In this appendix:

- *How Images are Loaded*
- *A Brief Tour of sun.awt.image*

D

Image Loading

D.1 How Images are Loaded

You have seen how easy it is to display an image on screen and have probably guessed that there's more going on behind the scenes. The `getImage()` and `drawImage()` methods trigger a series of events that result in the image being available for display on the `ImageObserver`. The image is fetched asynchronously in another thread. The entire process* goes as follows:

1. The call to `getImage()` triggers `Toolkit` to call `createImage()` for the image's `InputStreamImageSource` (which is a `URLImageSource` in this case; it would be a `FileImageSource` if we were loading the image from a local file).
2. The `Toolkit` registers the image as being "desired." Desired just means that something will eventually want the image loaded. The system then waits until an `ImageObserver` registers its interest in the image.
3. The `drawImage()` method (use of `MediaTracker` or `prepareImage()`) registers an `ImageObserver` as interested.
4. Registering an `ImageObserver` kicks the image's `ImageRepresentation` into action; this is the start of the loading process, although image data isn't actually transferred until step 9. `ImageRepresentation` implements the `ImageConsumer` interface.
5. The start of production registers the image source (`ImageProducer` `URLImageSource`) with the `ImageFetcher` and also registers the `ImageRepresentation` as an `ImageConsumer` for the image.

* This summary covers Sun's implementation (JDK). Implementations that don't derive from the JDK may behave completely differently.

6. The `ImageFetcher` creates a thread to get the image from its source.
7. The `ImageFetcher` reads data and passes it along to the `InputStreamImage-Source`, which is a `URLImageSource`.
8. The `URLImageSource` determines that `JPEGImageDecoder` is the proper `ImageDecoder` for converting the input stream into an `Image`. (Other `ImageDe-coders` are used for other image types, like GIF.)
9. The `ImageProducer` starts reading the image data from the source; it calls the `ImageConsumer` (i.e., the `ImageRepresentation`) as it processes the image. The most important method in the `ImageConsumer` interface is `setPixels()`, which delivers pixel data to the consumer for rendering onscreen.
10. As the `ImageConsumer` (i.e., the `ImageRepresentation`) gets additional information, it notifies the `ImageObserver` via `imageUpdate()` calls.
11. When the image is fully acquired across the network, the thread started by the `ImageFetcher` stops.

As you see, there are a lot of unfamiliar moving pieces. Many of them are from the `java.awt.image` package and are discussed in Chapter 12, *Image Processing*. Others are from the `sun.awt.image` package; they are hidden in that you don't need to know anything about them to do image processing in Java. However, if you're curious, we'll briefly summarize these classes in the next section.

D.2 A Brief Tour of sun.awt.image

The classes in `sun.awt.image` do the behind-the-scenes work for rendering an image from a file or across the network. This information is purely for the curious; you should never have to work with these classes yourself.

`Image`

The `Image` class in this package represents a concrete `Image` instance. It contains the basis for the `Image` class that is actually used on the run-time platform, which exists in the package for the specific environment. For instance, the `sun.awt.win32` package includes the `W32Image` (Java 1.0), the `sun.awt.windows` package includes `WImage` (Java 1.1), while the `sun.awt.motif` package includes the `X11Image`, and the `sun.awt.macos` package includes the `MacImage`.

`ImageRepresentation`

The `ImageRepresentation` is the `ImageConsumer` that watches the creation of the image and notifies the `ImageObserver` when it is time to update the display. It plays an important part in the overall control of the `Image` production process.

Image sources

A Java image can come from three different sources: memory (through `createImage()`), local disk, or the network (through `getImage()`).

- `OffScreenImageSource` implements `ImageProducer` for a single framed image in memory. When an `Image` created from an `OffScreenImageSource` is drawn with `drawImage()`, the `ImageObserver` parameter can be `null` since all the image information is already in memory and there is no need for periodic updating as more is retrieved from disk. You can get the graphics context of `OffScreenImageSource` images and use the context to draw on the image area. This is how double buffering works.
- `InputStreamImageSource` implements `ImageProducer` for an image that comes from disk or across the network. When an `Image` created from an `InputStreamImageSource` is drawn with `drawImage()`, the `ImageObserver` parameter should be the component being drawn on (usually `this`) since the image information will be loaded periodically with the help of the `ImageObserver` interface). This class determines how to decode the image type and initializes the `ImageDecoder` to one of `GifImageDecoder`, `JPEGImageDecoder`, or `XbmImageDecoder`, although that can be overridden by a subclass. It can use a `ContentHandler` to work with unknown image types.
- `FileImageSource` is a subclass of `InputStreamImageSource` for images that come from the filesystem. It uses the filename to determine the type of image to decode and checks the security manager to ensure that access is allowed.
- `URLImageSource` is a subclass of `InputStreamImageSource` for images that are specified by a URL.
- `ByteArrayImageSource` (Java 1.1 only) is a subclass of `InputStreamImageSource` for images that are created by calling `Toolkit.createImage(byte[])`.

Image decoders

An `ImageDecoder` is utilized to convert the image source to an image object. If there is no decoder for an image type, it can be read in with the help of a `ContentHandler` or your own class that implements `ImageProducer`, like the `PPMImageDecoder` shown in Chapter 12.

- `GifImageDecoder` reads in an image file in the GIF format.
- `JPEGImageDecoder` reads in an image file in the JPEG format.

- `XbmImageDecoder` reads in an image file in the XBM format. Although XBM support is not required by the language specification, support is provided with Netscape Navigator, Internet Explorer, HotJava, and the Java Developer's Kit from Sun.

`ImageFetcher`

The `ImageFetcher` class fetches the actual image from its source. This class creates a separate daemon thread to fetch each image. The thread is run at a higher priority than the default but not at the maximum priority.