need to provide runtime parameters similar to those provided by a Web browser. Most of the parameters on the General Properties page are related directly to the attributes associated with the HTML applet tag (see Figure 18-5).

The parameters on the General Properties page follow:

Type	Applet
Classpath	If your applet requires any special libraries, such as the `Bongo.zip` package, you can include them here.
Code	This is your main class file executed when the channel is launched. Simply include the name of the class file without the `.class` extension. This is identical to the `CODE` attribute of the HTML Applet tag.
Codebase	If your applet does not reside on the root level of your base directory, you can include a relative path here. This is identical to the `CODEBASE` attribute of the HTML Applet tag.

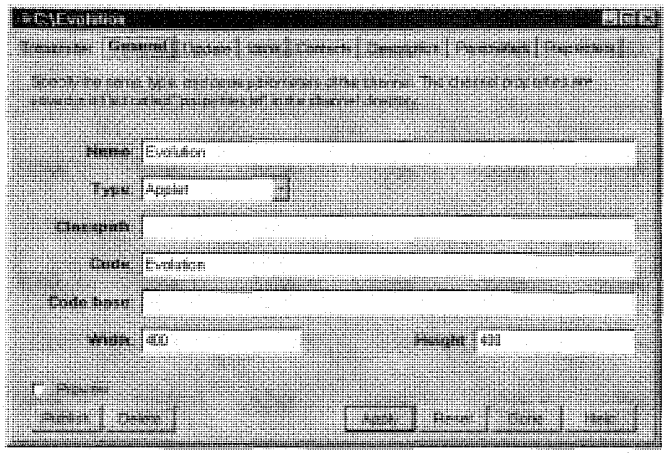


Figure 18-5
The General Properties page—applet channel settings.

Width	Width in pixels of your applet.
Height	Height in pixels of your applet.

18.4.4 Update Properties

In comparison to HTML channels, you have much more flexibility in setting the update properties of applet channels.

When your channel receives an automatic channel update, the Tuner can direct the channel to take several possible courses of action. This is known as the *data available action*. For *inactive* applications, three courses of action (or options) are possible:

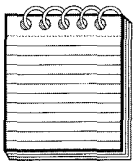
`ignore`	Installs the channel update to the user's hard drive and waits for the user to restart the channel. After the channel is restarted, it uses all the updated data and/or updated software components.
`start`	Installs the channel update to the user's hard drive and launches the applet, presentation, or application channel.
`notify`	Wakes up the inactive application, tells it that new data is available, and lets the application decide on a course of action. Primarily reserved for application channels. We will return to this option in the next chapter.

For *active* channels, three data available actions (or options) are possible:

ignore Installs the channel update to the user's hard drive. However, it lets the user continue to work with the current version of software. When the user exits and restarts the channel, the new data and/or software components are used.

restart Restarts the applet, presentation, or application channel.

install Primarily reserved for application channels. When new data arrives, the Tuner installs the data to the user's hard drive and notifies the application channel. The application channel then can retrieve and display the new information without restarting. This option is used primarily for real-time or live updates and is explored in greater detail in the next chapter.



NOTE: *Although the* **notify** *and* **install** *options are reserved primarily for application channels, it is possible to use these data available actions for applets. However, this requires that you significantly modify your existing source code. And I don't really consider it a useful option. If you really want your applets to take full advantage of the Castanet architecture, it is best to rewrite your entire code as an application channel (see Chapter 19).*

Table 18-1 presents a list of valid data available actions for each type of Castanet channel.

18.4.5 Applet Parameters

If your applet requires a set of parameters, you must enter them on the Parameters page (see Figure 18-6). Enter the parameters in **parameter=value** pairs, one per line. The Evolution applet, for example, requires you to set the number of columns and rows:

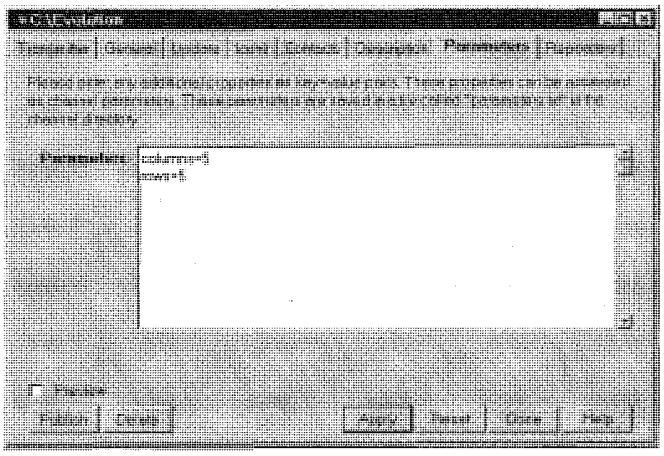
```
rows=5
columns=5
```

Table 18-1
Update Schedule Options

	HTML Channels	Applet Channels	Presentation Channels	Application Channels
Inactive schedule	never, hourly, daily, weekly	never, hourly, daily, weekly	never, hourly, daily, weekly	never, hourly, daily, weekly
Data Available Action options	ignore	ignore, start, notify	ignore, start	ignore, start, notify
Active Schedule	N/A*	never	never	never
		frequently	frequently	frequently
		5 minutes	5 minutes	5 minutes
		10 minutes	10 minutes	10 minutes
		15 minutes	15 minutes	15 minutes
		30 minutes	30 minutes	30 minutes
		45 minutes	45 minutes	45 minutes
		hourly	hourly	hourly
		daily	daily	daily
		weekly	weekly	weekly
Data Available Action options	N/A*	ignore, restart, install	ignore, restart	ignore, restart, install

*HTML channels are not considered active in the same sense as applet, presentation, and application channels. Therefore, there is no active schedule for HTML channels.

Figure 18-6
The Parameters page.



18.4.6 Testing Your Applet Channel

After you set all the channel properties, publish the channel. Then launch your Tuner and try subscribing to the applet. If all goes well, you should see my Evolution applet (see Figure 18-7).

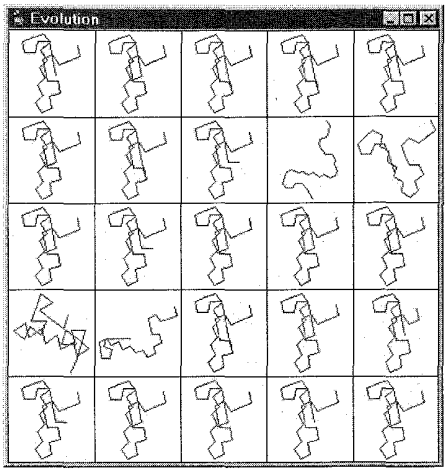


Figure 18-7
The Evolution applet—genetic algorithms in a box.

What the Evolution Applet Does

The Evolution applet is a spin-off of an idea proposed by the biologist Richard Dawkins. The applet begins with 25 squares. Each square contains a small picture of one or two lines. Click on a square, and the selected picture is copied over to the upper left corner of the applet. This picture then is mutated 24 times, and the offspring appear in the 24 remaining squares. Mutations could include the addition of new lines, the deletion of current lines, or the changing of line angles. Keep clicking on boxes, and you continue to create new generations of more complicated pictures. And, occasionally, you actually generate meaningful pictures (I once had a flower magically appear!).

Source: The Evolution applet was first proposed by Professor Jonathan Amsterdam, New York University.

18.5 Publisher Options

Now that you have had an opportunity to work with Castanet Publisher, you should be aware of several of its additional features. These additional features include previewing channel updates, publishing hidden channels, and executing Publisher from the command line.

18.5.1 Previewing Channel Updates

Each time you republish a channel, Publisher only sends differential updates to the Transmitter. If you want to determine which files will be transferred but do not want to actually transfer those files, you can preview the channel update. Just select the Preview option (located directly above the Publish button), and then publish your channel. Publisher displays its usual success/failure message with the list of updated files, followed by the message **`Publish Completed Successfully (no changes)`** (see Figure 18-8).

18.5.2 Hidden Channels

If you want to test your channel but do not want to advertise its existence, you can create a hidden channel. The Hidden option is located on the

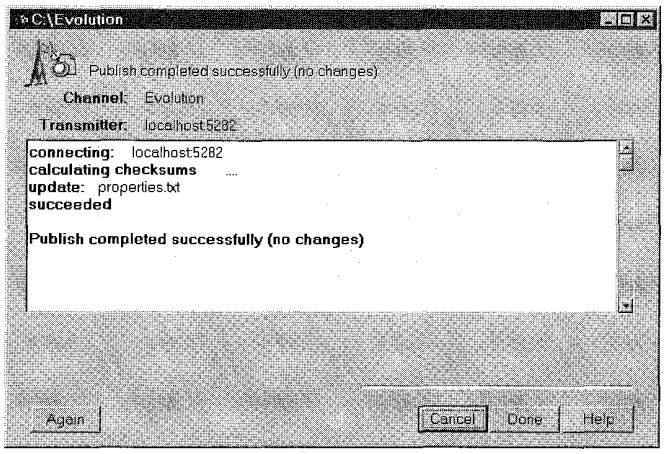


Figure 18-8
Previewing channel updates. Publisher lists the files to be transferred but does not actually transfer them.

Description Properties page of Castanet Publisher (see Figure 18-9). Hidden channels are not included in the list of available channels distributed by the Transmitter, but users and developers still can subscribe to the channel if they know its exact name. To subscribe, choose New Channel from the Tuner menu and enter the exact host and channel name (see Figure 18-10).

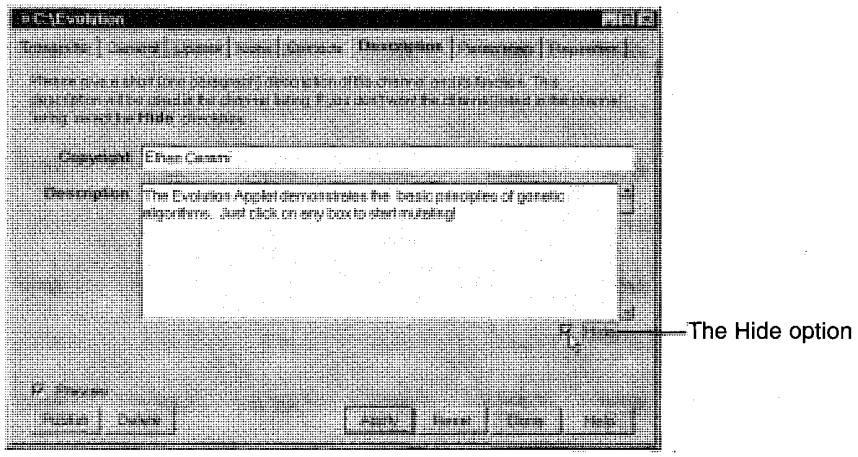


Figure 18-9
The Hide option is located on the Description Properties page.

