6 Chapter

MapXtreme JavaBeans

MapXtreme Java Edition 3.0 offers MapXtreme JavaBeans that allow you to easily embed live vector maps in a Java applet or application. The JavaBeans give you powerful user-interface elements, such as a Layer Control dialog and a "Wizard" for adding thematic shading to a map.

Overview

The MapXtreme JavaBeans utilize the latest Java technology and are based on Java's Swing components, part of the Java 2 Standard Edition Platform. If you are distributing applets using these JavaBeans, end users will need a suitable plug-in if their browsers do not support J2SE.

MapXtreme JavaBeans are easy to customize . They can be used to build applets that run in a user's browser or can be built into a stand-alone application. The Map Definition Manager that ships with MapXtreme Java is an example of an application that uses many of the JavaBeans discussed in this chapter.

MapXtreme JavaBeans are easy to use in a visual development environment such as JDeveloper, JBuilder, Visual Age, etc. The components needed to create your mapping applet can be easily dropped onto a form. Once on a form, the properties can be set.

Note: As of this writing Visual Cafe 4.0 and earlier does not fully support Java 2 for drag and drop GUI building using MapXtreme's JavaBeans.You can still compile and run 1.2 programs.

Developing with MapXtreme JavaBeans allows you to create a mapping applet quickly that provides more robust functionality than a regular HTML page with an embedded map can offer. They are best used in intranet environments with powerful servers and networks to handle the increased download time that it takes for an applet to load. For a summary of configuration options using applets, see Chapter 4: Planning Your Application.

Additionally, use the MapXtreme JavaBeans for rapid prototype development.

There are six types of MapXtreme JavaBeans that can be used in your applet or application.

- VisualMapJ Bean
- MapToolBar Bean
- MapTool Beans
- LayerControl Bean
- AddTheme Wizard Bean
- LegendContainer Bean

Each MapXtreme JavaBean is explained in detail in the next sections.

VisualMapJ Bean 

The VisualMapJ Bean is the main mapping component that displays maps within its content pane. It is built on top of the existing MapJ class library. VisualMapJ allows the most common mapping operations to be done in a visual development environment, rather than programmatically. Operations include adding map definitions or geosets, controlling zoom, controlling center, controlling bounds, etc. Less common, more sophisticated actions can be accomplished using the underlying MapJ object from the API.

VisualMapJ displays a map image based on the state maintained by its internal MapJ reference. VisualMapJ maintains a list of registered MapTools, and manages tool selection and deselection. It forwards mouse and keyboard activity to the currently selected Map Tool, and relies on the tool to manipulate its state.

VisualMapJ can be used in conjunction with a MapXtremeServlet, or stand-alone. The **StartupMapDefinition** property allows you to specify a map definition or geoset file to use to initialize VisualMapJ (MapJ). The Layers defined within this file will specify whether data access should be done directly or through an instance of MapXtremeServlet. When VisualMapJ is used within an applet you will want to set up data access through MapXtremeServlet to remain within the security constraints imposed by the applet environment.

The **Map Renderer** property is used to specify whether the map is rendered locally (within the process space of the applet or application embedding VisualMapJ), or remotely through an instance of MapXtremeServlet.

The **ShowToolTips** property specifies whether VisualMapJ will allow popup text to be displayed when a mouse hovers within its extents. It is up to individual MapTools to set the content of the popup text.

Finally,VisualMapJ has **Center** and **Zoom** properties to specify the maps initial bounds.

MapTool Beans

MapTool Beans are a collection of tools that allow you to carry out basic ma navigation and selection operations. Tool operations include pan, recenter, zoom, measure distance, get information and select features at a point or within confining boundaries. A list of tools is provided in the table below.

All of the MapTool Beans extend Swing's Action Interface. This makes them easy to add to a toolbar or menu. If you add the tools to both, they will remain synchronized.

MapTools act on VisualMapJ in the following areas by:

- specifying a tool tip to display
- specifying the cursor to use
- providing custom rendering on top of the map image (e.g., marquee selection box, ruler rubberbanding).
- modifying any properties of MapJ (center, zoom, Layers, Themes)
- prompting VisualMapJ to repaint.

The table below describes the available MapTools.

MapTool		Purpose
PanMapTool		Use to reposition a map within its window by dragging the map.
RecenterMapTool		Use to recenter the map on the clicked point.
RulerMapTool		Use to get distance between two or more points.
InfoMapTool		Use to get attribute information on the clicked point.
ZoomInMapTool		Use to get a closer area view of a map or a layer As user drags mouse, a thin black marquee appears.

MapTool		Purpose
ZoomInMapTool2		Use to get a closer area view of a map or a layer. As user drags mouse, a beveled marquee appears.
ZoomOutMapTool		Use to get a wider view of a map.
ObjectSelectionMapTool		Select individual features at the location of a mouse click.
BoundarySelectionMapTool		Select features contained within the bounding polygon of the unique region feature in the topmost visible layer.
RadiusSelectionMapTool		Select features contained within a bounding circle formed by a mouse drag operation.
RectangleSelectionMapTool		Select features contained within a bounding rectangle formed by a mouse drag operation.
PolygonSelectionMapTool		Select features contained within a bounding polygon formed by a series of mouse click actions.

