4 Chapter

Planning Your Application

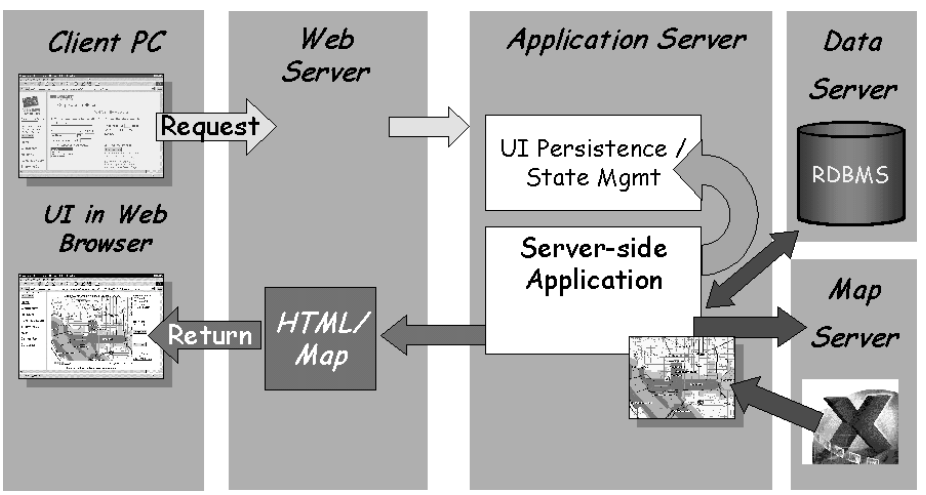
This chapter gives you "food for thought" as you consider designing and building your MapXtreme Java application. It begins with a broad introduction to web-based deployment, infrastructure requirements and necessary skills, followed by an overview of MapXtreme Java components and common configurations.

Web Deployment

Organizations worldwide have deployed mapping applications that give users access to mapping information, leading to better business decisions. In today's world, the web is driving these deployments. The advantages to web-based deployments are many:

- Reduced cost of ownership: It is cost effective to distribute applications across the web because it eliminates the need to install components on every machine that will access the application.
- Scalability: Web systems are served from powerful servers. If the number of users increases, more servers can be added.
- Access to data and software: Each user of your application accesses the latest version. You introduce software and data updates to all users at once.
- Security: Since security is centralized, you have more control over how you want to implement it.

This diagram illustrates a generalized view of a web-deployed mapping application.



On the client side, the user interacts with the mapping application via HTML pages and / or an applet in his or her browser. The interaction is based on a request / response scenario. The user makes a request, for example, to zoom in on an area of the map. That request goes to the web server. A server-side application communicates with the map server to provide the requested information. The updated map is returned to the client's browser embedded in the HTML page or applet. Should any of the data that is

needed to create the map reside in an RDBMS, calls are made to the database via JDBC to retrieve the data.

An example of a useful web-based mapping application serves the wireless telecommunications industry. Frequently, potential customers want to know if their location is inside or outside the company's coverage area. By providing coverage maps embedded in an HTML page that is deployed over the Internet, the telco company can provide an immediate response to customers in a self-service framework. The customer can interact with the map by using navigation tools such as zoom in or pan. Each click is a request that is sent to telco application to regenerate the map.

Infrastructure Requirements

In general, web-based mapping deployments utilize standard components, including a server machine running a web server, an application server (may be the same as the web server), and a database of map data for background maps and custom data specific to the application.

MapXtreme Java web-based deployments require Java 2 support. Additionally, the web server must support servlets since the mapping server for MapXtreme Java is deployed as a servlet. Data for maps can be stored locally or accessed from an RDBMS via JDBC. The application that you are to build can be in the form of a servlet, JavaServer Pages, Enterprise JavaBeans or an applet.

Necessary Skill Sets

MapXtreme Java is a product for Java developers experienced in writing servlets and applets. If you are building an application, skills in creating a Java GUI are also necessary.