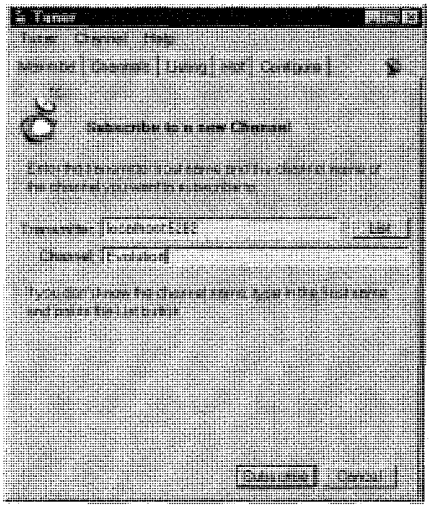


Figure 18-10
To subscribe to a hidden channel, choose New Channel from the Tuner menu and enter the exact channel name.

18.5.3 Running Publisher from the Command Line

In addition to using the Publisher GUI interface, you can use the command-line interface. This is useful particularly if you are creating an automated script that periodically adds new information to your channels.

The command-line executable **publish.exe** is located in the **commands** directory of Castanet Publisher. To use it correctly, you must specify the base directory (**-d** argument) of your channel:

```
c:\publish -d directory
```

As long as you use the **-d** option, Publisher automatically reads your channel's **properties.txt** file. This file contains all your channel settings, including the Transmitter name and password. For example, the following line publishes the Evolution applet (assuming that the channel resides in **c:\Evolution**):

```
C:\publish -d c:\Evolution
```

The command-line arguments for **publish.exe** follow:

-add ***destinationpath sourcepath***	Copies the files or directories specified in *sourcepath* to the *destinationpath* on the Transmitter.
-channel ***channelname***	Specifies the name of the channel to be published or deleted; an alternative to the **-d** argument.
-d ***basedirname***	Specifies the base directory of the channel to be published or deleted.
-delete	Deletes the channel.
-ignore ***suffix-list***	Does not copy these files over to the Transmitter.
-n	Lists the files that would be copied to the Transmitter but

	does not actually copy them. This is the same as the Preview option described in section 18.5.1.
`-q`	Specifies Quiet mode. Does not display the filenames as they are copied over.
`-password password`	Overrides the password specified in the channel's properties.txt file.
`-proxy hostname[:port][password]`	Specifies the Proxy Server to receive the channel update. Specifies the password if one is required.
`-transmitter` *`hostname[:port]`*	Overrides the name and port number of the Transmitter specified in the channel's `properties.txt` file.

SUMMARY

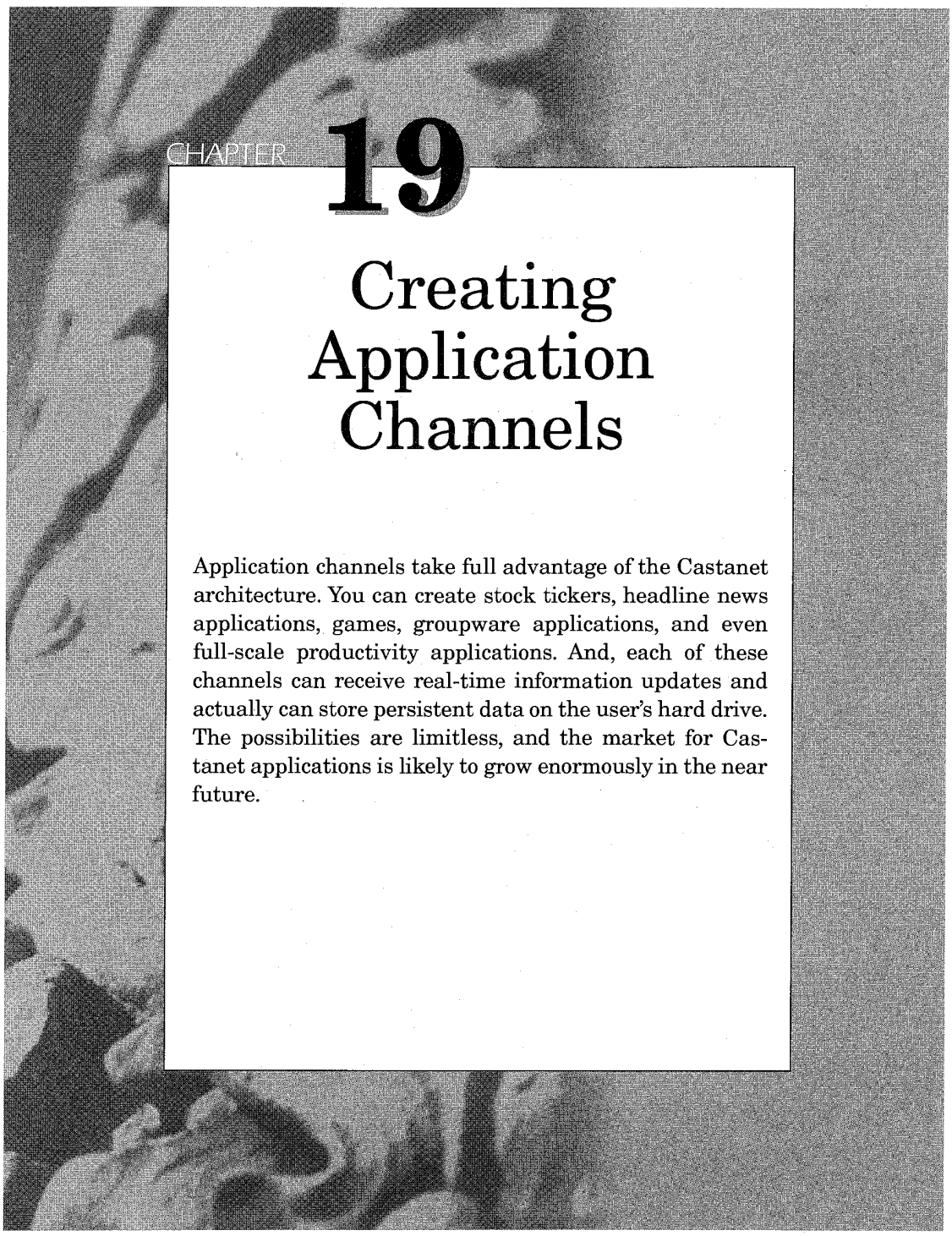
HTML channels are collections or directories of HTML documents. The HTML documents can, in turn, contain dynamic HTML, JavaScript, Java applets, or ActiveX controls. To create an HTML channel, first create your HTML pages. Then publish them to the Transmitter via Castanet Publisher. Castanet enables sophisticated logging capabilities for tracking user behavior online and offline.

Applet channels are regular Java applets that have been repackaged as channels. Most of the time, you easily can create applet channels with little or no changes to your existing code.

Each channel follows an Active Update schedule and an Inactive Update schedule. Each time a Tuner receives a channel update, it can follow one of several courses of action known as the data available action. Different schedules and actions are available for each type of Castanet channel. Consult Table 18-1 for an overview of the differences.

Online Resource

- From Applet To Channel, Step by Step
  `http://www.marimba.com/doc/Castanet_Developer_Docs/current/index.html`. Quick tutorial on repackaging your legacy applets as Castanet applet channels.



CHAPTER 19

Creating Application Channels

Application channels take full advantage of the Castanet architecture. You can create stock tickers, headline news applications, games, groupware applications, and even full-scale productivity applications. And, each of these channels can receive real-time information updates and actually can store persistent data on the user's hard drive. The possibilities are limitless, and the market for Castanet applications is likely to grow enormously in the near future.

This chapter explores the steps necessary for creating and deploying application channels. We will begin with a series of very simple sample applications. The goal of these examples is to present key concepts in their most elementary forms, enabling you to quickly grasp new concepts. We then will move on to more complicated sample applications and finally end the chapter with a full-scale application channel capable of delivering real-time news headlines to a scrolling screen ticker.

19.1 Getting Started

19.1.1 Prerequisites

To build application channels, you must know Java. Specifically, you should be familiar with a few key concepts:

- The general principles of Java's object-oriented facilities, including inheritance and method overriding
- The concept of interfaces and the particulars of implementing them
- The steps necessary for building applets

If you do not know Java, or if any of these concepts seem unfamiliar to you, you should take the time now to learn them. The stronger your knowledge of Java, the faster you will understand the concepts presented in this chapter.

19.1.2 Working with the JDK

To compile your Java application channels, you need a Java 1.0 or 1.1 compiler. You can use any of the commercially available compilers, including Microsoft Visual J++ or Symantec Café. Or, you can use the freely available *Java Development Kit* (JDK) from Sun Microsystems.

19.1.3 Setting the CLASSPATH Variable

To compile properly, application channels must have access to the Marimba API. The API is contained in a series of compressed zip files (do not unzip them!) that were installed during the installation of your Castanet Transmitter.

To indicate the location of the Marimba API files, you must set your **CLASSPATH** variable. When your program is compiled, the compiler searches the **CLASSPATH** directory for any additional packages or libraries. If the compiler fails to find the Marimba API, it returns with a series of errors.

The current version of the Castanet Transmitter installs the Marimba API files in the **Castanet Publisher** directory. If you followed the typical installation, the files will be located in this directory:

```
C:\Program Files\Marimba\Castanet Publisher\developers\
    compiling
```

Here, you see three zip files:

channel.zip	The Marimba API for creating application channels. This includes the application interface, the Marimba FastIO package, plus a number of miscellaneous utilities.
bongo.zip	The Marimba API for creating Bongo presentations and applications.
plugin.zip	The Marimba API for creating Castanet plug-ins.

Depending on your programming needs, you need to add any one or all of these zip files to your **class** path. The examples presented in this chapter only require access to the **channel.zip** file, though.

If you are using a commercially available compiler, check the documentation on setting the **CLASSPATH** variable. If you are using the JDK on Windows 95/NT, you need to set the **CLASSPATH** variable in your **autoexec.bat** file, which is located on drive C.