The Selection MapTools allow users to select Features in multiple layers through individual mouse clicks, and through more complex mouse drag operations. These mouse interactions may create SelectionThemes on a per layer basis to denote the selection that was made with the tool in the VisualMapJ map. For more on SelectionThemes, see Chapter 13.

MapToolBar Bean

You can add your map tools to any Swing JToolbar. There is a special MapToolBar component included with the MapXtreme JavaBeans. By default, the MapToolBar includes the ZoomInMapTool2, ZoomOutMapTool, PanMapTool, RulerMapTool, and InfoMapTool. The toolbar also contains a button to activate the LayerControl Bean. These default items can be removed, and additional tools can be added to the toolbar.

A requirement of using the MapToolBar Bean is that VisualMapJ and the MapToolBar have the same parent component and they must be added in a certain order to allow for proper registry. You should first add a VisualMapJ object to the form and then the MapToolBar Bean. When you add the components in this order, the MapToolBar Bean will register all of its MapTools with the VisualMapJ component. If you add the MapToolBar Bean first, you will need to manually associate the MapTools with VisualMapJ.

Once the toolbar has been added, users can make a tool the currently selected tool by clicking on its corresponding button.

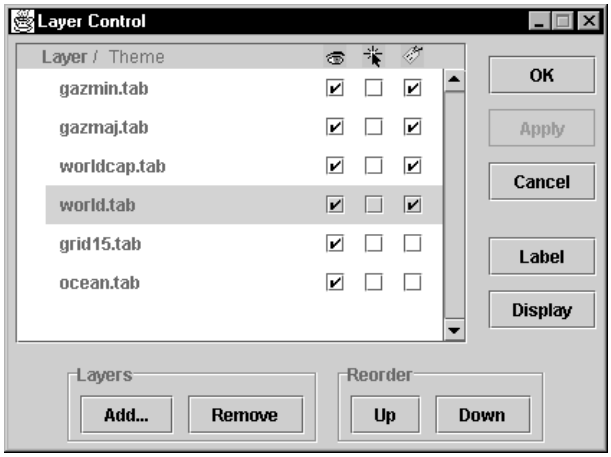
For more information on working with MapTools, see page 85.

LayerControl Bean 

To give your users the ability to control the layers in the map, include the Layer Control Bean in the ToolBar of your application or applet. The Layer Control Bean provides a user interface that allows you to set layer visibility, selectability, and autolabeling, as well as add, remove and reorder layers. Additionally, you can set the layer display characteristics and label properties, and manage themes.

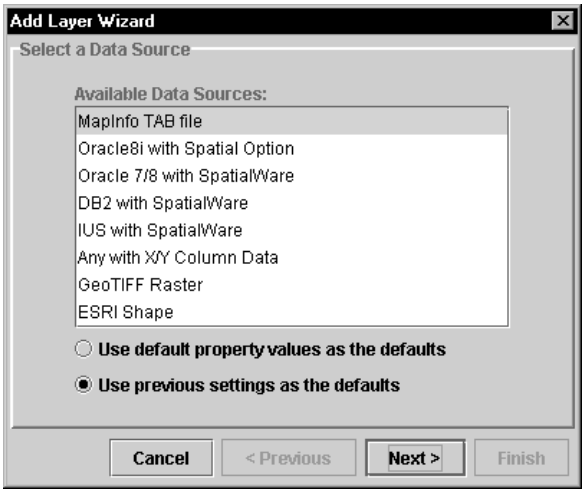
If you include the default MapToolBar in your GUI, you can display the Layer Control by clicking on the Layer Control button.

For a complete description of the elements that make up the Layer Control, see the section “Layer Control Command” on page 234 in Chapter 14: Managing Your Data.



AddLayer Wizard

An important element of the Layer Control is the ability to add layers to your map. By clicking on the Add button, the Add Layer Wizard displays that guides you through the process of adding layers from your data sources, whether they are locally stored tab files or data sources accessed via JDBC.



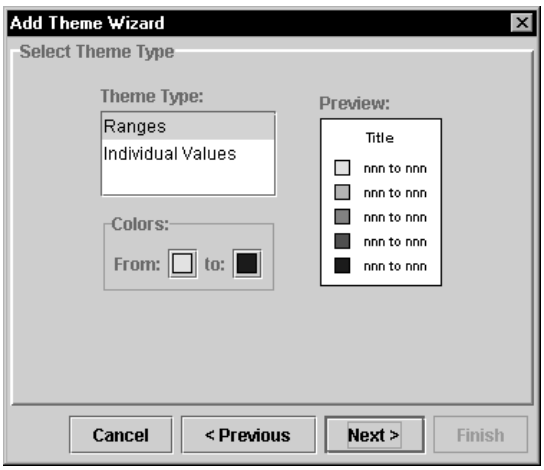
The Add Layer Wizard is configurable to help you help the user add a layer with ease. The configuration information is stored in the **addlayerwizard.properties** file, located in the MapXtreme Java directory where the jar files are found. In the properties file, you can define pre-set values for commonly used settings (e.g., Oracle8*i* host and port). The wizard remembers the last used values for each data provider.Specific data providers and named resources can be added or removed from the wizard through the properties file. Storing passwords in the file is optional (default is to not store passwords).

For more on the addlayerwizard.properties file, see Appendix A: Customizing the Add Layer Wizard via the addlayerproperties.file. For information on named resources, see Chapter 10: Accessing Remote Data.

