This chapter covers these topics:
- Introduction to Developer/2000 for the Web
- Benefits of Developer/2000 for the Web
- Forms Web Architecture
- Reports Web Architecture

1.1 Introduction to Developer/2000 for the Web

Developer/2000 for the Web is a new generation of Oracle development tools that enable you to deploy new and existing applications on the World Wide Web, either on an internal company intranet, or on the Internet. Developer/2000 for the Web takes advantage of the ease and accessibility of the Web, and elevates it from a static information-publishing mechanism to an environment capable of supporting complex, dynamic applications.

1.1.1 The best of the Web and client-server

Developer/2000 for the Web provides solutions that enable you to leverage the benefits of the Web, while maintaining the strengths of client-server computing. The Web greatly reduces the costs of administering and maintaining applications, while allowing for a thin, low-cost client. At the same time, it allows you to leverage your existing client-server applications, which may be mission-critical applications that support—and often drive—your business practices. These applications must be scalable to large numbers of users and open to all of your environments.
1.1.2 Three-tiered architecture

In most client-server implementations today, running applications is a highly client-intensive process. Though data is extracted from a remote database server, applications run on client machines, which often have limited processing power and memory capacity.

Developer/2000 for the Web supports a three-tiered architecture that delivers the benefits of both client-server and the Web in a single application. In a Web implementation, application logic and processing are focused on a middle tier of application servers instead of on desktop client machines.

<table>
<thead>
<tr>
<th>Tier</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>front-end</td>
<td>any number of client desktop machines</td>
</tr>
<tr>
<td>middle</td>
<td>one or more application servers</td>
</tr>
<tr>
<td>back-end</td>
<td>one or more database servers</td>
</tr>
</tbody>
</table>

![Figure 1–1 Three-tiered architecture](image)
1.2 Benefits of Developer/2000 for the Web

Developer/2000 for the Web benefits groups across the spectrum of the application life-cycle, including managers, developers, and end users. Each group benefits by gains in productivity, reduced resource requirements, and lower costs:

- **Low-cost deployment and maintenance** Web applications are deployed and maintained on centralized application servers, from which they download to end users’ Web browsers at runtime. To roll out a Web application, simply give end users the application’s URL. This distribution method reduces the time, cost, and complexity of deploying applications to a large or geographically-dispersed end-user base, all without installing application software on their desktop machines.

- **Client-server migration** Developer/2000 for the Web extends and leverages the traditional strength of Oracle’s client-server development suite. Developer/2000 for the Web supports all GUI widgets available in a client-server application, so you can Web-deploy any existing client-server application without changing the form definition (.FMB file)\(^1\). You can run the same Forms executable (generated .FMX file) in client-server mode or on the Web. Programmers experienced with Developer/2000 do not need to learn Java, JavaScript, or any other new language.

- **“Thin client” architecture** Any end user with a Java-enabled\(^2\) Web browser can run Web applications. This single client-side requirement greatly reduces processor and memory requirements for a user’s desktop machine.

- **Support for various business objectives** With Developer/2000 for the Web, you can develop internal, transaction-intensive applications (deployed on a company intranet), and electronic commerce applications (deployed on the Internet). End users accustomed to client-server applications will enjoy similar performance with Developer/2000 for the Web.

- **NC support** Developer/2000 for the Web is designed in accordance with Oracle’s *Network Computing Architecture (NCA)*. Developer/2000 applications run as NCA-compliant Web cartridges.

---

\(^1\) See Section 3.2.8, “Feature restrictions for Forms on the Web.”

\(^2\) Java 1.1.2 or higher
Platform independence  Developer/2000 for the Web leverages Java—the emerging standard platform-independent language—to map GUI widgets to their native counterparts on other platforms. The look and feel of widgets will differ only slightly between end user platforms (Windows, Mac, Motif, and so on).

Dynamic reporting and Web authoring  With no additional training, you can use the declarative, point-and-click interface of Developer/2000 Reports to create Web reports. To Web-enable existing reports, simply regenerate them in HTML or PDF. Use the CGI interface to set up reports to run dynamically from a Web browser, and to author Web pages on the fly.

Drill-down reports  Developer/2000’s Web-publishing capabilities include reporting features that enable end users to navigate through layers of a particular report. You can associate a URL with a field in a report; end users then can click the field to navigate to another Web page. The new page displays detailed report data corresponding to the clicked field.
1.3 Forms Web Architecture

To run new or existing Forms applications on the Web, you should install and operate Developer/2000 for the Web on the middle tier of a distributed three-tiered architecture (see Figure 1–1).

1.3.1 About the Forms Client and Forms Server


1.3.1.1 Forms Client

The Forms Client is a Java applet—downloaded at runtime from an application server to an end user’s Web browser—that displays the form’s user interface and manages interaction between end users and the Forms Server. The Forms Client receives “bundles” of interface commands from the Forms Server and translates them (in sets) into interface objects for the end user. Some interface events handled by the Forms Runtime Engine in a client-server implementation (such as typing characters in a text field, or moving around a dialog) occur only on the Forms Client in the Web implementation, with no interaction with the Forms Server Runtime Engine.