To set the **CLASSPATH** variable, type **SET CLASSPATH=** followed by an absolute pathname. Separate each zip file with a semicolon.

Here, for example, is my **CLASSPATH** variable.

```
SET CLASSPATH=C:\Program Files\Marimba\Castanet
    Publisher\developers\compiling\channel.zip;C:\
    Program Files\Marimba\Castanet Publisher\developers\
    compiling \bongo.zip
```

19.2 The Application Interface

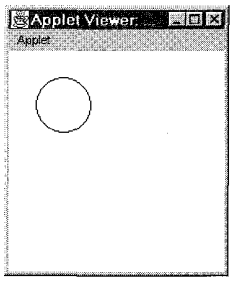
19.2.1 Applets versus Application Channels

Creating application channels is enormously similar to creating applets. To create a Java applet, you first must extend the Applet class. You then can override any of the applet methods, including **init**, **start**, and **stop**. Listing 19-1, for example, is a simple applet that draws a circle to the screen and overrides the **init**, **start**, and **stop** methods (see Figure 19-1).

Listing 19-1
A sample applet.

```
/* <applet code="Circle" width=200 height=200></applet>*/

import java.awt.*;
import java.applet.*;

public class Circle extends Applet {

    public void init () {
        System.out.println ("Initializing...");
    }

    public void start () {
        System.out.println ("Starting...");
    }

    public void stop () {
        System.out.println ("Stopping...");
    }

    public void paint (Graphics g) {
        g.drawOval (25,25,50,50);
    }
}
```

Figure 19-1
A sample applet.



In contrast, to create an application channel, you extend a class provided by the Marimba API. Creating application channels therefore is as easy as creating applets, and the more you know about creating applets, the better off you are.

19.2.2 The Application Interface

So, which Marimba class do you extend to create a channel? Not so fast.

The Marimba API first includes an *interface* for creating application channels. An interface represents a broad skeletal outline of what a class should do but gives no hint whatsoever on implementation. I like to think of interfaces as the ideal boss. Your boss tells you that you must succeed in achieving a specific set of goals: "I don't care how you get it done, just get it done!" You therefore have complete flexibility with your time and resources, and you can do whatever you want as long as you get the job done.

Interfaces work the same way. An interface tells a class that it *must* implement a series of methods. But the interface does not tell the class *how* to implement those methods.

The application interface provided by Marimba requires that you implement four specific methods (see Listing 19-2):

`setContext()`	When your application is launched, the Castanet Tuner calls this method first. `setContext` gives your application a sense of its runtime environment. You can think of this as the equivalent of a channel passport: It tells you application who it is and where it is living, and it outlines its special privileges and responsibilities.
`start()`	Next, the Tuner calls the `start()` method. The `start()` method includes all your program initialization and is called only once. It therefore is similar to the `init()` applet method.
`handleEvent()`	`handleEvent` is used to process channel updates. When you receive a channel update, for example, you can choose to restart the application or immediately install and display the new information.
`stop()`	Clean up your program and get ready to exit. Use this method to close any open connections or stop any running threads. This is equivalent to the `stop()` applet method.

Listing 19-2
The Application Interface

```
public interface Application {

    //  Give the channel a sense of its runtime
    //  environment
    void setContext (ApplicationContext context);

    //  Initialize the Channel
    void start();

    //  Handle Channel Updates
    boolean handleEvent (Event evt);

    //  Stop the channel:  perform any clean-up
    void stop();

}
```

19.2.3 Hello, World!

No programming example would be complete without a Hello, World! application (and this chapter has several of them). Listing 19-3 shows a sample application that implements the application interface and writes `Hello, World!` to the Java console of the Castanet Tuner.

Here are a few details to note:

- In line 4, we import the `marimba.channel` package. This enables us to access the application interface.
- We have implemented all four methods required by the application interface.
- The `setContext`, `start`, and `stop` methods display short, informative messages. All calls to `System.out.println` are displayed in the Java console of the Castanet Tuner.

19.2.4 Publishing the Hello World Example

If you can successfully compile the Hello World example, your `CLASSPATH` variable is set correctly and you will be able to compile all the examples in this chapter. Otherwise, check section 19.1.3, "Setting the `CLASSPATH` Variable," again or consult the online documentation accompanying your compiler (the JDK, for example, includes a README file with useful information regarding `CLASSPATH` variables).

Listing 19-3
Hello, World!
Version 1.0

```
1   //     Example Application Channel: HelloWorld!
2
3   import java.awt.*;
4   import marimba.channel.*;
5
6   public class HelloWorld implements Application {
7
8         //    Set Application Context
9         public void setContext (ApplicationContext ctx) {
10              System.out.println ("Setting Context...");
11        }
12
13        //  Initialize Channel
14        public void start () {
15              System.out.println ("Hello World!");
16        }
17
18        //  Stop Channel
19        public void stop () {
20              System.out.println ("Stopping...");
21        }
22
23        //  Handle channel updates
24        public boolean handleEvent (Event evt) {
25              return true;
26        }
27  }
```

Unfortunately, you cannot run your example from the command line. To run your program, you first need to publish it to the Castanet Transmitter and then use your Castanet Tuner to subscribe to the channel.

To publish your channel, launch Castanet Publisher and follow the same steps required for publishing applet or HTML channels. In particular, note the following:

- Remember to include the full absolute path to the **`HelloWorld`** directory.
- On the Transmitter page, make sure to enter the correct hostname, port number, and any password, if required.
- On the General page, set the type to **`Application`**. And, in the **`Main`** class, enter **`HelloWorld`**. The **`Main`** class must implement the application interface and always is launched first.

After you successfully publish the channel, use your Castanet Tuner to subscribe to the channel. Then open the Java console window and launch the channel (see Figure 19-2).

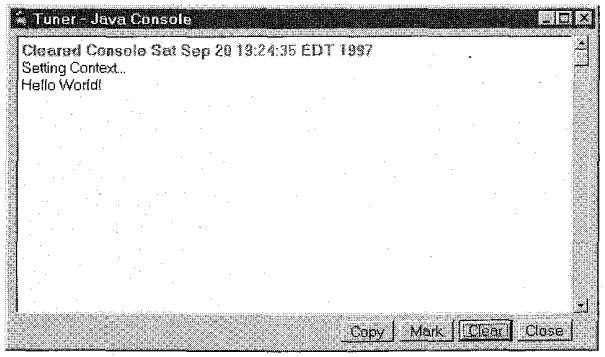


Figure 19-2
Hello, World!
Version 1.0.

19.2.5 The Channel Development Process

Now that you have your first channel up and running, let's take a moment to discuss the channel-development process. When compiling standalone Java applications or Java applets, you always have the option of running your program from the command line. To run standalone applications, you can use the **java** command, for example, and for applets, you always can use the JDK applet viewer. Unfortunately, no such options exist for channel applications.

This results in a longer, more complicated debugging process. Here, in a nutshell, are the steps required for developing, testing, and debugging channels:

1. Write you application channel and compile it.
2. Publish the channel to the Transmitter.
3. Subscribe to the channel with the Castanet Tuner.
4. Run the channel and test it. Check for bugs.
5. Edit you application channel and recompile it.
6. Republish the channel to the Transmitter.
7. Using the Castanet Tuner, manually update the channel.
8. Go back to step 4 and repeat.

Unfortunately, this can become a rather arduous and tiresome process. But it currently is the only method of testing and debugging channels. If you are developing a rather sophisticated application channel, your best option is to first create a standalone Java application. You then can run

and debug the application from the command line. Once the application is working, you then can rework your application into an application channel. Then continue debugging.

The final example in this chapter is a scrolling ticker channel application. To ease the channel development, I first created a standalone Java application that displays a single scrolling line of text. I therefore was easily able to debug the graphical layout and the scrolling animation simply by running it from the command line. Once I had the animation in working order, I transformed the application into an application channel by implementing the application interface. Breaking your channel development into these two distinct phases can save you lots of time and aggravation.

19.2.6 Hello, World! Version 2.0

Before we proceed any further, let us modify our first example application so that it is slightly more useful. This time, instead of displaying to the Java console, we'll create a small frame window and display `Hello, World!` in it (see Figure 19-3). Listing 19-4 extends the `Frame` class provided by the Java *Abstract Windowing Toolkit* (AWT) and implements the application interface.

Here are the main highlights of the example in Listing 19-4:

- The `setContext` method stores a copy of the `ApplicationContext`. The `ApplicationContext` gives the channel access to the Castanet Tuner and is explored in detail in later sections. To take full advantage of the Castanet architecture, you always should store a local copy of the `ApplicationContext`.
- The start method sets the frame size, adds a single label, and displays the frame window.

19.3 The `ApplicationFrame` Class

19.3.1 The `ApplicationFrame` Class Defined

You can choose to create your applications from scratch by implementing the application interface, or you can choose to use Marimba's `ApplicationFrame`

Listing 19-4
Hello, World!
Version 2.0

```
1   //    Example Application Channel: HelloWorld!,
    //    Version 2.0
2   //    Extends Frame, Implements Application
3
4   import java.awt.*;
5   import marimba.channel.*;
6
7   public class HelloWorld extends Frame implements
    Application {
8       private ApplicationContext context;
9
10      //   Set Application Context
11      public void setContext (ApplicationContext
        context) {
12          context = context;
13          System.out.println ("Setting Context....");
14      }
15
16      //  Initialize Channel
17      public void start () {
18          this.setSize (200,100);
19          Font font = new Font ("Helvetica",
            Font.BOLD, 20);
20          Label msg = new Label ("Hello World!");
21          msg.setFont (font);
22          msg.setAlignment (Label.CENTER);
23          this.add (msg);
24          this.show();
25      }
26
27      //  Stop Channel
28      public void stop () {
29          System.out.println ("Stopping...");
30      }
31
32      //  Handle channel updates
33      public boolean handleEvent (Event evt) {
34          return true;
35      }
36  }
```

class. The **ApplicationFrame** class provided by Marimba extends the **Frame** class provided by the AWT and implements the application interface with a number of default implementations. It therefore is very similar to our example in Listing 19-4.

The **ApplicationFrame** class makes programming considerably easier and makes application channels as easy as applets. Listing 19-5 contains an outline of the **ApplicationFrame** class.