AddTheme Bean 

The AddTheme Bean is a guided tool to allow you to easily add a Ranged Theme or Individual Value theme to your map. The theme can be based on any supported column and layer in the current map. Currently there is support for creating a theme based on numeric, string and date column data, and on point, line and region layers.

Part of the operation will be to create a default theme legend which is associated with the new theme. The legend can be customized to change the title, fonts, insets, descriptive text and colors.



Through the Wizard Bean the user can choose a name for the theme. For RangedThemes, the user can also choose the number of ranges (bins), distribution method (equal count, equal ranges, etc.), and the value that all break-points (bin upper-bound values) should be rounded to. Users can also specify the rendition (color) of the points, lines or regions that will be thematically shaded by setting the rendition for the first and last ranges. Appropriate renditions will then be computed for the ranges in between.

For IndividualValueThemes, the user can choose which values in a layer should be given a distinct shading to distinguish them from other values.

For information on creating Ranged and Individual Value themes programmatically, see Chapter 13: Theme Mapping and Analysis.

At run-time, the AddTheme Bean hooks up to an existing VisualMapJ instance. In order for this to work, VisualMapJ must be a child of the component that the AddTheme Bean was added to. For example, if the AddTheme Bean is added to a JPanel to which a VisualMapJ instance was already added, then the VisualMapJ instance will be found and hooked up to. If the automated hookup cannot occur, then you must use the setVisualMapJ(VisualMapJ) method of the AddTheme Bean to seed the Bean with a working instance of VisualMapJ.

The AddTheme Bean extends AbstractAction, so it can be added to a JMenu or JToolbar (menu and toolbar will stay synchronized). When the menu item or toolbar button is clicked, the AddTheme Wizard will display.

LegendContainer Bean

As a way of managing how legends are laid out and displayed, MapXtreme Java includes a LegendContainer Bean. It can be dropped in along-side a VisualMapJ object and display any legends that are associated with the Layers in VisualMapJ. If the LegendContainer detects the addition or removal of a theme, it updates its display accordingly.

Legends are Swing JPanels. This allows the legends to be laid out within the LegendContainer and render themselves into a LegendContainer.

Tutorial: Creating an Application with MapXtreme JavaBeans

This section describes the steps to create a stand-alone mapping application using the MapXtreme JavaBeans.

The discussion uses JDeveloper 3.0 as its RAD environment. For other IDE's, use this discussion as a guide, but refer to your IDE documentation for specific instructions.

There are three parts covered in this tutorial:

1. Specifying the location of the MapXtreme jar files.
2. Adding the JavaBeans to JDeveloper
3. Creating the mapping application using JDeveloper IDE.

Specifying the Location of the MapXtreme Jar Files

Before you can reference the MapXtreme Java classes, you must configure your development environment so that it can find the MapXtreme jar files. The precise configuration steps depend on which development environment you use. For example, using JDeveloper 3.x:

1. From the Tools menu, select Default Project Properties
2. Switch to the Paths tab (JDeveloper 3.0) or the Libraries tab (JDeveloper 3.1).
3. Click the Add button to launch the Add Library dialog.
4. Click the New button, and enter the following settings:

   `Name: MapXtreme Java 3.0`

   `Class Path:` Enter a semicolon-separated list[1] of all the jar files installed by MapXtreme (found in the MapXtreme/server directory). Also include in the list a path indicating the location of the MapXtreme Server directory (e.g., c:\mapxtreme\server).

   `Source Path, Doc path: leave blank.`

5. Press OK to save the new "MapXtreme Java 3.0" library entry. The new library should now appear in the Java Libraries list in the Paths tab. Click OK.

1. e.g., c:\mapxtreme\server\mxtj30.jar; c:\mapxtreme\server\devsupp30.jar;c:\mapxtreme\server\dpext30.jar; c:\mapxtreme\server\xml4j_1_1_16.jar; c:\mapxtreme\server\mistyles.jar;c:\mapxtreme\server\mxloc30.jar;c:\mapxtreme\server\classes12.zip;
dpext30.jar is used if you are accessing data in an RDBMS. classes12.zip is necessary for accessing Oracle8*i* data. mxloc30.jar is necessary for non-U.S. locales.

Now that you have made the jar files available to your IDE, references to class names such as "com.mapinfo.mapj.MapJ" can compile successfully.

Adding the JavaBeans to your IDE

Before you can use the MapXtreme JavaBeans in your IDE, you must configure the IDE. To configure JDeveloper 3.0, do the following:

1. If you have not already done so, make the Component Palette visible (from the View menu, choose Toolbar and then make sure the Component Palette choice is checked).
2. Right-click on any of the tabs in the Component Palette, and choose Properties from the context menu. The Palette Properties dialog appears.
3. Click Add to add a new "page" (a new tab name) to the palette. Specify a name such as "MapXtreme Java 3.0."
4. Click OK on the Add Page dialog. The page name "MapXtreme Java 3.0" now appears in the Pages list. Select the MapXtreme Java 3.0 list entry.
5. Click the Add From Archive tab, then click the Browse button to locate the jarfile mxtj30.jar (which is in the /MapXtreme/server directory.) Several MapXtreme class names will appear in the JavaBeans Found In Archive list.
6. Select all of the MapXtreme class names, then click the Install button. Click OK.