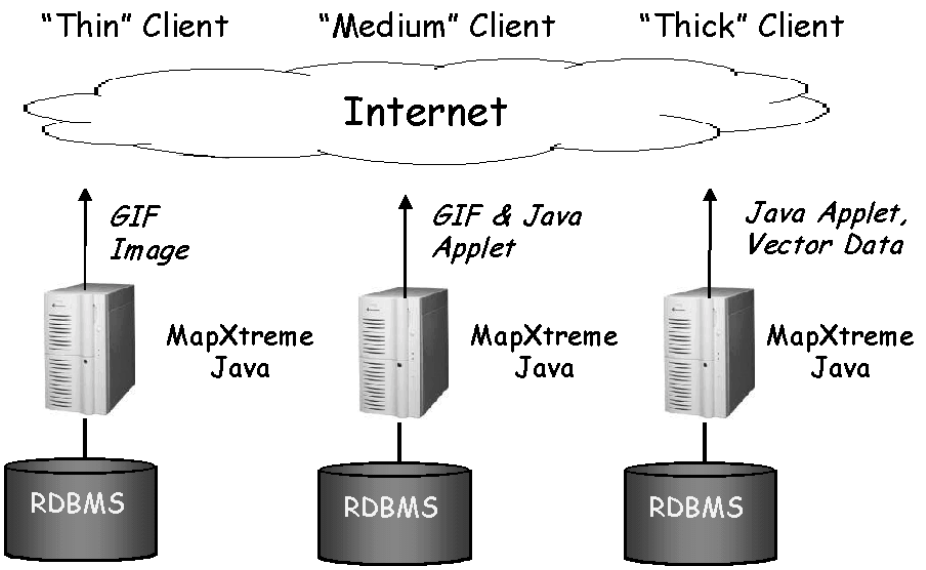
Additionally, since MapXtreme Java is deployed on the web, you should have web development and HTML expertise available to your project. How much you will need is dependent on the development tool that you are using.

If you are accessing data in an RDBMS, skills in database usage and administration are also necessary.

Finally, because this is a mapping application you are creating, some familiarity with mapping concepts and / or MapInfo mapping products is helpful. The basics of mapping is covered in Chapter 5.

Deployment Options

Deployment options for MapXtreme Java can be catagorized into three types. The illustration below shows a generalized view of these deployments: thin client, medium client, and thick client. The difference is in how much software and data is sent to the client.



An overview of each type is presented below, followed by a discussion of the components that make up MapXtreme Java. Configuration details, including pros and cons of each, conclude the chapter.

Thin Client

In a thin client deployment, the user interacts with HTML pages in a browser. The map is typically a GIF image embedded in the HTML. The map request processing occurs on the server. This is the classic Internet deployment that does not require Java on the client.

To build this type of application, you will need to know how to develop the server-side application that generates the HTML.

Thick Client

A thick client is at the other end of the spectrum. The client downloads a Java applet that provides a more sophisticated user interface than straight HTML. Additionally, MapXtreme Java can return vector data instead of a raster image. Because of the increased download time for the applet, this deployment is better handled on intranet systems where the client side can be better controlled.

To build this type of application, you will need to know how to build a Java applet and use JavaBeans.

Medium Client

In between the thin and thick options is the medium client. Like the thick client, the medium client downloads an applet so the client must support Java. Like the thin client, the medium client receives a raster image of the map. The applet can give you a more sophisticated user interface than straight HTML and additional map tools, such as a marquee selection tool.

To build this type of application, you will need to know how to develop an applet and a server-side application that interact.

Now that you've had a general description of web-based mapping, let's look in detail at the specific components of MapXtreme Java. The chapter concludes with a discussion on MapXtreme Java configurations and design considerations, including pros and cons for choosing the best one for your needs.

MapXtreme Java Overview

There are four main components to MapXtreme Java Edition: MapXtremeServlet, the MapJ object, Data Providers, and Renderers. These components work together to access geographic data, manipulate it, and provide a map or data to your application.

MapXtremeServlet

The MapXtremeServlet is the mapping server provided in the MapXtreme Java product. It services three types of client requests:

- requests for map images
- requests for vector map data
- requests for map metadata (e.g. the column names of a Layer in a map).

MapXtremeServlet responds to HTTP POST requests. Currently MapJ objects are the only clients that are capable of communicating with MapXtremeServlet.