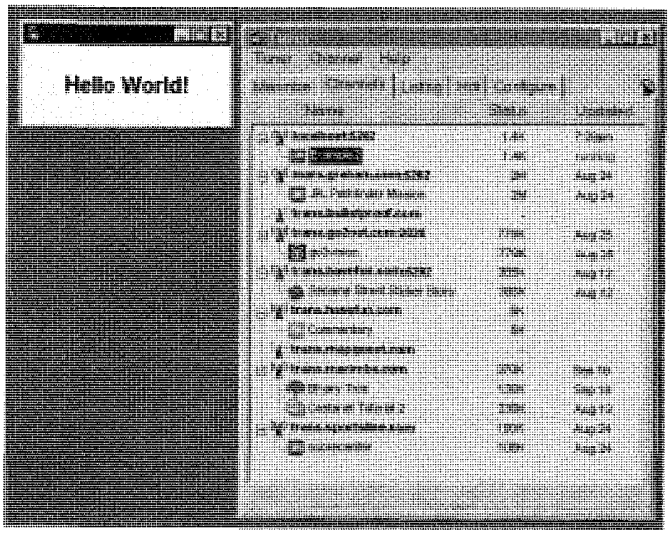
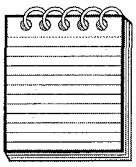


Figure 19-3
Hello, World!
Version 2.0.

19.3.2 Hello, World! Version 3.0

Here then is our final Hello, World! example. This time, we extend the `ApplicationFrame` and override the `start()` method, as shown in Listing 19-6.

This is more typical of the application channels you will create. And, you now can see that application channels are extremely similar to applets. Therein lies the true simplicity of Castanet.



NOTE: *If you are creating channels that use the Bongo software library, Marimba also provides the* `ApplicationPlayerFrame` *class. Extend this class instead of* `ApplicationFrame` *if your application uses of any Bongo GUI elements or presentations.*

19.4 The Application Context

19.4.1 The Application Context Defined

We have now alluded several times to the *application context*. The application context is very similar to the applet context normally associated with regular Java applets.

Listing 19-5
An Outline of the **ApplicationFrame** Class

```
package marimba.channel;

import java.awt.*;

public class ApplicationFrame extends Frame implements
      Application {

      protected ApplicationContext context;

      //  Save the ApplicationContext
      public void setContext(ApplicationContext context) {
            this.context = context;
      }

      //  Show the Frame
      public void start() {
            show();
      }

      //  Destroy the channel, hide and dispose the frame
      public void stop() {
            hide();
            dispose();
      }

      //  Handle Channel Updates
      public boolean handleEvent(Event evt) {
      ...
      ...
      }
}
```

The *applet context* enables the applet to communicate with the Web browser that is hosting it. You can use the applet context to retrieve images or audio files, for example, to display messages in the browser's status box (**showStatus**), or to open a specified URL page (**showDocument**).

In much the same manner, the application context enables the application channel to communicate with the Castanet Tuner that is hosting it. In contrast to Web browsers, though, the Castanet Tuner enables much greater functionality. You can use the **ApplicationContext** to read or write from a specified directory on the user's hard drive, for example, or you can use it to direct automatic updates and installations.

Listing 19-6
Hello, World!
Version 3.0

```
1   //    Example Application Channel: HelloWorld!, Version 3.0
2   //    Extends ApplicationFrame
3
4   import java.awt.*;
5   import marimba.channel.*;
6
7   public class HelloWorld extends ApplicationFrame {
8
9         //  Initialize Channel
10        public void start () {
11              this.setSize (200,100);
12              Font font = new Font ("Helvetica",
                Font.BOLD, 20);
13              Label msg = new Label ("Hello World!");
14              msg.setFont (font);
15              msg.setAlignment (Label.CENTER);
16              this.add (msg);
17              this.show();
18        }
19  }
```

19.4.2 Application Context Methods

The application context includes a great number of public methods that enable access to the Castanet Tuner. Here, for your reference, is a listing of the most important methods. We will return to the specifics of some of these methods in sections 19.5 and 19.6 when we discuss the storing of persistent data. For a complete listing of the **`ApplicationContext`** API, consult Appendix A, "Marimba Castanet API."

`public URL getBase()` Gets the base URL of this channel. Used primarily to retrieve image and data files.

`public String getChannelName()` Gets the name of this channel.

`public String getChannelStatus (String channelName)` Gets the status of a channel on this Transmitter. The status will be **`unsubscribed`**, **`subscribed`**, or **`running`**.

`public URL getDataBase()` Gets the URL for the data directory of this channel. This is the only directory where the channel can store persistent data.

`public String getDataDirectory()` Gets the data directory string of this channel. This is the only directory where the channel can store persistent data.

`public String getParameter()` Gets an application parameter. The application parameters are specified in the `properties.txt` file associated with each channel. Parameters are case-insensitive.

`public Updates getPendingUpdates()` Gets the update report for this channel. Returns `null` if there are no existing updates of any kind.

`public String getServerName()` Gets the name of the server of this channel. The server name is of the form `hostname:port`.

`public void installData (String dir)` Requests the installation of all or part of new channel data. Pass an empty string to install the entire channel.

`public long publishTime()` Returns the time at which this version of the channel was published.

`public void restart()` Restarts the application.

`public void update()` Requests a channel update.

`public long updateTime()` Returns the time the channel was last updated with new data.

19.4.3 Example Application

The application context is extremely simple to use. Listing 19-7 displays runtime information concerning the channel and its host Tuner (see Figure 19-4).

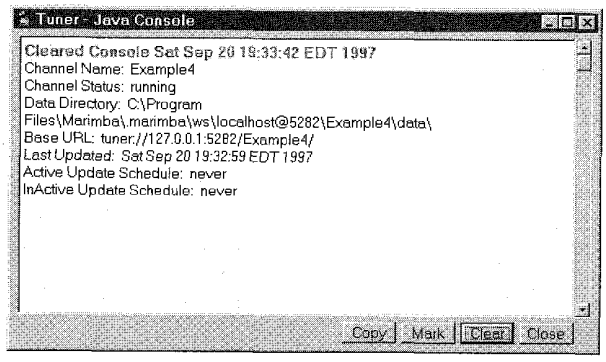
This example refers to several important methods that we will revisit in the next two sections. First, note the `getDataDirectory()` method. This method returns an absolute path to the channel's data directory. With the modified Castanet security sandbox, channels can read from and write to this directory only. All other directories are strictly off limits.

Second, the `getBase()` method returns a URL to the channel's base directory. You can use this URL to retrieve image or data files that were downloaded along with your Java class files. This is the equivalent of the applet `getDocumentBase()` method, except that the Tuner retrieves these

Listing 19-7
Example Application Context

```
//     Basic Application Channel
//     Extends ApplicationFrame
//     Displays ApplicationContext Data to the Java Console

import java.awt.*;
import java.util.*;
import marimba.channel.*;

public class BasicChannel extends ApplicationFrame {

  //  Initialize Channel
  public void start () {
     String name = context.getChannelName();
     System.out.println ("Channel Name:  "+name);
     System.out.println ("Channel Status:"+context.
          getChannelStatus(name));
     System.out.println ("Data Directory:
          "+context.getDataDirectory());
     System.out.println ("Base URL:  "+context.getBase());
     System.out.println ("Last Updated:
          "+new Date(context.updateTime()));
     System.out.println ("Active Update Schedule:
          "+context.getParameter
            ("update.active"));
     System.out.println ("InActive Update Schedule:
          "+context.getParameter
          ("update.inactive"));
     }
}
```

Figure 19-4
Output: Using the application context.



files from the user's hard drive and does not actually make a connection back to the Transmitter.

Finally, note the use of `getParameter`. This is the equivalent of the `getParameter` method normally associated with applets. But this `getParameter` enables you to access any channel settings. These channel settings reside in the `properties.txt` file that is generated when you publish your channel. Here, we query the properties file for the Active and Inactive Update schedules.

19.5 Storing Persistent Data

One of the great advantages of Castanet application channels is the capability to store persistent data to the user's hard drive. This enables you to create full-scale applications that can save state between executions. And, not only does Marimba provide this extra functionality, but it also includes a series of classes for easier and faster I/O access.

19.5.1 The Marimba Fast I/O Classes

The Marimba library includes two classes for fast input and output. The `FastInputStream` class enables you to read data. The `FastOutputStream` class enables you to write data. Both these classes are similar to the `BufferedInputStream` provided by the Java API, but they provide faster I/O access and actually are considerably easier to use.

An overview of the Fast I/O API follows. For the complete API, consult Appendix A.

Before we proceed in using the Marimba fast I/O classes, however, let us clarify a potential source of confusion.

19.5.2 The Base Channel Directory versus the Data Directory

Each time you subscribe to a channel, the Castanet Tuner downloads an entire directory of contents. From now on, we will refer to this directory as the *base channel directory* or simply the *base directory*. This base

FastInputStream

Constructors

`FastInputStream(byte[])`	Creates a stream that reads from a buffer.
`FastInputStream(byte[], int, int)`	Creates a stream that reads from a buffer.
`FastInputStream(File)`	Opens a file.
`FastInputStream(InputStream)`	Combines two streams.
`FastInputStream(InputStream, int)`	Combines two streams given a buffer size.
`FastInputStream(RandomAccessFile)`	Accesses a `RandomAccessFile`.
`FastInputStream(String)`	Opens a file.

Main Methods

`public void close() throws IOException`	Closes `FastInputStream`.
`public boolean getError()`	Returns `True` if there has been a read error. I/O errors are not reported during reads. Instead, they are reported when the stream is closed.
`public int read(byte[], int, int)`	
`public boolean readBoolean()`	
`public byte readByte()`	
`public char readChar()`	
`public double readDouble()`	
`public float readFloat()`	
`public void readFully(byte[])`	
`public void readFully(byte[], int, int)`	
`public int readInt()`	
`public String readLine()`	

(Continues)

```
public long readLong()
public short readShort()
public int readUnsignedByte()
public int readUnsignedShort()
public String readUTF()
```

FastOutputStream

Constructors

`FastOutputStream(byte[])`	Creates a memory output stream with a given size.
`FastOutputStream(File)`	Opens a file for writing.
`FastOutputStream(int)`	Creates a memory output stream with a given size.
`FastOutputStream(OutputStream)`	Combines two streams.
`FastOutputStream(OutputStream, int)`	Combines two streams given a buffer size.
`FastOutputStream(RandomAccessFile)`	Opens using a `RandomAccessFile`.
`FastOutputStream(String)`	Opens a file for writing.