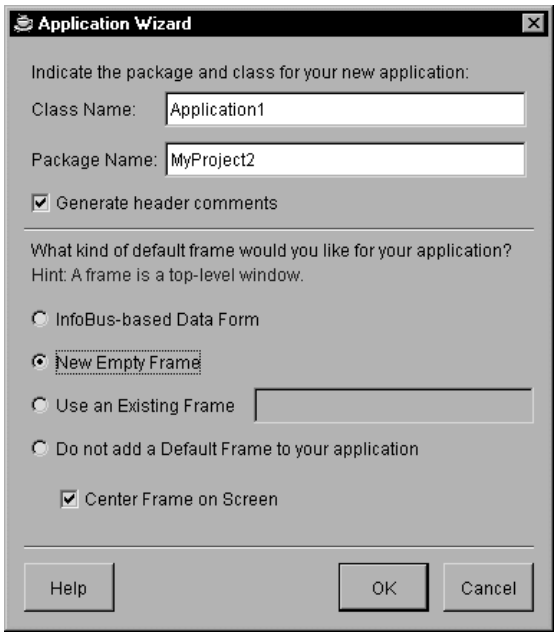
The Component Palette now displays a "MapXtreme Java 3.0" tab which shows you the MapXtreme JavaBeans.

Creating a Mapping Application in JDeveloper

Now that you have set the jar files location and made the JavaBeans available to JDeveloper, you are ready to build your mapping application. Begin by creating a new project:

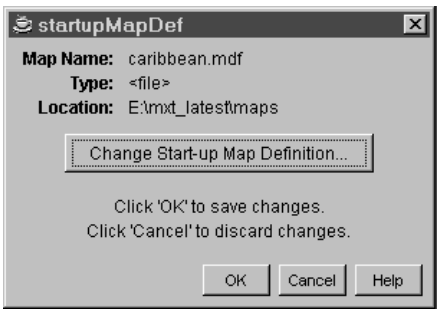
1. In JDeveloper, choose File > New Project. The Project Wizard displays. Click Next.
2. At the Step 1 of 3: Project Type dialog, choose a name for your project. For the Project Type, choose the button "A Project Containing a New..." and choose "Application" from the drop-down menu. Click Next.
3. At the Step 2 of 3: Project Options dialog, accept or change the default settings for the project package, source path, and output directory. Click Next.

4. At the Step 3 of 3: Project Information dialog, fill out the dialog as desired. Click Next to view a summary of your settings or click Finish. The Application Wizard dialog displays.
5. Set the default frame type to "New Empty Frame." Click OK.



6. At the Frame Builder dialog, accept the defaults. Click OK.
7. In the upper left pane of the screen, double-click on the Java file that represents your frame (e.g., Frame1.java). The source code displays in a frame in the center of your screen.
8. To view a blank frame in WYSIWYG mode, click on the tab labeled "Design" along the bottom edge of the source code window.
9. On the Properties tab on the right side of the screen, set the Frame's layout to null by clicking on the layout property (BorderLayout) and choosing Null from the now visible drop-down list.
10. In the "UI" object hierarchy displayed at the lower left part of the screen, click the "JPanel1" object. Then in the Properties tab, change JPanel1's layout property to null.
11. Select the VisualMapJ Bean from the row of buttons on the MapServer tab and draw a rectangle on the blank frame. (To view a tooltip that identifies the name of the Bean, hold the cursor over a toolbar button.)

12. On the Properties tab for the VisualMapJ object, click on the property startupMapDef, then click on the button showing the ellipsis (...) to display the startupMapDef dialog.



To select a map definition, click the Change Start-up Map Definition button. A map definition chooser dialog allows you to choose which map to load; note that you can choose a map definition file from your local filesystem (a .mdf file), or, if you are set up with access to a database such as Oracle, you can load a map definition from that database.

If you have not yet created any map definition files, you can create them by running the Map Definition Manager.

13. Select the MapToolbar button from the MapServer tab and draw a rectangle on the frame. A toolbar with six buttons displays.
14. Choose Run from the Run menu to automatically compile the application and display a map with the Map Toolbar.

    Tip: If you see the exception "java.lang.NoClassDefFoundError: org.w3c.dom.Node" when you try to run the application, it probably means that your project does not include a reference to the XML parser jar file. Check your project properties, and make sure your project includes a library which references xml4j_1_1_16.jar (which is installed into the MapXtreme\server directory).
15. Click on each tool to learn more about their functionality. For example, on the map you can zoom in, zoom out, pan the map, measure distance with the ruler, view information about a map object with the Info tool and view the Layer Control dialog. Layer Control allows you to control the display and labeling characteristics of each layer.

    To load a table that was not in your Map Definition, click the Layer Control button, then click Add.
16. Choose File > Save All to save the project, workspace, and all files.