The Forms Client is:

- **Generic** You are not required to deploy a separate Java applet for each application you wish to deploy on the Web.
- **Dynamic** The Forms Client dynamically reacts to the current form at runtime, requesting and displaying only the information and user interface elements necessary to represent the current state of the application at any given time.
- **Feature-rich** The Forms Client supports all user interface widgets and tools available in a client-server implementation. Due to Java object standards, the look and feel of some Forms widgets may vary slightly when deployed on the Web. For more information, refer to Section 3.2.8, “Feature restrictions for Forms on the Web.”
- **Thin** At startup, only those class files necessary to render the initial state of an application are downloaded to the end user’s machine. Additional class files are downloaded dynamically (as needed) to support additional user interface functionality.
1.3.1.2 Forms Server

The Forms Server consists of two components:

- **Listener** The Forms Server Listener initiates the Forms runtime session and establishes a connection between the Forms Client and the Forms Server Runtime Engine.

- **Runtime Engine** The Forms Server Runtime Engine is a modified version of the Forms 4.5 Runtime Engine, with user interface functionality redirected to the Forms Client. It handles all form functionality except UI interaction, including trigger and commit processing, record management, and general database interaction.
1.3.1.3 Client-server architecture vs. Web architecture

There is a significant difference between deploying a Forms applications in client-server mode and deploying the same application on the Web:

- **Client-server** The Forms Runtime Engine (and all application logic) are installed on end users’ desktop machines. Although your application can include database-server-side triggers and logic, typically all user interface and trigger processing occurs on client machines.

![Diagram of Client-Server Architecture](image-url)

*Figure 1-2  Forms: client-server architecture*
Web The Forms Server Runtime Engine (and all application logic) are installed on application servers, not on client machines. All trigger processing occurs on database and application servers, while user interface processing occurs on the Forms Client.

Figure 1–3 Forms: Web architecture
1.3.1.4 Call and response: Forms Client and Forms Server

Once a direct network connection is established between the Forms Client and Forms Server, the two components communicate through a series of requests and responses—via compressed messages passed over a network.

Requests from the Forms Client are events (such as “click button” or “display LOV”). Responses from the Forms Server are a series of changes to the user interface (such as value changes, and adding and removing components), all of which the Forms Client converts to display objects.

For example, the Forms Client might receive a response from the Forms Server similar to “create four green text items on canvas CAN_12.” The Forms Client translates the response into actual interface objects, in this case, the verdant text items.

The Forms Client contacts the Forms Server when users perform:

- high-level operations (such as accepting or canceling a dialog)
- operations (such as checking a checkbox or navigating between fields) that involve validation processing and cause default and user-defined triggers to fire
1.3.2 Running Forms applications on the Web

To start and run a Forms application on the Web, end users use a Java-enabled Web browser to access a URL. The following sequence occurs automatically:

1. The URL corresponds either to a static (non-cartridge) HTML page, or to an application cartridge, residing on the application server.

2. An HTML page, and then the Forms Client applet, are downloaded from the application server to the user’s browser.

3. The Forms Client sends a request to the Forms Server Listener (which resides on a specific port of the machine from which the Forms Client was downloaded).

---

Figure 1–4 Running Forms applications on the Web: process flow

To start and run a Forms application on the Web, end users use a Java-enabled Web browser to access a URL. The following sequence occurs automatically:

1. The URL corresponds either to a static (non-cartridge) HTML page, or to an application cartridge, residing on the application server.

2. An HTML page, and then the Forms Client applet, are downloaded from the application server to the user’s browser.

3. The Forms Client sends a request to the Forms Server Listener (which resides on a specific port of the machine from which the Forms Client was downloaded).
4. The Listener contacts the Forms Server Runtime Engine and connects to a Forms runtime process (either by starting a new process, or by connecting to an existing process). If included in the HTML page, Forms command-line parameters (such as form name, user ID and password, database SID, menu name, and so on) and any user-defined Forms parameters are passed to the process by the Listener.

5. The Listener establishes a direct socket connection with the Runtime Engine, and sends the socket information to the Forms Client. The Forms Client then establishes a direct socket connection with the Runtime Engine. The Forms Client and Runtime Engine then communicate directly, freeing the Listener to accept startup requests from other end users. The Forms Client displays the application’s user interface in an applet window outside the main window of the end user’s Web browser.

6. As in a client-server implementation, the Runtime Engine communicates directly with the database through SQL*Net (or another driver, for non-Oracle datasources).