Main Methods

`public void close() throws IOException`	Closes `FastOutputStream`.
`public boolean getError()`	Returns `True` if there has been a write error. I/O errors are not reported during writes. Instead, they are reported when you flush or close the stream.

```
public final void print(char)
public final void print(long)
public final void print(String)
```

```
public final void println()
public final void println(char)
public final void println(long)
public final void println(String)
public final void write(byte[], int, int)
public final void write(int)
public final void writeBoolean(boolean)
public final void writeByte(int)
writeBytes(String)
writeChar(int)
writeChars(String)
writeDouble(double)
writeFloat(float)
writeInt(int)
writeLong(long)
writeShort(int)
writeUTF(String)
```

directory includes everything needed to make your application run, including all of your Java class files, image files, sound clips, and various data files. When your channel is updated with new information, the new information also is installed here.

Your application channel can retrieve or read any of these files. But, when you direct the Tuner to retrieve these files, it simply retrieves them from the user's hard drive and does not need to make any network connections back to the Transmitter. These files therefore represent one set of files the application can retrieve from the user's hard drive. But remember that your application can only read these files. It cannot modify them in any way. And it cannot create new files in the base directory.

In contrast to the base channel directory, each channel also gets a data directory. This is where you store persistent data. Your application has privileges to read, modify, or create new files in the data directory. The data directory is the second method by which you access data from the user's hard drive.

Even though both the base channel directory and the data directory exist on the same hard drive, they are accessed in different ways. Understanding these differences is crucial.

19.5.3 Reading Files from the Base Channel Directory

To retrieve a file from the base channel directory, you *must* access it with the URL received from the application context `getBase()` method. The following code, for example, opens the `data.txt` file in the base directory and reads one line of data:

```
URL datafile = new URL(context.getBase(), "data.txt");
FastInputStream in = new FastInputStream
      (datafile.openStream());
String text = in.readLine();
```

Here, you can see that we are using the `FastInputStream` and the `readLine` method to read a single line of text from the data file.

19.5.4 Reading from and Writing to the Data Directory

To read from or write to a file in the data directory, you can use the URL provided by the application context `getDataBase` method or the file path provided by the `getDataDirectory` method. Here, for example, we read the `hiscore.dat` file and store the high score in memory:

```
File datafile = new File (context.getDataDirectory(),
      "hiscore.dat");
FastInputStream in = new FastInputStream (datafile);
int hiscore = in.readInt();
```

If the user ends up beating the high score, we simply use `FastOutputStream` to write the new high score back out to the `hiscore.dat` file.

19.5.5 Example Application

To further illustrate these points, the example in Listing 19-8 uses both the base channel directory and the data directory. When you publish this

channel, make sure to include a **data.txt** file in your base channel directory. The file should include a single line of text or a single sentence.

The application first opens the **data.txt** file from the channel base directory and extracts the single sentence. It then takes this sentence and stores it on the user's hard drive in a file called **storage.txt**. The application also prints the full pathname of the data directory so that you can manually verify that the **storage.txt** file exists and contains the correct message.

It is important to note that your application must be prepared to catch the **IOException**. This is standard for all of Java's I/O classes. If you are creating a new URL to access the base directory, you also must remember to catch the **MalformedURLException**. The exception handling in this code simply prints a stack trace to the Java console.

Listing 19-8
Handling Persistent Data

```
//    Basic Application Channel
//    Extends ApplicationFrame
//    Reads Data file from the Channel's Base URL
//    Then Writes Data out to local data directory

import java.awt.*;
import java.io.*;
import java.net.*;
import marimba.channel.*;
import marimba.io.*;

public class BasicChannel extends ApplicationFrame {

   //  Initialize Channel
   public void start () {
      String text = read_url("data.txt");
      write_file (text, "storage.txt");
   }

   //  To read files in the channel base directory,
   //  you must use URL addresses formed with
   //  context.getBase()
   private String read_url (String filename) {
      String text = null;
      try {
         System.out.println ("BaseURL:  "+context.
         getBase());
         URL data = new URL(context.getBase(), filename);
         FastInputStream in = new FastInputStream
         (data.openStream());
         text = in.readLine();
         System.out.println ("Retrieving Text:  "+text);
         in.close();
      }
```

(Continues)

Listing 19-8
Continued

```
        catch (MalformedURLException e) {
            e.printStackTrace();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
        finally {
            return text;
        }
    }

    //  Write to file in data directory
    private void write_file (String text, String filename) {
        try {
            System.out.println ("Data Directory:
            "+context.getDataDirectory());
            File datafile = new File (context.getDataDirec-
            tory(),filename);
            FastOutputStream out = new FastOutputStream
            (datafile);
            System.out.println ("Writing out text:  "+text);
            out.writeChars (text);
            out.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

19.6 Handling Updates

We now are edging closer to creating full application channels. Our next hurdle is understanding Castanet's channel-update mechanisms.

19.6.1 The Update Events

Java is an event-driven programming language. At any point in time, it could handle any number of events. A user might click a button, an animation timer might go off, or a user simply might hold down the Shift key. To process any of these events, you must include specific event handlers in your code.

The Castanet Tuner generates its own events to signal the arrival of new data or new channel updates. In fact, it generates four specific channel-update events. Understanding the differences between these events is crucial to creating dynamic application channels.

As much as possible, the Castanet Tuner attempts to follow the update schedule recommended by the channel provider. But the exact steps followed by the Tuner during channel updates depends largely on whether the application being updating is active or inactive.

19.6.2 Handling Active Channel Updates

Let's first examine what happens when the application being updated is active and running on the user's desktop.

Each time the Tuner initiates a channel update, it first sends a warning signal, known as the **DATA_UPDATE_EVENT**:

DATA_UPDATE_EVENT tells the currently active application that an update is *about* to occur. Most commonly, you will want to catch this event only if you are creating sophisticated Castanet plug-ins. You can catch the event, for example, and direct the application to immediately save any logging or profile data so that the information is automatically relayed back to the Castanet Transmitter. If you are not creating any plug-ins, you can safely ignore this event.

The Tuner then proceeds to check the Transmitter for new information. If there is no new information, the Tuner goes back to sleep and waits for the next update interval. But, if there is new data, the Tuner stores the updated channel files to a temporary directory and sends a **DATA_AVAILABLE_EVENT**.

DATA_AVAILABLE_EVENT tells the currently active application that a new channel update has arrived. The Tuner awaits your instructions on how to proceed.

When you receive a **DATA_AVAILABLE_EVENT**, your application has three basic options:

1. You simply can ignore the event. The next time the user launches the application, the Tuner installs and uses the newly updated channel. But, for now, it sticks with the currently running version.
2. You can restart your application by calling the application context `restart()` method (see section 19.4.2, "Application Context Methods").

This directs the Tuner to stop the channel, install the channel update, and then restart the channel. If you are planning to follow this option, it is usually a good idea to give the user the option or at least some warning.

3. You can keep your channel running but install the new data by calling the application context `installData()` method. This directs the Tuner to install the new data to the base channel directory, where your application can access the new information. This is the preferred method of handling dynamic updates.

If you choose the third option, the Tuner installs the channel update and sends one last event: **DATA_INSTALLED_EVENT**.

DATA_INSTALLED_EVENT tells the currently active application that the new channel update has been installed successfully. After the application is installed, it can access the new channel data.

Handling Active Channel Updates

1. The Tuner sends a **DATA_UPDATE_EVENT**. This tells the application that an update is about to occur. Direct your application to save any profile or logging information.
2. The Tuner checks the Transmitter for new information. If no new information is available, the Tuner goes back to sleep and waits until the next update interval.
3. If new information is available, the Tuner sends the **DATA_AVAILABLE_EVENT**. This tells the application that new information is available. The Tuner waits for your application's instructions.

   An active application channel has three basic options for handling a **DATA_AVAILABLE_EVENT**:

   Ignore it. Use the updated channel the next time around.

   Restart the application. Call `ctx.restart()`.

   Keep the application running and install the data now. Call `ctx.installData()`.

4. If the application follows the third option, the Tuner installs the new channel update. When the update is installed successfully, the Tuner sends the **DATA_INSTALLED_EVENT**.

19.6.3 Handling Inactive Channel Updates

If the application channel being updated is inactive, the Tuner follows a different course of action.

First, because the application is inactive, there is no need to give it advanced warning and therefore no need to send the `DATA_UPDATE_EVENT`. Second, the Tuner checks the Transmitter. If new information is available, the Tuner installs the data to the user's hard drive. It then wakes up a small part of the application channel (the Tuner basically launches just enough of the application to call its event-handler routines, but it refrains from calling the application's `start()` method). The Tuner then sends a `DATA_NOTIFY_EVENT`.

`DATA_NOTIFY_EVENT` tells the application that a new channel update has arrived and has been installed. If the event handler returns `True`, the Tuner launches the channel by calling its `start()` method. Otherwise, the Tuner closes the channel by calling its `stop()` method.

The `DATA_NOTIFY_EVENT` gives developers and users the capability to inspect new data before making a decision on whether to launch the channel. As an elaborate example, this might be useful for a sophisticated stock-analysis channel. The user makes a specific request to be notified if Company XYZ's stock falls below 70. When the new data arrives, the Tuner sends the `DATA_NOTIFY_EVENT`. The application wakes up and inspects the new data. If the price is above 70, it returns `False` and goes back to sleep. But, if the price is below 70, it returns `True` and the channel is launched, immediately displaying the alarming news.

Handling Inactive Channel Updates

1. The Tuner checks the Transmitter for new information. If no new information is available, the Tuner goes back to sleep and waits until the next update interval.
2. If new information is available, the Tuner installs the channel update, wakes up the application, and sends the `DATA_NOTIFY_EVENT`.
3. The application has a chance to inspect the data. If the application's event handler returns `True`, the Tuner calls the application's `start()` method. Otherwise, the Tuner calls the application's `stop()` method.

19.6.4 The `ApplicationFrame` `notifyAvailable()` and `notifyInstall()` Methods

Once you understand the differences between the various update events, all you need to do is catch the events by implementing the application interface `handleEvent()` method. Listing 19-9, for example, handles three channel-update events.

Most of the time, however, it is much easier to use the event handlers built right into the `ApplicationFrame` class. Listing 19-10 includes the event handling methods from the `ApplicationFrame` source code. Take a look at the listing before we examine it in detail.

Note the following in Listing 19-10:

- By default, the `ApplicationFrame` handles the `DATA_AVAILABLE_EVENT` and `DATA_INSTALLED_EVENT`. It will not handle the `DATA_NOTIFY_EVENT` or the `DATA_UPDATE_EVENT`, however. You will have to override this method to provide handling for these methods.
- When the `ApplicationFrame` receives a `DATA_AVAILABLE_EVENT`, the `notifyAvailable()` method is called. The `notifyAvailable()` method checks the channel's publishing properties to determine a course of action. These are the channel properties that were set with Castanet Publisher. If the channel's data available action (known here as `update.action`) is set to `restart`, the `ApplicationContext` `restart()` method is called. If, on the other hand, the channel's data available action is set to `install`, the `ApplicationContext` `installData()` method is called. Passing an empty string to the `installData()` method directs the Tuner to install all the updated channel files. If you want an easy way of handling the `DATA_AVAILABLE_EVENT`, the best option is simply to override the `notifyAvailable()` method.
- When the `ApplicationFrame` receives a `DATA_INSTALLED_EVENT`, the `notifyInstall()` method is called. This method does not actually do anything, but it does provide a code stub for catching the `DATA_INSTALLED_EVENT`. Simply override the `notifyInstall()` method in your own code.

Listing 19-9
Handling Channel-Update Events