MapXtremeServlet is designed to leverage the capabilities of its parent servlet container. MapXtremeServlet is stateless, it relies on the client request to fully describe the state of the map. Image requests are handled within MapXtremeServlet by a multi-threaded "Renderer server". Similarly, requests for map data are handled by a multi-threaded "DataProvider server". These factors make MapXtremeServlet highly scalable when deployed within a parent servlet container

While MapXtremeServlet focuses on fulfilling mapping tasks, its parent servlet container can handle load balancing, fault tolerance, and security management. Servlet containers are found in web servers such as Sun's JavaWebServer, and in application servers such as BEA's WebLogics. Web servers such as Apache's Web Server or Microsoft IIS do not include a servlet container. In these cases a separate servlet container plug-in such as JRun or Tomcat must be used.

MapJ Object

The MapJ object manages the state of a map. It maintains a map's center and zoom, coordinate systems, distance units, and the Layers that collectively comprise the map. MapJ is the topmost level of MapXtreme's client API.

MapJ objects can be configured to work with different types of Renderers and DataProviders. In the most typical configuration MapJ is a client of MapXtremeServlet. MapJ sends requests to a MapXtremeServlet instance and as part

of the request provides the servlet with its current state. MapJ obtains map images and data from the servlet.

MapJ can also work stand-alone to directly obtain map data and produce map images. A strength of MapXtreme's component based design is that MapJ can be configured with other variations. For instance, MapJ can be configured to access map data via one or more instances of MapXtremeServlet, but still be responsible for displaying the map image.

Since MapJ's primary purpose is to maintain map state it has a small memory footprint. This makes MapJ ideally suited for being deployed in the middle tier of n-tier architectures. See page 57 for more on deployment configurations.

Renderers

Renderers display map data. There are two types of Renderers: LocalRenderer and MapXtremeImageRenderer. A LocalRenderer can be created from any Java AWT Component, and is "local to" or in the same process space as the MapJ object to which it is associated. It uses DataProviders to directly obtain map Features for each Layer in a map. The LocalRenderer then draws the Features into its Component's Graphics object.

A MapXtremeImageRenderer can be created from a URL reference to an instance of MapXtremeServlet. When MapJ uses a MapXtremeImageRenderer it signifies that it wants to defer map rendering to an instance of MapXtremeServlet. The servlet satisfies this request by returning a raster image to the MapJ client. Various raster formats including GIF, JPEG, and PNG are supported by MapXtremeServlet. Of note, MapXtremeServlet's "Renderer server" satisfies rendering requests by using instances of LocalRenderers and exporting images to the desired raster formats.

Data Providers

Data Providers are the key link between your MapJ object and your map data. Each Layer object which is part of MapJ has its own internal Data Provider. Data Providers are used to access data sources and return vector data. Data Providers are also invoked during rendering when MapJ uses a LocalRenderer.

MapXtreme has Data Providers for accessing the following data sources:

- MapInfo tables
- Oracle8*i* with Spatial Option
- SpatialWare for Oracle 7.0 or 8.0
- Informix Universal Server SpatialWare DataBlade
- DB2 SpatialWare Extender
- JDBC compatible tables containing longitude and latitude columns
- ESRI Shapefiles
- Raster files
- MapInfo Grid

A MapJ object has two ways of accessing a data source. The first approach is to directly access the data source.

The second method is to make a request to an instance of MapXtremeServlet to get the data. MapXtremeServlet will then use a DataProvider from its "Data Provider server" to directly access the data source. As MapXtremeServlet obtains data from the data source, it will stream the data back to the client MapJ object. MapXtremeServlet uses an extremely efficient compression scheme to stream the data. One of its capabilities is to take into acccount the needed resolution of the data. For instance, when the data is used for rendering a 640 x 480 image, the data can be transmitted at a much higher level of granularity than it may be stored.

Each Layer associated with MapJ specifies how it would like to access its underlying data source through a "Data Provider reference". A LocalDataProviderRef signifies that data access should occur "local to" or within the process space containing MapJ. A MapXtremeDataProviderRef denotes that a MapXtremeServlet instance will act as an intermediary in accessing the data source.