1.3.3 Security and encryption

Data passed between the database, the Forms Server, and the Forms Client is automatically encrypted before—and decrypted after—transmission by the following protocols:

- **RSA RC4 40-bit encryption** (for transmissions between the Forms Client and the Forms Server)
- **SQL*Net SNS/ANO** (for transmissions between the Forms Server and the database server)

Encryption is enabled by default, but can be disabled. To disable encryption between:

- the Forms Client and Forms Server, set the environment variable `FORMS45_MESSAGE_ENCRYPTION` to `FALSE`.
- the Forms Server and the database, refer to the *Oracle Advanced Networking Option Administrator’s Guide*. 
1.4 Reports Web Architecture

With traditional two-tiered, client-server architecture, displaying a report on the Web involves generating HTML or PDF output on the client, moving the file to a location on the server, and linking it to a Web page where users can access it from a Web browser. Automating the process with a third-party scheduling tool or operating system facility makes this a reliable method of publishing and updating reports on the Web.

There may be cases, however, when it is necessary to provide end users with the most current data, on demand. With the three-tiered architecture of Developer/2000 for the Web, end users can run reports dynamically on a remote application server from any client.

1.4.1 Dynamic Reporting with the Web Common Gateway Interface (CGI)

To enable end users to generate reports dynamically from a Web browser, Developer/2000 for the Web provides a CGI interface. R25CGI—the CGI equivalent of the Reports Runtime Engine—facilitates three-tiered, server-based report generation and easy communication with the Web server.
1.4.2 Running Reports applications on the Web

Reports Web Architecture

Figure 1–5 Running Reports on the Web: process flow

To start a Reports application on the Web, end users point to a URL from their Web browsers. The following sequence then occurs:

1. The Web browser passes the URL to the Web server, and the Web server sends the request to the CGI Runtime Engine.

2. The CGI Runtime Engine parses the request and converts it to a runtime command line. Then it executes the command line, retrieves the data from the database through SQL*Net, and generates the output in the Web format (HTML or PDF) specified.

3. The CGI Runtime Engine retrieves the report name and passes the report’s URL to the Web server.

4. The Web server executes HTTP redirection, passing the new URL back to the Web browser, and the report is displayed on the client.
Setting Up Developer/2000 for the Web

This chapter covers these topics:
- Installing Developer/2000 for the Web
- Configuring Forms for the Web
- Configuring Reports for the Web

2.1 Installing Developer/2000 for the Web

Before you design, generate, and deploy applications on the Web, you must install Developer/2000 for the Web.

To install Developer/2000 for the Web:

1. Use the Oracle Installer to install the software onto your application server.

   The Oracle Installer automatically places the required files and programs in the appropriate directories on your application server, and sets environment variables to the appropriate values.

   For information on using the Oracle Installer, refer to the Developer/2000 Release 1.4W Installation Guide for your operating system.
2. Ensure that the operating systems and software installed on your servers and client machines meet the version requirements listed below.

**Note:** Not all software listed below is required; the version requirements are provided for information only. For example, if you don’t plan to create cartridge-based Forms applications for the Web, you do not need to install Oracle Web Request Broker.

<table>
<thead>
<tr>
<th>Tier</th>
<th>Component</th>
<th>Brand / Version Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>back-end (database server)</td>
<td>RDBMS</td>
<td>• Oracle7 Server (7.1 or higher)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• ODBC-compliant RDBMS</td>
</tr>
<tr>
<td>middle (application server)</td>
<td>operating system</td>
<td>• Microsoft Windows NT server (4.0 or higher)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Sun Solaris (2.4 or higher)</td>
</tr>
<tr>
<td></td>
<td>Web server</td>
<td>any</td>
</tr>
<tr>
<td></td>
<td>Developer/2000</td>
<td>• Forms (4.5.8 or higher)</td>
</tr>
<tr>
<td></td>
<td>components</td>
<td>• Graphics (2.5.7.1.0 or higher)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Reports (2.5.5.2.0 or higher)</td>
</tr>
<tr>
<td></td>
<td>Other Oracle software</td>
<td>• Designer/2000 (1.3.2 or higher)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• SQL*Net (2.3.2.1.6A or higher)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Web Request Broker (2.1 or higher)&lt;sup&gt;1&lt;/sup&gt;</td>
</tr>
<tr>
<td>front-end (client machine)</td>
<td>operating system</td>
<td>any</td>
</tr>
<tr>
<td></td>
<td>Web browser</td>
<td>Java-enabled&lt;sup&gt;2&lt;/sup&gt;</td>
</tr>
</tbody>
</table>

<sup>1</sup> Required for cartridge-based Forms applications.

<sup>2</sup> Java 1.1.2 or higher.
3. Ensure that your application server meets the recommended memory levels.

Memory recommendations are based on the size of applications you wish to deploy on the Web.

<table>
<thead>
<tr>
<th>Application Size</th>
<th>Recommended Levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 to 6 MB¹</td>
<td>RAM 3 to 4 MB per user</td>
</tr>
<tr>
<td></td>
<td>Swap Space 12 MB per user</td>
</tr>
</tbody>
</table>

¹ Total size includes all .FMX and .PLL files, with each .FMX file between 100KB and 1MB.
2.2 Configuring Forms for the Web

When you have installed the software onto your application server, you must configure Developer/2000 for the Web, your Web server, and Oracle Web Request Broker\(^1\) in order to deploy Forms applications on the Web.