```
//  Handle Channel Updates
public boolean handleEvent(Event evt) {
switch (evt.id) {
        case DATA_UPDATE_EVENT:
                saveLog();
                return true;
        case DATA_AVAILABLE_EVENT:
                installUpdate();
                return true;
        case DATA_INSTALLED_EVENT:
                displayNewData();
                return true;
        }
...
}
```

Listing 19-10
Source Code for the **ApplicationFrame** Class

```
    public void notifyAvailable(String dir) {
        if ("restart".equals(context.getParameter("update.action"))) {
            context.restart();
        } else if ("install".equals(context.getParameter("update.action")))
        {
            context.installData("");
        }
    }

    public void notifyInstall(String dir) {

    }

    public boolean handleEvent(Event evt) {
        switch (evt.id) {
          case DATA_AVAILABLE_EVENT:
            notifyAvailable((String)evt.arg);
            return true;

          case DATA_INSTALLED_EVENT:
            notifyInstall((String)evt.arg);
            return true;

          case Event.WINDOW_DESTROY:
            context.stop();
            return true;
        }
        return super.handleEvent(evt);
        }
}
```

The most interesting factor to notice here is that these update handlers are built right into *any* application that extends the `ApplicationFrame`. For example, recall that the example in Listing 19-8 reads a single line of text from the `data.txt` file and outputs this string to the Java Tuner console and to an output file. Now, try this:

1. First, publish the channel. But, this time, make sure that the Active Update Frequency option is set to Frequently and that the Active Data Available Action option is set to Restart.
2. Subscribe to the channel and leave it running.
3. Edit the `data.txt` file in your developer directory.
4 Republish your channel.
5. Wait a few minutes, and the channel updates itself. When the channel is updated, the `ApplicationFrame notifyAvailable()` method is called. This method checks the channel's publishing options and immediately restarts the application. And, once restarted, the Application displays the newly updated data text. It's as simple as that!

19.6.5 Example Application Channel

Let's now move onto a simple application channel that really takes advantage of the Castanet update architecture and really takes advantage of the `ApplicationFrame` event handlers.

The example in Listing 19-11 provides a very simple headline news application. It is actually a precursor to the scrolling ticker case study presented in section 19.7.

Before explaining the code, let's first examine what it does. When the channel is launched, it opens a `data.txt` file in the channel base directory. The data file contains a single news headline that is displayed immediately to the Java Tuner console. Most of this functionality is identical to the example in Listing 19-8.

The channel then simply sits and waits for a channel update. But, this time, instead of restarting the application, the channel handles the update dynamically and immediately displays the new headline to the user in the Java console (see Figure 19-5).

Take a look at the code before we examine it in detail.

Listing 19-11
Headline News Channel, Version 1.0

```
//       Headline News Channel, Version 1.0
//       Extends ApplicationFrame
//  Displays Updated data to the Tuner Console

import java.awt.*;
import java.io.*;
import java.net.*;
import java.util.*;
import marimba.channel.*;
import marimba.io.*;

public class HeadlineNews extends ApplicationFrame {

    //  Initialize Channel
    public void start () {
        String text = getData("data.txt");
        System.out.println ("News:   "+text);
    }

    //  Process Channel Updates
    //  New Data is now available.  Install the data.
    public void notifyAvailable (String dir) {
        System.out.println ("New Data Available.");

        // Print list of updated files
        // Depending on the list of files, you can choose to
        // install or restart the application.
        Updates update = context.getPendingUpdates();
        Enumeration files = update.getUpdates();
        while (files.hasMoreElements()) {
            String filename = (String) files.nextElement();
            System.out.println ("updated files:  "+filename);
        }
        context.installData ("");
    }

    //  Process Channel Updates
    //  New data is now installed.  Read new data from
    //  data.txt file.
    public void notifyInstall (String dir) {
        System.out.println ("New Data Installed.");
        String text = getData ("data.txt");
        System.out.println ("Updated News: "+text);
    }

    //  Get News Headline from data.txt file
    //  Must use context.getBase() method
    private String getData (String filename) {
        String text = null;
        try {
```

Continues

Listing 19-11
Continued

```
            URL data = new URL(context.getBase(), filename);
            FastInputStream in = new FastInputStream
            (data.openStream());
            text = in.readLine();
            in.close();
        }
        catch (MalformedURLException e) {
            e.printStackTrace();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
        finally {
            return text;
        }
    }
}
```

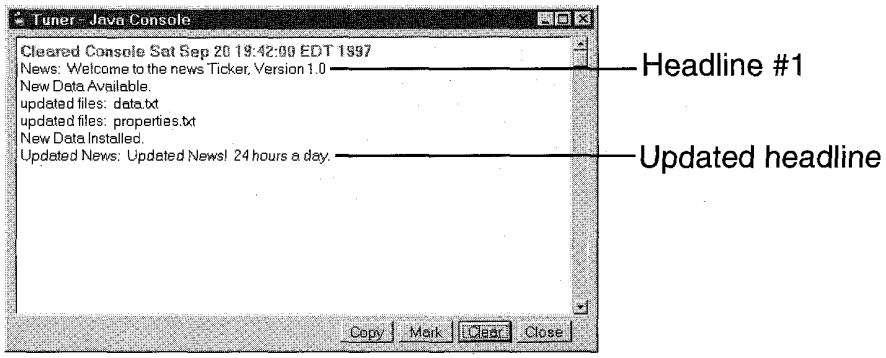


Figure 19-5
Headline News Ticker, Version 1.0

As you examine the code in Listing 19-11, note the following:

- First, the `getData()` method reads in a single line of text from the `data.txt` file. By now, the `FastInputStream` should look familiar.
- Second, we have overridden the `ApplicationFrame notifyAvailable()` method. Take a close look at this method, and you will notice that we have referenced the `ApplicationContext getPendingUpdates()` method. This method returns a Castanet `Update` object, which can

be queried for a list of pending updates. Specifically, the **Update** object has three public methods:

public Enumeration getDeletes()

Returns the enumeration of all the deletions that are pending or **null** if there are none.

public Enumeration getCreates()

Returns the enumeration of all the creations that are pending or **null** if there are none.

public Enumeration getUpdates()

Returns the enumeration of all the updates that are pending or **null** if there are none.

Here, we query the **Update** object for a list of the updated files and display them to Java Tuner console. We then call the **ApplicationContext installData()** method. Note that **installData** again is called with an empty string, directing the Tuner to install *all* of the newly updated channel files.

At times, however, it might be necessary to specifically check each of the updated, deleted, and newly created files and then selectively install only some of them. This is why the **Update** object can be important. If the channel developer publishes a new **data.txt** file, for example, the best option is to explicitly install just the data file: **ctx.installData** (**data.txt**). But what happens if the developer has updated the **HeadlineNews.class** file instead? The best option, in this case, is to simply call the application context **restart()** method and start over with a newly updated application.

- The third and final point to note about Listing 19-11 is that we have overridden the **notifyInstall()** method. When the new data has been installed successfully, we again call the **getData()** method to retrieve and display the new headline.

Try the example. Then, after you have the channel running, try editing the **data.txt** file and then republish the channel. Confirm that the new headline does indeed appear in the Java console.

19.7 Case Study: Headline News Ticker

We now are ready to examine a fully functional (and hopefully useful!) application channel. The Headline News Channel builds on the example in Listing 19-11 to provide an animated scrolling ticker on the user's desktop (see Figure 19-6).

The program itself is not much more complicated than our last example. In fact, the program uses much of the same structure provided by all the previous examples. Take a look at Listing 19-12 before we examine it in detail.

Again, the actual headline is stored in the `data.txt` file of the base channel directory. If you modify this file and republish your channel, the news ticker immediately displays the new headline. Now, note the following:

- First, we have defined two classes here. The `HeadlineNews` class handles the channel updates. The `Scroller` class handles the on-screen scrolling animation.
- The `Scroller` class really has nothing to do with the Castanet architecture. It merely paints the scrolling text with a separate animation thread and a double buffering graphics technique for smoother animation. As far as the `Headline` class is concerned, the only important method is the `setText()` method, which sets the currently displayed headlines. As noted in section 19.2.5, "The Channel Development Process," I first created the `Scroller` class as a standalone application. After I got the scroller working, I integrated it with the `HeadlineNews` class.
- The `HeadlineNews` class extends the `ApplicationFrame` class and overrides the `start()`, `notifyAvailable()`, and `notifyInstall()` methods.

Figure 19-6
Headline News Ticker, Version 2.0

Listing 19-12
Headline News Channel, Version 2.0

```
//       Headline News Channel, Version 2.0
//       Extends ApplicationFrame
//  Displays Updated Data in Scrolling Ticker
//  Created by Ethan Cerami, 1997

import java.awt.*;
import java.io.*;
import java.net.*;
import java.util.*;
import marimba.channel.*;
import marimba.io.*;

public class HeadlineNews extends ApplicationFrame {
   private Label header;
   private Scroller scroller;

   // Initialize Channel Frame
   // Use Borderlayout and add Scrolling Panel
   public void start () {
      String text = getData("data.txt");
      System.out.println ("News:  "+text);

      this.setSize (350,100);
      this.setTitle ("Headline News Ticker");
      this.setLayout (new BorderLayout());
      header = new Label ("Updated:  "+(new  Date (context.updateTime())));
      header.setBackground (Color.blue);
      header.setForeground (Color.white);
      header.setAlignment (Label.CENTER);

      scroller = new Scroller (text, 16, 20);
      this.add ("North", header);
      this.add ("Center", scroller);
      this.show();
      scroller.start();
}

   // Process Channel Updates
   // New Data is now available.  Install the data.
   public void notifyAvailable (String dir) {
      System.out.println ("New Data Available ->  Installing Data.");
      context.installData ("");
   }

   // Process Channel Updates
   // New data is now installed.  Read new data from
   // data.txt file.
   public void notifyInstall (String dir) {
      System.out.println ("New Data Installed ->  Updating Ticker");
      header.setText ("Updated:  "+(new Date (context.updateTime())));
      scroller.setText (getData ("data.txt"));
   }
```

Continues

Listing 19-12
Continued.