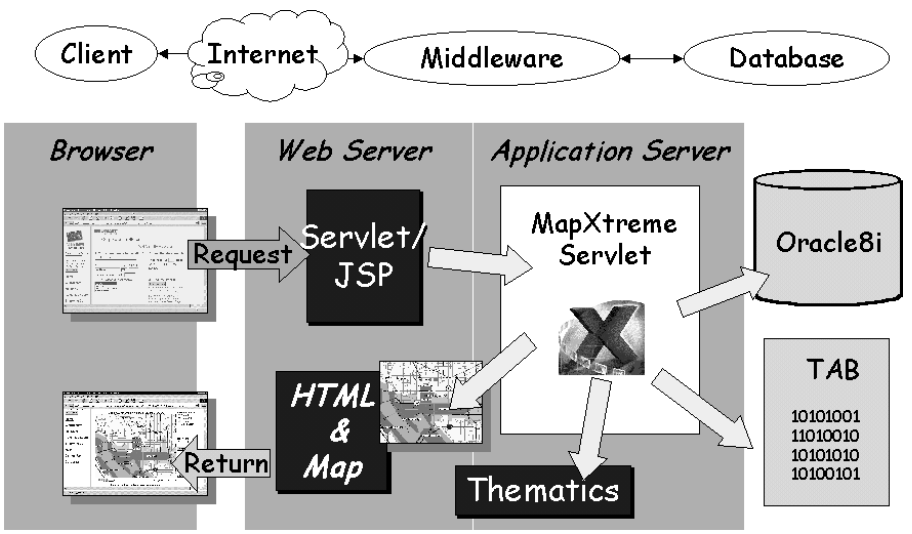
Data Providers are discussed in Chapter 9: Mapping in Layers.

Configuration Options

Now that you are familiar with MapXtreme's components and had an introduction to deployment options, let's look at how you can use them to your best mapping advantage. MapXtreme Java is suited to two- and three-tier web applications. The difference between them is where the MapJ component is located, on the server or client.

Three-Tier Configuration

The following illustration shows the most common configuration — a browser as the client, your business logic which uses MapJ objects in the middle tier, MapXtremeServlet in the middle tier, and the database in the database tier. Your application may utilize any combination of servlets, JavaServer Pages, or Enterprise JavaBeans.



When the client issues a request through the browser, your web server forwards the request to your application, which in turn, may update the state of a MapJ object. The MapJ object is then used to communicate a mapping request to MapXtremeServlet. If the request is for an image, the MapXtremeServlet will return a raster image of the map to the application. The application can then embed this image within an HTML page and return the page to the end-user's browser.

The middle tier application can also leverage MapXtreme's servlet library to aid in building the HTML page. For instance, there are library methods for creating an HTML- based layer control.

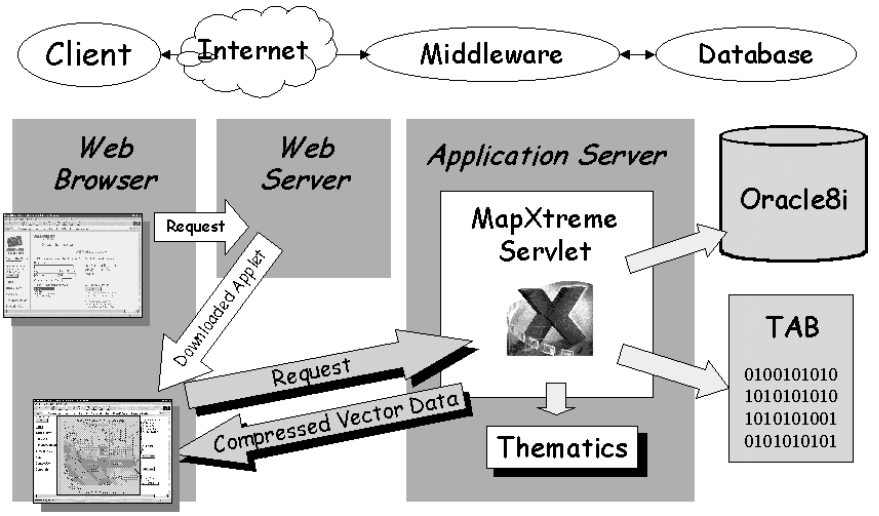
This three-tier architecture has the following characteristics:

- MapJ is deployed in the middle tier within custom application.
- MapXtremeServlet is deployed in middle tier.
- Java is not required on client. Client can send HTTP requests and can receive HTML pages as responses.
- Produces minimal network traffic: Applets are not required, so there is no applet to download.Vector data is not sent to the client, only HTML pages with embedded raster images. Raster formats such as GIF typically produce map images that are 15 - 25K in size.