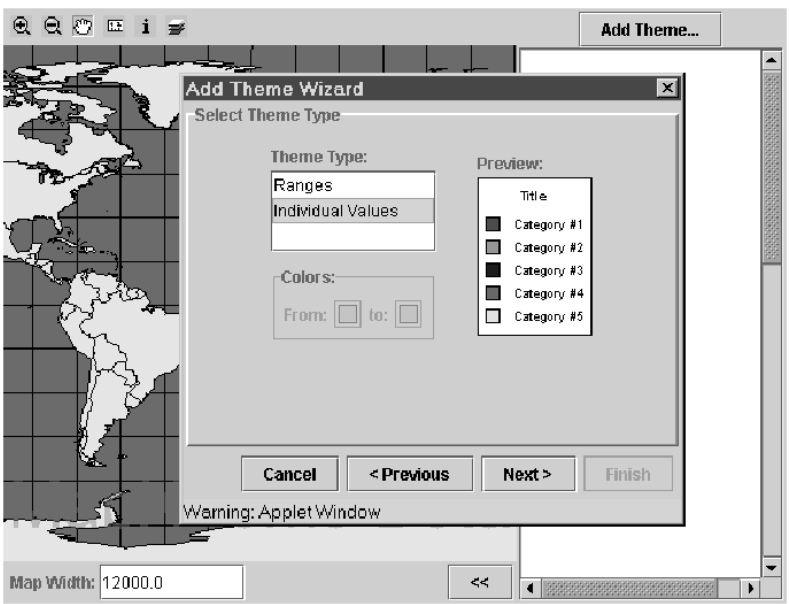
SimpleMap Applet Sample

Provided as one of the sample applications with MapXtreme Java is an example of an applet called SimpleMap that uses the MapXtreme JavaBeans. This applet requires that MapXtreme is running in a web server.

This section is an overview to the sample. For further information on configuring and running SimpleMap, see the readme.html in the \sampleapps\Java\simplemap directory.

SimpleMap is an applet that provides basic map-viewing capabilities to:

- Display a map (using the MapXtreme JavaBean, VisualMapJ).
- Display a toolbar containing tools for map navigation, an Info tool, and a Layer Control tool (using the MapToolbar Bean).
- Allow the user to add thematic shading by clicking an Add Theme button (using the AddTheme Bean).
- Display a corresponding theme legend in a panel to the right of the map (using the LegendContainer Bean).
- Provide a text field that displays the current map zoom, and allows the user to type in a new zoom width.



The SimpleMap sample consists of following files:

- Readme.html – Document describing the applet and steps to take to deploy it.
- SimpleMap.java – The main code module. The source code is very simple, because it uses MapXtreme Beans – specifically, the VisualMapJ Bean, Map Toolbar Bean and AddTheme Bean – to do most of the work.
- SimpleMapBeanInfo.java – A code module provided for users running the Java Plug-in with Internet Explorer. This is not actually used by the applet; however, some users report that when Java Plug-in is used with Internet Explorer, IE attempts to load <classname>BeanInfo.class; if the class is not present, IE may not to load the applet.
- simplemap.html – A sample HTML file that shows how to run the applet using Sun's Java Plug-in. Instead of using the <APPLET> tag, this HTML page uses the <OBJECT> and <EMBED> tags that are used by Sun's Java Plug-In. If you do not already have the Java Plug-In installed, viewing this HTML page will prompt you to download the Plug-in.

SimpleMap Applet Setup

This is a summary of the procedure you will need to carry out to set up SimpleMap. Refer to the readme.html in the SimpleMap directory for complete details.

1. Configure and launch the servlet container used to run MapXtremeServlet. (Chapter 3: Getting Started, provides steps.)
2. Compile the applet.
3. Copy simplemap*.class files, simplemap.html, and the MapXtreme jar files to a directory that is accessible through your web server.
4. Modify the tags in simplemap.html to specify values appropriate for your web server.
5. Load the applet by opening the HTML file in your browser.

Keep in mind the following must be configured properly for SimpleMap to run:

- The MapXtremeServlet must be running (i.e., your servlet container must be running).
- The applet must be compiled correctly.
- The web browser must have Java Plug-in installed.
- The HTML file used to load the applet must specify appropriate param tags.

Map Definitions and SimpleMap

If you configure SimpleMap to load a Map Definition, make sure the map definition was constructed using the "Access Data Via Remote MapXtremeServlet" option in the Add Layer Wizard dialog. If an attempt is made to access local resources, a security exception will occur. For more on creating map definitions, see Chapter 14: Managing Your Data.

Including MapTools in Your Applet or Application

Most mapping applications allow the user to perform various tasks by interacting with the map with mouse actions (such as clicking and perhaps dragging). If you want to support this type of map interaction, you will want to add map tools to your application or applet.

If your development environment supports visual GUI design, you can create and manipulate GUI elements, such as toolbars and tools, visually.For example, a JDeveloper user can draw GUI elements onto a form when in "design mode." If your development environment does not support visual GUI building, you will need to create map tools by hand, by typing in code, as shown below.

Creating a MapToolBar

You can add a set of map tools to your code very quickly and easily by instantiating a MapToolBar object, and adding it to your layout. The default MapToolBar object provides several common map tools, including Zoom In, Zoom Out, Pan, Info and Ruler, as well as a button that launches the Layer Control dialog.

You can instantiate a MapToolBar very simply, with no arguments:

```
MapToolBar mapToolBar1 = new MapToolBar();
```

Or, if you want to make major changes to the set of map tools included in the tool bar, you can use the other MapToolBar constructor, which takes a Vector of map tools.