```
    // Get News Headlines from data.txt file
    // Must use content.getBase() method
    private String getData (String filename) {
        String text = null;
        try {
            URL data = new URL(content. getBase()), filename));
            FastInputStream in = new FastInputStream (data.openStream());
            text = in.readLine());
            in.close();
        }
        catch (MalformedURLException e) {
            e.printStackTrace();
        }
        catch (IOException e) {
            e.printStackTrace();
        }
        finally {
            return text;
        }
    }
}

// Scroller Class is responsible for scrolling the
// current news headline.
class Scroller extends Panel implements Runnable{
    private String text;
    private String newText;
    private int stringwidth;
    private Dimension d;
    private Graphics offScreenGraphics;
    private Image offScreen;
    private int currentx, currenty;
    private Thread anim;
    private Font font;
    private int speed;

    /**  Constructor Function
    @param text = news text to appear in ticker
    @param fontsize = ticker font size
    @param speed = animation delay  */
    public Scroller (String text, int fontsize, int speed) {
        this.text = text;
        font = new Font ("Helvetica", Font.BOLD, fontsize);
        this.speed = speed;
        stringwidth = 0;
    }

    /**  Set new headline text
    @ param text is the new headline to appear in ticker */
```

Listing 19-12

```
public void setText (String text) {
    newText = text;
}

/**  Paint Method  */
public void paint (Graphics g) {
    update (g);
}

/** Update Method:  Provides Double Buffering for Smoother Animation  */
public void update (Graphics g) {

    //  Prepare for Double Buffering
    if (d==null) {
        d = this.getSize ();
        offScreen = createImage (d.width, d.height);
        offScreenGraphics = offScreen.getGraphics();
        currentx = d.width;
    }

    offScreenGraphics.setFont (font);

    //  Get width of current news headline and set currenty
    if (stringwidth == 0) {
        FontMetrics fm = offScreenGraphics. getFontMetrics();
        stringwidth = fm.stringWidth (text);
        currenty = (fm.getAscent() + (d.height - (fm.getAscent() +
        fm.getDescent()))/2);
    }

    offScreenGraphics.setColor (Color.black);
    offScreenGraphics.fillRect (0,0,d.width, d.height);
    offScreenGraphics.setColor (Color.green);

    offScreenGraphics.drawString (text, currentx,currenty);
    g.drawImage (offScreen, 0,0, null);
}

/** Start Method:  Start Thread Animation */
public void start () {
    anim = new Thread (this);
    anim.start();
}

/** Stop Method:  Stop Thread Animation */
public void stop () {
    anim.stop();
}
```

Continues

Listing 19-12
Continued.

```
    /** Run Method:  Scroll Ticker;  check boundaries */
    public void run () {
        anim.setPriority (Thread.MIN_PRIORITY);
        while (true) {
            repaint();
            //  Perform automatic wrap-around
            //  and smooth transition for new headline
            if (currentx < 0-stringwidth) {
                currentx = d.width;
                if (newtext!=null) {
                    text = newtext;
                    newtext = null;
                    stringwidth =0;
                }
            }
            else currentx-;
            try { anim.sleep (speed);
            } catch (InterruptedException e) {
            }
        }
    }
}
```

- The **`start()`** method initializes the scroller with the values retrieved from the **`getData`** method. It then lays out the frame with the AWT **`BorderLayout`** Manager. The header of the ticker also is set to display the date and time of the last updates. For this, we use the application context **`updateTime()`** method.
- When a channel update occurs, the Tuner sends the **`DATA_AVAILABLE_EVENT`**, immediately triggering the **`notifyAvailable()`** method. We choose to install the entire channel update by calling **`installData()`** with an empty string.
- After the Tuner successfully installs the new data, it sends the **`DATA_INSTALLED_EVENT`**, immediately triggering the **`notifyInstall()`** method. Here, we set the new update time and update the scroller with the string returned from the **`getData()`** method.

Online Resource

- Castanet Developer Documentation, Channel Development: The Big Picture
  `http://www.marimba.com/doc/Castanet_Developer_Docs/current/channeldev.html`. Ten concise pages on developing your own application channels. Special emphasis on handling update events.

Appendix A

Marimba Castanet API

The complete Marimba Castanet API is available online at `http://www.marimba.com/doc/`.

A.1 Package `marimba.channel`

A.1.1 Interface `marimba.channel.Application`

This is the application interface for the Marimba Tuner. A class that implements this interface can act as an application in the context of the Tuner.

```
public interface Application extends Object {

        // Constants
        public final static int DATA_UPDATE_EVENT;
        public final static int DATA_AVAILABLE_EVENT;
        public final static int DATA_INSTALLED_EVENT;
        public final static int DATA_NOTIFY_EVENT;

        // Public Instance Methods
        public abstract void setContext(ApplicationContext);
        public abstract void start();
        public abstract void stop();
        public abstract boolean handleEvent(Event evt);
}
```

A.1.2 Interface

marimba.channel.ApplicationContext

This provides the context for an application. It is the interface of the application to the Tuner.

```
public interface ApplicationContext extends Object {

        // Public Instance Methods
        public abstract URL getBase();
        public abstract URL getCodeBase();
        public abstract URL getDataBase();
        public abstract boolean channelFileExists(String
               path);
        public abstract String[] listChannelDirectory(String
               directory);
        public abstract String getDataDirectory();
        public abstract Updates getPendingUpdates();
        public abstract String getChannelDirectory();
        public abstract String getServerName();
        public abstract String getChannelName();
        public abstract String getParameter(String nm);
        public abstract void installData(String dir);
        public abstract boolean appendLog(byte data[]);
        public abstract boolean appendLog(String data);
        public abstract byte[] getProfile();
        public abstract boolean setProfile(byte data[]);
        public abstract long publishTime();
        public abstract long updateTime();
        public abstract void restart();
        public abstract void stop();
        public abstract void update();
        public abstract void startChannel(String serverName,
               String channelName);
        public abstract void subscribeChannel(String
               serverName, String channelName);
        public abstract void unsubscribeChannel(String
               channelName);
        public abstract void removeChannel(String
               channelName);
        public abstract String[] listChannels();
        public abstract String getChannelStatus(String
               channelName);
        public abstract AudioClip getAudioClip(URL url);
        public abstract Image getImage(URL url);
        public abstract void showDocument(URL url);
        public abstract void showDocument(URL url, String
               frame);
        public abstract void showDocument(String url);
        public abstract void showDocument(String url, String
               frame);
        public abstract void showStatus(String msg);
}
```

A.1.3 Class `marimba.channel.ApplicationFrame`

A helper class for creating channel applications. This class creates, shows, and destroys an AWT frame as the base frame of the application.

```
public class ApplicationFrame extends Frame implements
       Application {

       //  Public Constructor
       public ApplicationFrame()

       //  Public Instance Methods
       public void setContext(ApplicationContext context);
       public ApplicationContext getContext();
       public void stop();
       public void notifyAvailable(String dir);
       public void notifyInstall(String dir);
       public boolean handleEvent(Event evt);
}
```

A.1.4 Class `marimba.channel.ApplicationPlayerFrame`

A presentation player for channels built with Bongo.

```
public class ApplicationPlayerFrame extends PlayerFrame
       implements Application {

       // Public Constructor
       public ApplicationPlayerFrame();

       //  Protected Instance Variable
       protected ApplicationContext context;

       // Public Instance Methods
       public void setContext(ApplicationContext context);
       public ApplicationContext getContext();
       public void stop();
       public void notifyAvailable(String dir);
       public void notifyInstall(String dir);
       public boolean handleEvent(Event evt);
}
```

A.1.5 Class `marimba.channel.Updates`

This class generates a list of pending updates, creates, and deletes that would be performed if all the changes for the specified channel were installed right now.

```
public class Updates extends Object {

        //  Public Constructor
        public Updates(Vector updates, Vector creates,
               Vector deletes)

        // Public Instance Methods
        public Enumeration getDeletes();
        public Enumeration getCreates();
        public Enumeration getUpdates();
}
```

A.2 Package `marimba.io`

A.2.1 Class `marimba.io.FastInputStream`

A fast, unsynchronized buffered input stream. This stream combines a lot of the functionality of `BufferedInputStream`, `ByteArrayInputStream`, and `DataInputStream`.

```
public class FastInputStream extends FilterInputStream
implements DataInput {

        //  Public Constructors
        public FastInputStream(String file) throws
               IOException;
        public FastInputStream(File file) throws IOException;
        public FastInputStream(RandomAccessFile in);
        public FastInputStream(InputStream in);
        public FastInputStream(InputStream in, int size);
        public FastInputStream(byte buf[]);
        public FastInputStream(byte buf[], int off, int
               len);

        //  Protected Instance Variables
        protected byte buf[];
        protected int count;
        protected int count;
        protected int markpos;

        //  Public Instance Methods
        public boolean getError();
        public boolean backup(int n);
        public int read();
        public int read(byte b[], int off, int len);
        public long skip(long n);
        public int available() throws IOException;
        public void close() throws IOException;
        public void readFully(byte b[]);
        public void readFully(byte b[], int off, int len);
        public int skipBytes(int n);
```

```
        public boolean readBoolean();
        public byte readByte();
        public int readUnsignedByte();
        public short readShort();
        public int readUnsignedShort();
        public char readChar();
        public int readInt();
        public long readLong();
        public float readFloat();
        public double readDouble();
        public String readLine();
        public String readUTF();
        public PropertyObject readObject();
        public boolean markSupported();
        public void mark(int limit);
        public void reset();
        public void seek(long p) throws IOException;
        public long getFilePointer() throws IOException;
}
```

A.2.2 Class `marimba.io.FastOutputStream`

A fast, unsynchronized buffered output stream. This stream combines the functionality of `DataOutputStream`, `PrintStream`, and `BufferedOutputStream`.