This section covers these topics:

- Step 1: Generating and deploying .FMX files
- Step 2: Starting and stopping the Forms Server Listener
- Step 3: Making applications available to end users
- Step 4: Configuring the Forms Client

**Note:** If, after installation and configuration, you experience problems running Forms applications on the Web, refer to Section 2.2.5, “Troubleshooting your Forms configuration.”

\(^1\) Required only for cartridge implementation.
2.2 Configuring Forms for the Web

2.2.1 Step 1: Generating and deploying .FMX files

Once you have developed a form definition (.FMB file), and you want to deploy the form on the Web, you must do the following:

1. Generate a Forms runtime executable (.FMX file).
   You must generate .FMX files on the same platform as the application server on which you will deploy them. For example, if your application server’s operating system is Sun Solaris, you must use the Solaris version of the Forms Generate component to generate .FMX files.

2. Deploy the .FMX file on your application server.
   You can deploy .FMX files in any directory on your application server, provided you include the appropriate directory path and filename in the cartridge or HTML file that end users access to run the application.
   If you provide only the .FMX filename in your application cartridge or HTML file (with no directory path), the Forms Server Runtime Engine looks in two places for the .FMX file:
   - ORACLE_HOME\bin\ The Runtime Engine searches here first.
   - FORMS45_PATH
     If the Runtime Engine cannot find the .FMX file in ORACLE_HOME\bin\, it then searches the directory tree defined by the FORMS45_PATH environment variable for the first occurrence of the .FMX file.

Notes:

- For information on developing form definitions, refer to the Forms 4.5 Developer’s Guide.
- You can view and set ORACLE_HOME, FORMS45_PATH, and other environment variables. For more information, refer to the “Developer/2000 Configuration” chapter of the Developer/2000 Release 1.4W Installation Guide for your operating system.
2.2 Configuring Forms for the Web

2.2.2 Step 2: Starting and stopping the Forms Server Listener

Before users can run your Web-enabled Forms applications, you must start a Forms Server Listener process on your application server. The Listener handles Web application requests from end users.

Below are instructions for:

■ Starting a Forms Server Listener process
■ Checking the status of the Forms Server Listener
■ Stopping a Forms Server Listener process

2.2.2.1 Starting a Forms Server Listener process

On Microsoft Windows NT:

1. On the NT taskbar, click Start to display the Start menu.
2. Choose Programs → Developer/2000 for NT → Forms Server Listener

   or

   Choose Run, then type <ORACLE_HOME>\bin\f45srv32 port=<port> and click OK. For example: c:\orant\bin\f45srv32 port=5555

   A Listener process starts running on the specified port.

On Unix:

1. From the Unix prompt, type f45ctl start port=<port> pool=<pool> and press Enter.

   For example: f45ctl start port=4321 pool=5

   The f45ctl command executes a script that starts a Listener process running in the background.

Notes:

■ The port and pool parameters are optional; Forms will use default values (port=9000, pool=1) if you leave them blank. The pool parameter (Unix only) specifies the number of Runform processes to start and make available for allocation to users. When a pool process is allocated to a user, another Runform process replaces it in the pool.

■ The port number on which you start the Listener process must match the port number you specify in an application’s cartridge settings, cartridge HTML file, or static (non-cartridge) HTML file. For more information, refer to Section 2.2.3, “Step 3: Making applications available to end users.”
2.2.2.2 Checking the status of the Forms Server Listener

On Microsoft Windows NT:

1. With your mouse, point to a blank area of the NT taskbar, and display the popup menu.
   Most users click the right mouse button to display a popup menu.
2. Choose Task Manager.
3. In the Task Manager, click the Processes tab.
   If a Listener process is running, the Task Manager will display a process called f45srv32.exe, and multiple occurrences of the f45web32.exe process (one for every active connection).

On Unix:

- At the Unix prompt, type `ps -ef | grep f45` and press Enter.
  A list of process IDs will appear on the screen. If the Listener is running, the list will include a process called f45srvm (including its port and pool settings), and multiple occurrences of the f45runw process (one for every active connection, plus any spare connection(s) ready for the next user).

2.2.2.3 Stopping a Forms Server Listener process

On Microsoft Windows NT:

1. Check the status of the Forms Server Listener.
   If the Listener is running, the Task Manager will display a process called f45srv32.exe.
2. Select f45srv32.exe and click End Process.