Once you have created the MapToolBar, add it to the same container to which you added the VisualMapJ object.

```
this.getContentPane().add(visualMapJ1,
   BorderLayout.CENTER);

this.getContentPane().add(mapToolBar1,
   BorderLayout.NORTH);
```

Associating MapTools with the Map

The tools in the MapToolBar must be associated with the VisualMapJ object. If they are not properly associated, then the tools will not have any effect on the map.

The simplest way of associating the tools with the VisualMapJ object is to add the MapToolBar object to the same container (e.g. the same JPanel) where you placed VisualMapJ. When the MapToolBar and VisualMapJ objects are in the same container, the tools are automatically associated with the map.

If your map tools are not automatically associated with the map (for example, because your GUI layout required that you place the toolbar in a different JPanel), you will need to call your VisualMapJ object's add method, which registers a map tool with the map; see below.

Adding Individual Tools

If you find that you need more map tools than are provided with the default MapToolBar object, you can add more tools. For example, the default MapToolBar object does not include selection tools, such as the RadiusSelectionMapTool, so you might want to add that tool to the toolbar. To add individual tools to the toolbar:

1. Create any tool objects that you plan to use.

   ```
   RadiusSelectionMapTool radTool = new
   RadiusSelectionMapTool();
   ```

2. If necessary, call VisualMapJ's add method to associate the tool with the map. (As described above, this step is not necessary if your MapToolBar is on the same container as the VisualMapJ.)

   ```
   myVisualMapJ.add(radTool); // may not be required...
   ```

3. Add your tools and/or your toolbar to your GUI. Typically, you add your tools to a MapToolBar, and add that MapToolBar to the same component (e.g. the same JPanel) where you added your VisualMapJ object.

   ```
   mapToolBar1.add(radTool);
   ```

Note that the order of statements is important. If you want your map tools to be automatically associated with the map, you should create your map tools, add them to your MapToolBar, and then add your MapToolBar to the same container as the VisualMapJ, in that order.

You can also add map tools to a menu (e.g. by calling a JMenu object's add method). For an example of how map tools appear when on a menu, see the Map Definition Manager, which displays tools on its Map menu. If you have added tools to both a tool bar and a menu, the tool bar and menu are kept in sync; when you select a tool on the toolbar, the same tool appears selected on the menu.

Removing Individual Tools

You can remove individual tools from the MapToolBar by calling its **removeMapTool** method. For example, if you want to use the default MapToolBar object, except that you do not want to include a Ruler tool, you could remove the ruler tool as follows:

```
for (Enumeration e = mapToolBar1.getMapTools();
   e.hasMoreElements(); ) {
MapTool mt = (MapTool) e.nextElement();
 if (mt instanceof RulerMapTool) {
  mapToolBar1.removeMapTool(mt);
  break;
 }
}
```

Alternately, you could build aVector containing only the map tools you do want to use, then pass that Vector to the MapToolBar constructor.

Configuring and Controlling the Standard Map Tools

Once you have included a map tool in your application, you may want to adjust how the tool behaves. Some examples of configuration options include:

- You could disable a map tool.
- You could reconfigure the InfoMapTool to not display its secondary "Info Tool" window.
- You could control whether the RadiusSelectionMapTool saves a circle in an annotation layer.

As a general rule, you control the behavior of a map tool by setting its properties. Map tool properties that you can set are described below.If you are using a visual development environment that supports property sheets the tool properties can be set

by using the appropriate property sheet editors. Otherwise you can adjust the properties programmatically by calling a property's appropriate set method. For example, to configure the enabled property, you would call the setEnabled method.

General Settings

All map tools share some common properties:

- **enabled** – To enable or disable a tool, call its **setEnabled** method.
- **cursor** – To assign a custom cursor, call the **setCursor** method.
- **selected** – To select a tool, call its **setSelected** method. When your application runs, the tool will appear selected automatically. However, this will not stop the user from selecting a different tool.

  ```
  radTool.setSelected(true);
  ```

Zoom In / Zoom Out Settings

The three zoom tools (ZoomInMapTool, ZoomInMapTool2, and ZoomOutMapTool) can be customized with the following properties:

- **zoomFactor** – this property controls how far in or out the map is zoomed when the user clicks the map. The default setting is 2; in other words, if your map shows an area 500 miles wide, and you click to zoom out, the resulting map will show an area 1000 miles wide.
- **zoomMode** – controls whether the map is recentered at the point where the user clicked, or whether the map is simply zoomed in or out, while remaining centered at the same location.

RulerMapTool Settings

The Ruler tool provides the following properties:

- **calculationMetho**d – controls whether distances are measured using Cartesian or Spherical calculations.
- **distanceUnit** – specifies the unit, such as miles or kilometers, used for displaying distances.
- **distanceWindowVisible** – controls whether the Ruler tool opens a secondary window to display distance measurements. To suppress the secondary window, call setDistanceWindowVisible(False).

The RulerMapTool also has a bound "Distance" property so that listeners can be notified as the distance changes.