```
public class FastOutputStream extends FilterOutputStream
implements DataOutput {

        // Public Constructors
        public FastOutputStream(String file) throws
               IOException;
        public FastOutputStream(File file) throws
               IOException;
        public FastOutputStream(RandomAccessFile file) throws
               IOException;
        public FastOutputStream(OutputStream out);
        public FastOutputStream(OutputStream out, int size);
        public FastOutputStream();
        public FastOutputStream(int size);
        public FastOutputStream(byte buf[]);

        // Protected Instance Variables
        protected byte buf[];
        protected int count;

        //  Public Instance Variables
        public OutputStream out;

        //  Public Instance Methods
        public boolean getError();
        public final void write(int b);
        public final void write(byte b[], int off, int len);
        public final void flush() throws IOException;
        public void close() throws IOException;
```

```
    public void justClose();
    public int size();
    public byte[] getByteArray();
    public byte[] toByteArray();
    public String toString();
    public final void writeBoolean(boolean v);
    public final void writeByte(int v);
    public final void writeChar(int v);
    public final void writeInt(int v);
    public final void writeLong(long v);
    public final void writeFloat(float v);
    public final void writeDouble(double v);
    public final void writeBytes(String s) throws
           IOException;
    public final void writeChars(String s) throws
           IOException;
    public final void writeUTF(String str);
    public final void writeObject(PropertyObject obj);
    public final void print(char ch);
    public final void println(char ch);
    public final void print(String str);
    public final void println(String str);
    public final void print(Object obj);
    public final void println(Object obj);
    public final void print(long n);
    public final void println(long n);
    public final void println();
    public void seek(long p) throws IOException;
    public long getFilePointer() throws IOException;
}
```

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

R

S

ABOUT THE AUTHOR

Ethan Cerami is an Associate in the Research and Development division of Cognitive Communications, L.L.C., a firm specializing in Intranet development for Fortune 500 companies. He is also currently completing his masters degree in computer science at New York University, where he teaches undergraduate courses in computer programming.

SOFTWARE AND INFORMATION LICENSE

The software and information on this diskette (collectively referred to as the "Product") are the property of The McGraw-Hill Companies, Inc. ("McGraw-Hill") and are protected by both United States copyright law and international copyright treaty provision. You must treat this Product just like a book, except that you may copy it into a computer to be used and you may make archival copies of the Products for the sole purpose of backing up our software and protecting your investment from loss.

By saying "just like a book," McGraw-Hill means, for example, that the Product may be used by any number of people and may be freely moved from one computer location to another, so long as there is no possibility of the Product (or any part of the Product) being used at one location or on one computer while it is being used at another. Just a book cannot be read by two different people in two different places at the same time, neither can the Product be used by two different people in two different places at the same time (unless, of course, McGraw-Hill's rights are being violated).

McGraw-Hill reserves the right to alter or modify the contents of the Product at any time.

This agreement is effective until terminated. The Agreement will terminate automatically without notice if you fail to comply with any provisions of this Agreement. In the event of termination by reason of your breach, you will destroy or erase all copies of the Product installed on any computer system or made for backup purposes and shall expunge the Product from your data storage facilities.

LIMITED WARRANTY

McGraw-Hill warrants the physical diskette(s) enclosed herein to be free of defects in materials and workmanship for a period of sixty days from the purchase date. If McGraw-Hill receives written notification within the warranty period of defects in materials or workmanship, and such notification is determined by McGraw-Hill to be correct, McGraw-Hill will replace the defective diskette(s). Send request to:

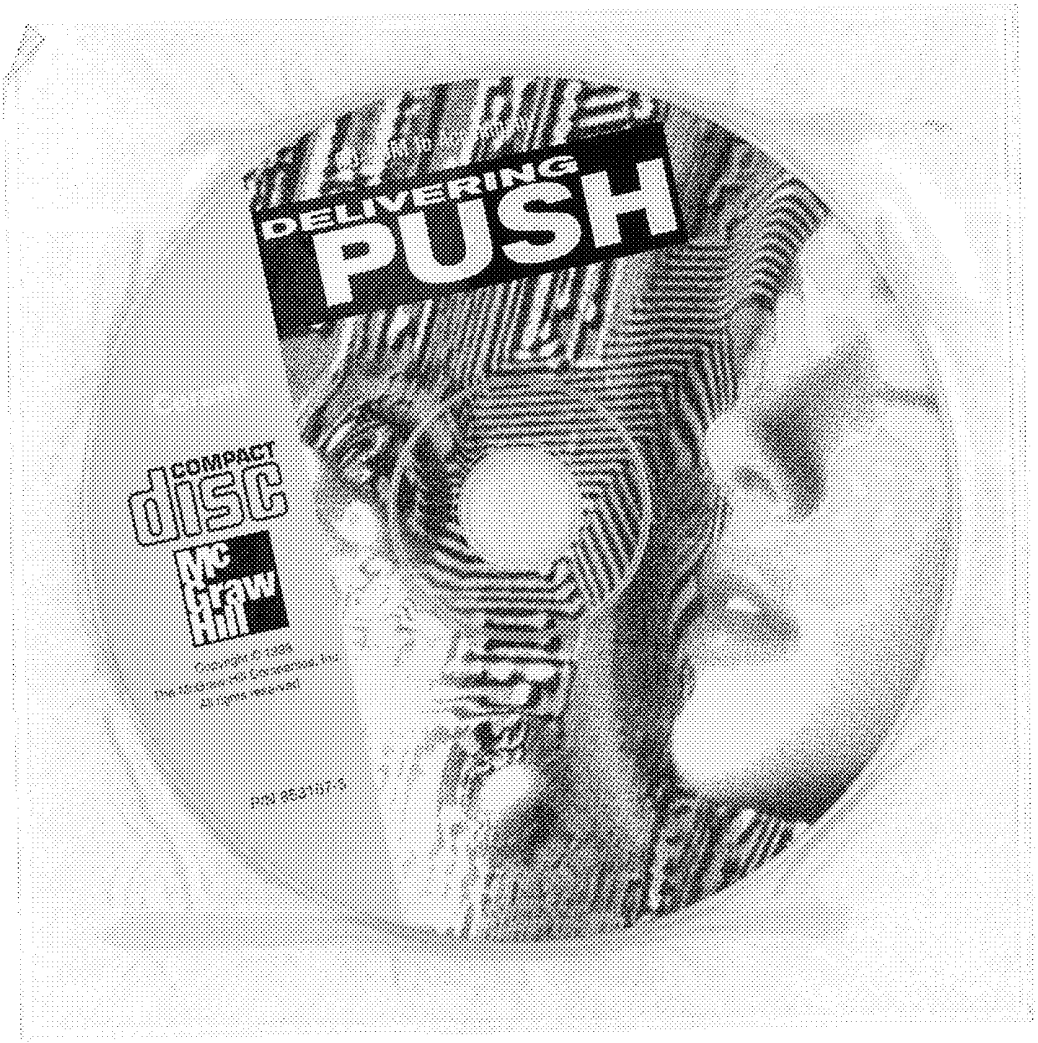
Customer Service
McGraw-Hill
Gahanna Industrial Park
860 Taylor Station Road
Blacklick, OH 43004-9615

The entire and exclusive liability and remedy for breach of this Limited Warranty shall be limited to replacement of defective diskette(s) and shall not include or extend any claim for or right to cover any other damages, including but not limited to, loss of profit, data, or use of the software, or special, incidental, or consequential damages or other similar claims, even if McGraw-Hill has been specifically advised as to the possibility of such damages. In no event will McGraw-Hill's liability for any damages to you or any other person ever exceed the lower of suggested list price or actual price paid for the license to use the Product, regardless of any form of the claim.

THE McGRAW-HILL COMPANIES, INC. SPECIFICALLY DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Specifically, McGraw-Hill makes no representation or warranty that the Product is fit for any particular purpose and any implied warranty of merchantability is limited to the sixty day duration of the Limited Warranty covering the physical diskette(s) only (and not the software or information) and is otherwise expressly and specifically disclaimed.

This Limited Warranty gives you specific legal rights; you may have others which may vary from state to state. Some states do not allow the exclusion of incidental or consequential damages, or the limitation on how long an implied warranty lasts, so some of the above may not apply to you.

This Agreement constitutes the entire agreement between the parties relating to use of the Product. The terms of any purchase order shall have no effect on the terms of this Agreement. Failure of McGraw-Hill to insist at any time on strict compliance with this Agreement shall not constitute a waiver of any rights under this Agreement. This Agreement shall be construed and governed in accordance with the laws of New York. If any provision of this Agreement is held to be contrary to law, that provision will be enforced to the maximum extent permissible and the remaining provisions will remain in force and effect.



Web Development

$49.95 U.S.A.

Includes Netscape Navigator

- Learn about all major push players, with snapshots of today's– and tomorrow's– market
- Take advantage of plug-and-play solutions for Netscape and Microsoft
- Use and create Active channels, Netcaster channels, and PointCast channels and animations
- Use the Castanet tuner, transmitter, and publisher with complete instructions, including Bongo
- Develop application channels, using coded models
- Master applet channel solutions

Push Solutions

Delivering Push gives you at-hand solutions-real code you can use. You get clear and detailed instructions for creating push solutions for both Microsoft and Netscape browsers! This book gives you everything you need to slash development time–from ready-to-run code to expert, written-for-developers procedures for creating, installing, and configuring applications.

Inside **Delivering Push** you'll find introductions to unfamiliar technology as well as expert, step-by-step solutions for developing advanced push solutions–including plenty of sample code. This complete, professional solutions kit helps you reduce development time and solve complex problems quickly with made-to-order applications. Covering all major push players, this book gives Web developers the answers you want in language that's clear, to the point, and written just for you. The McGraw-Hill Web Developer Series was written by developers for developers.

About the Author

Ethan Cerami is an associate in the Research and Development division of Cognitive Communications, L.L.C., a firm specializing in Intranet development for Fortune 500 companies. He is currently completing his masters degree in computer science at New York University, where he teaches undergraduate courses in computer programming.

Cover Design: Dorothy Wachtenheim
Cover Photo: © Fritz/Superstock

Visit us on the World Wide Web at
www.computing.mcgraw-hill.com

McGraw-Hill
A Division of The McGraw-Hill Companies