On Unix:

1. Check the status of the Forms Server Listener.
   A list of process IDs will appear on the screen. Note the port number of the f45srvm process you want to stop.
2. At the Unix prompt, type `f45ctl stop port=<port>` and press Enter.
   You must specify a port number when stopping a Forms Server Listener process.
2.2 Configuring Forms for the Web

2.2.3 Step 3: Making applications available to end users

Once you have generated and deployed .FMX files for the Forms applications you want to deploy on the Web, make the applications available to end users by doing the following:

1. Creating virtual directories on your Web server
2. Choosing an implementation: cartridge vs. non-cartridge
3. Creating an application cartridge
   or
   Creating a static (non-cartridge) HTML file
4. Providing application URLs to end users
2.2.3.1 Creating virtual directories on your Web server

To establish pointers to various physical directories on your application server, create virtual directories with your Web server.

<table>
<thead>
<tr>
<th>File-System Directory</th>
<th>Flag</th>
<th>Virtual Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>C:\orant\forms45\java\</td>
<td>NR</td>
<td>/web_code/</td>
</tr>
<tr>
<td>C:\apps\web\html\</td>
<td>NR</td>
<td>/web_html/</td>
</tr>
<tr>
<td>C:\apps\web\jars\</td>
<td>NR</td>
<td>/web_jars/</td>
</tr>
</tbody>
</table>

Creating virtual directories offers the following benefits:

- **Simplicity** When you create an application cartridge or custom HTML file, or use a URL to point to the cartridge or HTML file, you can refer to a virtual directory instead of specifying a lengthy physical directory path.

- **Portability** If you move files to different physical directories in the future, you only need to modify the virtual directory settings in your Web server (instead of modifying all existing cartridge settings or HTML files).
You should create virtual directories for the following:

- **Applet codebase** Create a virtual directory (for example: `/web_code/`) to point to the physical directory where the Forms Client will search for Java class files (`ORACLE_HOME/forms45/java`).
  
  **Note:** Do not create a virtual directory called `/oracle/` for applet codebase.

- **HTML files** Create a virtual directory (for example: `/web_html/`) to point to the physical directory where you store your cartridge or static HTML files.

- **JAR files** Create a virtual directory (for example: `/web_jars/`) to point to the physical directory where the Oracle JAR files (and any custom JAR files) are stored.

Below are some sample virtual directory definitions. The physical and virtual directories are provided as examples only, and are not meant as guidelines or requirements, with one exception: the physical directory you define for your applet codebase virtual directory must be `ORACLE_HOME/forms45/java`.

Other than the physical directory for applet codebase, you can create physical and virtual directory settings according to your own configuration and needs.

<table>
<thead>
<tr>
<th>Purpose</th>
<th>Example Physical Directory</th>
<th>Example Virtual Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>applet codebase</td>
<td>c:\orant\forms45\java\</td>
<td>/web_code/</td>
</tr>
<tr>
<td>HTML files</td>
<td>c:\apps\web\html\</td>
<td>/web_html/</td>
</tr>
<tr>
<td>JAR files</td>
<td>c:\apps\web\jars\</td>
<td>/web_jars/</td>
</tr>
</tbody>
</table>

Notes:

- For information on cartridge HTML files, refer to Section 2.2.3.3.4, “Creating a base HTML file for the cartridge.”
- For information on static HTML files, refer to Section 2.2.3.4, “Creating a static (non-cartridge) HTML file.”
- For information on JAR files, refer to Section 2.2.4.2.1, “About JAR files.”
2.2.3.2 Choosing an implementation: cartridge vs. non-cartridge

When an end user first selects a URL to start a Web-enabled Forms application, an HTML file downloads to the user’s Web browser. The HTML file contains all necessary applet tags, parameters, and parameter values required to run the selected application on the Web.

The initial HTML file can be built in two ways—dynamically or statically—depending on the implementation you choose:

- **Cartridge implementation** In a cartridge implementation, the initial HTML file is built dynamically by the *Forms Cartridge Handler*. The benefits of this implementation are that you can create a generic application cartridge once, and then reuse it continually (for each of your applications). To use the cartridge with another application, you can simply modify the URL of the application.

  **Note:** To create application cartridges, you must install the Oracle Web Request Broker on your application server. Oracle WRB provides a framework for cartridges, and manages client connections to them.

  For instructions on cartridge implementation, refer to Section 2.2.3.3, “Creating an application cartridge.”

- **Non-cartridge implementation** In a non-cartridge implementation, you must create a static HTML file for each application. Each static HTML file you create contains hardcoded information specific to the individual application.

  The benefit of static HTML file delivery is that you do not need to install Oracle Web Request Broker on your application server.

  For instructions on non-cartridge implementation, refer to Section 2.2.3.4, “Creating a static (non-cartridge) HTML file.”
2.2 Configuring Forms for the Web

2.2.3.3 Creating an application cartridge
Create an application cartridge for a Forms application by doing the following:

- Setting cartridge parameters and values
- Registering the cartridge with the Forms Cartridge Handler
- Creating a virtual directory for the cartridge
- Creating a base HTML file for the cartridge

2.2.3.3.1 Setting cartridge parameters and values
The cartridge parameters and values that you set specify runtime settings of your Web application.