InfoMapTool Settings

The Info tool provides the following properties:

- **searchMode** – controls which layers the InfoMapTool searches. For example, to search all map layers, call:

  ```
  myInfoMapTool.setSearchMode(SearchMode.FULL);
  ```

- **infoWindowVisible** – controls whether a secondary "Info Tool" window is opened to display the text returned by the Info tool. In some cases, you might want to suppress the secondary window, and display the resulting text somewhere else in your GUI.To prevent the Info tool from displaying its secondary window, call:

  ```
  setInfoWindowVisible(false)
  ```

Note: Once you have suppressed the InfoTool window, you will want to display the results in another place. Create an object that implements the InfoObtainedListener interface, (meaning that the object provides an infoObtained method), then register your listener with the InfoMapTool. When the user selects the InfoMapTool and clicks the map, your infoObtained method will be called, and an InfoObtainedEvent object will be passed in. You can use the methods on the InfoObtainedEvent object to extract the information returned by the InfoMapTool. It is then up to you to display that information somewhere in your GUI.

SelectionMapTool Settings

MapXtreme provides several selection tools: BoundarySelectionMapTool, PolygonSelectionMapTool, RadiusSelectionMapTool, RectangleSelectionMapTool, and ObjectSelectionMapTool. If you include one or more of these tools in your application, the user will be able to choose the selection tool, then click (or, depending on the tool, click and drag) on the map to select one or more features on the map.

The selection map tools have several properties that you can configure, as follows. (Not all properties apply to the ObjectSelectionMapTool.)

- **saveAnnotation** – all selection tools except for ObjectSelectionMapTool allow you to save annotations. For example, the RadiusSelectionMapTool can save a circle annotation representing the search radius that the user selected.
- **searchType** – controls the search criteria that are used to determine whether a feature should be selected. For example, if a region is only partially inside a search radius, you might or might not want the region to be selected. To

control this option, call the selection tool's setSearchType method, and specify one of these values:

SearchType.mbr – returns features whose minimum bounding rectangle intersects the search region. This is the least restrictive of the search types, and returns the maximum number of features.

SearchType.partial – returns features that intersect the search region, at least partially.

SearchType.entire – returns features that are completely contained within the search region (default); this is the most restrictive search type.

- **selectionMode** – this property controls how many layers are searched. To control which layers are searched, call setSelectionMode, and specify one of the following:

  SelectionMode.VISIBLE – searches all selectable layers that are currently visible. This option skips layers that are currently invisible due to zoom layering.

  SelectionMode.FULL – searches all selectable layers.

  SelectionMode.FIRST – searches all visible and selectable layers; however, the search stops as soon as one layer has features within the search area.

- **selectionThemeType** – controls whether a SelectionTheme is created to highlight selected features, and also controls how the theme assigns colors to the selected features. To control the shading of the selected features, call setSelectionThemeType with one of these options:

  SelectionMapTool.SELECTIONTHEME_OVERRIDE – the selected features will be shaded, and the shading will use the color that was explicitly specified by the themeOverrideColor property.

  SelectionMapTool.SELECTIONTHEME_NONE – no theme will be created; the appearance of the selected features will not change after being selected.

  SelectionMapTool.SELECTIONTHEME_DEFAULT – selected features will be shaded using a color that is the inverse of the original feature color

- **themeOverrideColor** – Sets the override color (the color value used to display selected features when and if selectionThemeType is set to SELECTIONTHEME_OVERRIDE). For example, if you want all selected features to appear in green, call setThemeOverrideColor(Color.green), and also call setSelectionThemeType (SelectionMapTool.SELECTIONTHEME_OVERRIDE).

- **coordinateType** – the Radius and Rectangle selection map tools allow you to specify whether the selection area (the circle or rectangle drawn by the user) should use screen coordinates or map coordinates. Set this property to one of the following values:

  ConfiningSelectionMapTool.COORD_MAP – radius and rectangle search tools will create features in "map coordinates" (i.e. a radius may not look circular on the screen, depending on the map projection, just as the earth itself does not appear round in some map projections, but the radius will accurately represent the total area within a specified distance of a center point).

  ConfiningSelectionMapTool.COORD_SCREEN – radius and rectangle search tools will create features in "screen coordinates" (e.g. a radius will look circular on the screen).

All SelectionMapTools fire SelectionToolEvents when a selection occurs, so further customization is possible if you create and register a SelectionToolListener.

Creating Custom MapTools

If you need functionality that is not provided by any of the standard MapXtreme map tools, you can create your own custom tools. To define a custom tool, you must implement your own Java class which satisfies the MapTool interface. Once you have created this class, you can instantiate your custom tool, and add it to your application, the same way that you added standard tools (such as the Radius tool example shown above).

Custom Tool Example: SimpleRulerMapTool

An example of how to create a custom tool is provided in the \sampleapps\beans directory, which contains the following modules:

- **SimpleRulerMapTool.java** – an example of a custom map tool. This class implements a customized ruler tool, for measuring distances on the map. Unlike the standard Ruler tool, this custom tool displays the current distance as a ToolTip when you stop moving the mouse.
- **SimpleRulerMapToolBeanInfo.java** – a BeanInfo class, which controls how the tool's properties are exposed (e.g. when you inspect properties using a visual development environment).
- **SimpleToolFrame.java** – a class that extends JFrame. This is the class that displays a map and a toolbar. This is also the place where we instantiate the SimpleRulerMapTool object.
- **SimpleToolApp.java** – a very simple class that provides the main method.

Before studying the source code that defines the SimpleRulerMapTool, you may want to familiarize yourself with how the tool behaves. You can see the custom tool in action by compiling all four java classes, then running the SimpleToolApp class. NOTE: Before compiling, you may need to edit SimpleToolFrame.java to change the filename and/or path specified in the loadGeoset call, so that they identify the name and location of a file on your system.