These characteristics make the three-tier architecture ideally suited to Internet deployments in which you as the Web application developer have little control over your client's configuration. This deployment is a "lowest common denominator" approach and can be used to satisfy clients that do not have Java capable browsers and/or have low network bandwidth.

Two-Tier Configuration

In a two-tier, or "thick client" configuration MapJ and your business logic are deployed client-side, typically as an applet within a browser. A main advantage with this type of deployment is that it allows you to use MapXtreme's JavaBeans. Applications can be created much more rapidly working with MapXtreme's JavaBeans in a visual RAD environment than working at the lower level MapJ API. MapXtreme's JavaBeans provide visual map tools, toolbars, wizards, and map display components ready for inclusion in your application. Another benefit with thick clients is the potential for users to interact with local data on their machines.



In a two-tier deployment, a client will first download an applet containing the JavaBeans from your Web site. Once the client starts running the applet no further communication with the Web server is necessary. For instance, the applet may be configured to access data sources directly and to also perform the map rendering. In this case, when the applet needs to draw a map, it will fetch the data from the data source and then do local rendering.

The two-tier architecture has the following characteristics:

- MapJ is deployed client-side.
- MapXtremeServlet may or may not be deployed in the middle tier.
- Java required on client: The client browser must have support for a Java 2 Platform VM (or have a suitable plug-in).
- Heavier network traffic: The applet containing the JavaBeans must be downloaded. Vector data may be sent to the client and the size of the vector data is much more variable than the size of raster files.

These characteristics make the two-tier architecture most suited to intranet deployments in which the deployment environment is more homogenous and controlled. When the applet is responsible for rendering and data access, a high network bandwidth is required. More powerful machines may also be required on the client.

Two-Tier Hybrid Configuration

Hybrid configurations are also possible. For instance the applet may be configured to go through a MapXtremeServlet instance to obtain some or all of its data. Using MapXtremeServlet is recommended if you are accessing an RDBMS. A JDBC driver is required to access an RDBMS, and these do not work well within an applet. It is much simpler to keep the JDBC access in the middle tier with MapXtremeServlet and allow it to stream the data to the applet. Furthermore, the applet can be structured to use MapXtremeServlet to obtain map images.

Design Considerations

Now that you've looked over the configuration options, keep these elements in mind as you plan your mapping application.

Client Side

- Are you deploying over the Internet or via a corporate intranet?
- What is the network bandwidth?
- Is the client an applet or stand-alone application? Will the client need additional software or resources, such as JDBC drivers, to run?
- Are you designing your application for a specific platform, for any platform, or a mixed environment?
- What browser will your users be using? Is Swing support or other browser plug-ins required?
- How much mapping functionality is needed client-side? How useful are the JavaBeans such as the AddThemeWizard and LegendContainer for your needs?

Server-Side

- How complex of an application are you building? Do you have the necessary hardware?
- How many users do you expect to use your application? What is the peak user load expected?
- What services do you want or need from aWeb server and / or application server?
- Do you have the appropriate skill sets for the type of application you are building? It can include Java programming, database administration, web development, etc.

- Have you considered any security or network issues?
- What other software will your application need to interact with?
- What version of Java will be used? Do all the components support a common version?

Whether you decide you need a thin client deployment for map images or thick applet that provides additional functionality, MapXtreme Java is a flexible development tool that can help you contruct the web mapping application you need.

The next chapter introduces you to the basics of mapping. Chapters 6 and 7, MapXtreme JavaBeans and Writing Your Own Servlets, respectively, can help jumpstart your application development with pre-defined mapping components and example applications. In Chapters 8-13 we turn our attention to the MapJ API, including the MapJ object, Layers, Data Providers, Features, Searches, Labeling, Renditions, and Themes. Chapter 14 covers the Map Definition Manager, the tool for creating and customizing your background maps.