To set cartridge parameters and values for an application:

1. Open the Oracle Web Request Broker to access the WRB Administration page.
2. Under the Modify Web Request Broker Configuration heading, select the appropriate Web server listener and click Modify.
   The WRB Cartridge Administration page appears.
3. Under the Cartridges heading, enter a name for the new cartridge.
   For example: web_cart
4. Click Create Cartridge.
   The WRB Cartridge Configuration page appears.
5. Under the Cartridge Parameters heading, enter the following parameters and parameter values:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseHTML&lt;sup&gt;1&lt;/sup&gt;</td>
<td>Enter the physical directory path and filename for the base cartridge HTML file accessed by the Forms Cartridge Handler at runtime. For information on creating a cartridge HTML file, refer to Section 2.2.3.3.4, “Creating a base HTML file for the cartridge.”</td>
</tr>
<tr>
<td>HTMLdelimiter&lt;sup&gt;2&lt;/sup&gt;</td>
<td>Enter the one-character delimiter used to denote parameter values in the application’s cartridge HTML file. The delimiter can be any character. The default is % (percent sign).</td>
</tr>
<tr>
<td>Code</td>
<td>Enter the following (case-sensitive): \texttt{oracle.forms.uiClient.v1_4.engine.Main}</td>
</tr>
<tr>
<td>Codebase</td>
<td>Enter the name of the virtual directory you defined to point to the physical directory \texttt{ORACLE_HOME/forms45/java}.</td>
</tr>
<tr>
<td>Archive</td>
<td>Provide the directory path and filename of any JAR file(s) you want downloaded to end users’ Web browsers at application startup. For more information, refer to Section 2.2.4.2.4, “Referencing JAR files.”</td>
</tr>
<tr>
<td>serverPort</td>
<td>Enter the number of the port on which the Forms Server Listener process was started. In most cases, the port is 9000 (the default). For more information, refer to Section 2.2.2, “Step 2: Starting and stopping the Forms Server Listener.”</td>
</tr>
<tr>
<td>&lt;forms_param&gt;</td>
<td>Enter valid Forms command-line parameters (such as module) and corresponding values.</td>
</tr>
<tr>
<td>&lt;user_param&gt;</td>
<td>Enter valid user-defined parameters and corresponding values.</td>
</tr>
<tr>
<td>serverApp</td>
<td>Enter the name of the application class you wish to apply to the application. Use application classes for creating application-specific font mapping and icon path settings. For more information, refer to the Appendix: Section B, “Working with Application Classes.”</td>
</tr>
</tbody>
</table>

<sup>1</sup> Parameters marked with asterisks (*) are required if using a cartridge HTML file.

For examples of parameters and values, see Figure 2–2.
2.2 Configuring Forms for the Web

**Figure 2–2** Setting application cartridge parameters and values

6. Click **Modify Cartridge Configuration**.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseHTML</td>
<td>c:\apps\web\html\web_cart.html</td>
</tr>
<tr>
<td>HTMLdelimiter</td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>oracle.forms.uiClient_4.engine.Main</td>
</tr>
<tr>
<td>Codebase</td>
<td>\web\code\</td>
</tr>
<tr>
<td>Archive</td>
<td>\web\jars\h5web.jar, \web\jars\acme.jar</td>
</tr>
<tr>
<td>serverPort</td>
<td>8000</td>
</tr>
<tr>
<td>module</td>
<td>c:\apps\web\order.lmx</td>
</tr>
<tr>
<td>userid</td>
<td>clerk@ord_db</td>
</tr>
<tr>
<td>serverApp</td>
<td>ord_class</td>
</tr>
</tbody>
</table>
2.2.3.3.2 Registering the cartridge with the Forms Cartridge Handler

At runtime, the Forms Cartridge Handler dynamically constructs an HTML file (to pass back to the end user’s Web browser) by merging information from three sources:

- the application’s cartridge parameter settings
- information from the application cartridge HTML file (if any)
- parameters and values from the application’s URL

Note: If the same parameter is set both in the application’s cartridge settings and in the application’s URL, the URL settings will override settings from the cartridge.

Figure 2–3  Forms Cartridge Handler process flow
To register a cartridge with the Forms Cartridge Handler:

1. In Oracle Web Request Broker, access the WRB Cartridge Administration page.
2. In the Applications and Objects section, enter values for the following fields:

<table>
<thead>
<tr>
<th>In this field...</th>
<th>Enter this...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>The cartridge name.</td>
</tr>
<tr>
<td></td>
<td>For example: web_cart</td>
</tr>
<tr>
<td>Object Path</td>
<td>The directory path and filename of the Forms Cartridge Handler.</td>
</tr>
<tr>
<td></td>
<td>On Microsoft Windows NT: ORACLE_HOME\bin\f45webc.dll</td>
</tr>
<tr>
<td></td>
<td>On Unix: ORACLE_HOME/lib/f45webc.so</td>
</tr>
<tr>
<td>Entry Point</td>
<td>form_entry</td>
</tr>
<tr>
<td>Min / Max</td>
<td>The minimum and maximum number of users, respectively, that can simultaneously connect to the cartridge.</td>
</tr>
</tbody>
</table>