When the application runs, the custom SimpleRulerMapTool appears at the end of the toolbar. The user can select the tool, click the map once to start measuring a distance, move the mouse, then pause; when the user pauses, the current distance is displayed as a ToolTip. The user can click again to start measuring from a different location. The user can cancel the current line segment altogether by pressing the ESC key.

Basic MapTool Requirements

Map tools are **Action** objects, so that they can be added to JMenu orJToolBar objects. Since map tools are actions, any custom map tool that you create must define an **actionPerformed** method. Whenever the user selects your custom tool, either from the toolbar or the menu, your custom tool class's actionPerformed method is called.

Also, any custom map tool class that you create must implement the **MapTool** interface (com.mapinfo.beans.tools.MapTool). This means that you must provide the following methods:

- **getCursor** – Returns the java.awt.Cursor object associated with this MapTool.
- **isSelected** – Returns True or False to indicate whether this MapTool is currently selected.
- **setCursor** – Sets the tool's Cursor object.
- **setSelected** – Sets whether the tool is currently selected. Note that selecting one tool automatically de-selects the previously selected map tool.

For examples of these methods, see SimpleRulerMapTool.java.

Once you have built a custom tool that addresses these basic requirements, you might consider adding more functionality to your custom tool.Your map tool can implement some, all, or none of the following types of behavior:

- Responding to click and drag actions
- Responding to keyboard input
- Displaying ToolTip Text
- Drawing on top of the map

Each of these areas is described below.

Responding to Click and Drag Actions

Since map tools allow the user to interact with the map, most map tools implement the **MapMouseListener** interface (com.mapinfo.beans.vmapj.MapMouseListener). Implementing this interface means providing the following methods, which correspond to various mouse actions:

- **mouseClicked** – called when the user completes a Click action (pressing and releasing the mouse button)
- **mouseDragged** – called when the user moves the mouse while holding down the mouse button

- **mouseEntered** – called when the mouse enters the VisualMapJ display area
- **mouseExited** – called when the mouse exists the VisualMapJ display area
- **mouseMoved** – called when the mouse moves, and the mouse button is not being held down
- **mousePressed** – called when the user presses down on the mouse button
- **mouseReleased** – called when the user releases the mouse button

The SimpleRulerMapTool class demonstrates how some of these mouse event methods are used. This tool allows the user to click to set the starting point for measuring a distance; therefore, the **mouseClicked** method contains code that stores starting coordinates.

Next, the user can move the mouse, causing the SimpleMapRulerTool to display a rubberband line. To produce this rubberband effect, the **mouseDragged** and **mouseMoved** methods both call the **handleStretch** method.

In the SimpleRulerMapTool example, the other mouse methods are included only to satisfy the MapMouseListener interface; therefore, those methods are empty. For example:

```
public void mouseEntered(MapMouseEvent e){}
```

Depending on how you want your custom tools to behave, you may want to add code to those empty methods.

Responding to Keyboard Input

In some cases, you may want your custom map tool to handle keyboard input. For example, the SimpleMapRulerTool allows the user to press the ESC key to cancel the display of the rubberband line.

If your custom map tool needs to respond to keystrokes, implement the **KeyListener** interface (java.awt.event.KeyListener), which entails implementing the following methods:

- **keyPressed** – called when the user presses down on a key
- **keyReleased** – called when the user releases a key
- **keyTyped** – called when the user finishes typing a key (pressing and releasing a key)

The SimpleMapRulerTool checks for the ESC key in the keyReleased method. If it was pressed, the keyReleased method calls the cleanup method, which resets the tool.

In the SimpleRulerMapTool example, the keyPressed and keyTyped methods are included only to satisfy the KeyListener interface, so those methods are empty.

Displaying ToolTip Text

Map tools can have two different types of ToolTip text:

- When the user positions the mouse over the tool on the toolbar, a ToolTip appears, displaying the name of the tool. To set the text that appears in this type of ToolTip, call putValue; for example:

  ```
  putValue(SHORT_DESCRIPTION,"Ruler");
  ```

- When the user has selected your custom tool, and the cursor is over the map, you can display ToolTip text. This is accomplished by implementing the **ToolTipTextSetter** interface (com.mapinfo.beans.tools.ToolTipTextSetter) by providing a getToolTipText method.

For example, the SimpleMapRulerTool uses ToolTip text to display the distance between the current cursor location and the location where the user last clicked. Accordingly, SimpleMapRulerTool.java provides a getToolTipText method, which simply returns a string representing a distance measurement (e.g. "123.45 mi").

NOTE: Even if your custom tool implements the ToolTipTextSetter interface, you will not see ToolTips unless you call the VisualMapJ object's setShowToolTips method. For an example, see SimpleToolFrame.java, which makes the following call:

```
visualMapJ1.setShowToolTips(true);
```

Drawing on Top of the Map

Some map tools make temporary changes to the appearance of the map while the user is using the tool. To draw on the map in this manner, implement the MapPainter interface (com.mapinfo.beans.vmapj.MapPainter), and its **paintOnMap** method.

For example, the SimpleMapRulerTool draws a rubberband line on top of the map, and updates this line as the user moves the mouse within its paintOnMap method.