**Applications and Objects**

<table>
<thead>
<tr>
<th>App.</th>
<th>Object Path</th>
<th>Entry Point</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>web_cart</td>
<td>C:\orant\bin\f45webc.dll</td>
<td>form_entry</td>
<td>0</td>
<td>10</td>
</tr>
</tbody>
</table>

*Figure 2–4  Registering a cartridge with the Forms Cartridge Handler*
2.2.3.3 Creating a virtual directory for the cartridge

At runtime, end users pass URLs to your Web server. If a request contains the virtual directory associated with your cartridge, the Web server automatically transfers the request to the WRB. The WRB, in turn, starts the cartridge associated with the virtual directory.

To create a virtual directory for the cartridge:

1. In Oracle Web Request Broker, access the WRB Cartridge Administration page.
2. Under the Applications and Directories section, enter values for the following fields:

<table>
<thead>
<tr>
<th>In this field...</th>
<th>Enter this...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual Path</td>
<td>A virtual directory which—when included in an application’s URL—will automatically start its associated cartridge. Do not include a trailing slash. For example: <code>/web_cart</code></td>
</tr>
<tr>
<td>Application</td>
<td>The name of the cartridge the WRB will execute whenever a URL containing the associated virtual directory is received by the WRB. For example: <code>web_cart</code></td>
</tr>
</tbody>
</table>
| Physical Path    | The directory path for the Forms Cartridge Handler:  
|                  | - On Microsoft Windows NT: `ORACLE_HOME/bin`  
|                  | - On Unix: `ORACLE_HOME/lib` |

![Application and Directories](image)

*Figure 2–5 Creating a virtual directory for a cartridge*
2.2.3.3.4 Creating a base HTML file for the cartridge

To create a base HTML file that the Forms Cartridge Handler will merge with information from your application’s cartridge settings and URL, create an application cartridge HTML file.

This is an optional step, and can be used for the following purposes:

- to include text or images on the HTML page that users see when they first start up a Web application
- to specify Forms command-line parameters (such as Module and Userid)
- to specify user-defined parameters

The easiest way to create an application cartridge HTML file is to modify the template provided by Oracle.

To create a base HTML file for the cartridge:

1. Make a copy of `cartridg.html` (the Oracle template).
2. Rename the copy.
   
   For example: `web_cart.html`
3. Open the file in a text editor and modify the text and applet tags as needed.
   
   For information on the Oracle cartridge HTML file template (including a snapshot, instructions on modifying the template, and examples of customized files), refer to Section A.2, “Working with the Cartridge HTML File Template.”
4. Place the new file in the physical directory that corresponds to the virtual directory you created for HTML files.

Note: For information on creating a virtual directory to point to the physical directory where your HTML files are located, refer to Section 2.2.3.1, “Creating virtual directories on your Web server.”
2.2.3.4 Creating a static (non-cartridge) HTML file

To deploy non-cartridge Web applications, you must create a static HTML file for each application you deploy on the Web.

The easiest way to do this is to modify the static (non-cartridge) HTML file template provided by Oracle.

To create a static HTML file for an application:

1. Make a copy of `static.html` (the Oracle template).

2. Rename the copy.
   - For example, if your application is Order Info: `order.html`

3. Open the file in a text editor and modify the text and applet tags as needed.
   - For information on the Oracle static (non-cartridge) HTML file template (including a snapshot, instructions on modifying the template, and examples of customized files), refer to Section A.3, “Working with the Static (Non-Cartridge) HTML File Template.”

4. Place the new file in the physical directory that corresponds to the virtual directory you created for HTML files.

Note: For information on creating a virtual directory to point to the physical directory where your HTML files are located, refer to Section 2.2.3.1, “Creating virtual directories on your Web server.”
2.2 Configuring Forms for the Web

2.2.3.5 Providing application URLs to end users

Once you have created an application cartridge or static HTML file for an application, and have deployed the corresponding .FMX file(s), you must give end users access to the application. To do this, provide end users with the URL for the application cartridge or static HTML page. The URL you create will differ depending on whether you have implemented your application as a cartridge or non-cartridge application.

To run the application, end users contact the URL with their Java-enabled Web browser.

Tip: Instead of providing the URL of every application on your roster to end users, consider creating a Web application menu to serve as the central access point to all of your Web applications. You then provide just one URL to end users—the URL of the menu. From the menu, users select from various labelled links to access the application URLs. For more information, refer to Section 3.1.2, “Creating a Web application menu.”

2.2.3.5.1 About cartridge URLs

For a cartridge application, you must provide a URL that contains the virtual directory associated with the application cartridge. You also may be required to include parameter values in the URL (if the values are not provided in the application’s cartridge settings).

For example, to make your cartridge-based Inventory Check application available to end users, you might provide the following URL:

http://www.acme.com/web_cart?Module=inv_chk&Userid=clerk@stock&Deptid=90

The URL above consists of the following components:

- Transmission Protocol: http
- Domain: www.acme.com
- Web server listener port: 80 (default, implicit)
- Virtual directory associated with the web_cart cartridge: /web_cart
- Two Forms command-line parameters: Module (value = inv) and Userid (value = clerk@stock)
- A user-defined parameter for department ID: Deptid (value = 90)
Working with Template HTML Files

This appendix covers these topics:

- About Template HTML Files
- Working with the Cartridge HTML File Template
- Working with the Static (Non-Cartridge) HTML File Template

A.1 About Template HTML Files

Developer/2000 for the Web includes two “template” HTML files for your use in deploying Forms applications on the Web:

- cartridg.html\(^1\) (cartridge HTML file template)
- static.html\(^2\) (static, non-cartridge HTML file template)

Any time you wish to base an HTML file on a template, make a copy of the appropriate template, rename the copy, then customize the new file to fit your specific application(s).

**Note:** Due to an HTML syntax requirement, you must locate the tags that define the Forms Client applet between the `<BODY>` and `</BODY>` tags of an HTML file.

\(^1\) ORACLE_HOME\forms45\demos\webdemos\cartridg.html
\(^2\) ORACLE_HOME\forms45\demos\webdemos\static.html
A.2 Working with the Cartridge HTML File Template

When you deploy a Forms application on the Web in a cartridge implementation, you can create a cartridge HTML file for the application. The easiest way to do this is to modify cartridg.html, the cartridge HTML file template provided by Oracle.

At runtime, the Forms Cartridge Handler dynamically creates a new HTML file by merging information from the following sources:

- the application’s cartridge HTML file
- the application’s cartridge settings
- the application’s URL

The new (dynamically-created) HTML file then downloads to the end user’s Web browser.

For more information, refer to the following sections:

- Section 2.2.3.3, “Creating an application cartridge.”
- Section 2.2.3.5, “Providing application URLs to end users.”
A.2 Working with the Cartridge HTML File Template

A.2.1 Snapshot: the cartridge HTML file template

<HTML>

<!-- FILE: cartridg.html -->
<!-- Oracle Cartridge HTML File Template -->
<!-- Rename, and modify tags and parameter values as needed -->

<HEAD><TITLE>Developer/2000 for the Web</TITLE></HEAD>

<BODY><BR>Please wait while the Forms Client class files download and run.
    <BR>This will take a second or two...<P>

<!-- applet definition (start) -->
<APPLET CODEBASE="/codebase_vdir/
    CODE="oracle.forms.uiClient.v1_4.engine.Main"
    ARCHIVE="/jars_vdir/f45web.jar"
    HEIGHT=20
    WIDTH=20>

<PARAM NAME="serverPort"
    VALUE="9000">

<PARAM NAME="serverArgs"
    VALUE="forms_param=%forms_param%"
    USER_PARAM=%user_param%">

<PARAM NAME="serverApp"
    VALUE="default">

</APPLET>
<!-- applet definition (end) -->
</BODY>
</HTML>
A.2.2 Modifying the cartridge HTML file template

Modify the Oracle cartridge HTML file template to fit your application as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>codebase</td>
<td>Replace /codebase_vdir/ with the virtual directory you defined to point to the physical directory ORACLE_HOME/forms45/java/. For example: /web_code/</td>
</tr>
<tr>
<td>archive</td>
<td>Replace or add to /jars_vdir/f45web.jar to provide the virtual directory path and filename of any JAR file(s) you want to be downloaded to end users' Web browsers at application startup. For more information, refer to Section 2.2.4.2.1, &quot;About JAR files.&quot;</td>
</tr>
<tr>
<td>serverPort</td>
<td>Replace 9000 with the number of the port on which the Forms Server Listener process was started. In most cases, the port number will remain 9000 (the default). For more information, refer to Section 2.2.2, &quot;Step 2: Starting and stopping the Forms Server Listener.&quot;</td>
</tr>
<tr>
<td>serverArgs</td>
<td>Replace forms_param with any valid Forms command-line parameter. Replace user_param with any valid user-defined parameter. Notes:</td>
</tr>
<tr>
<td></td>
<td>- You can provide multiple Forms command-line and user-defined parameters.</td>
</tr>
<tr>
<td></td>
<td>- You must provide a physical directory path for the .FMX file. The .FMX suffix is optional.</td>
</tr>
<tr>
<td>serverApp</td>
<td>Replace default with the name of your application class (if any). Use application classes for creating application-specific font mapping and icon path settings. For more information, refer to Section B, “Working with Application Classes.”</td>
</tr>
</tbody>
</table>

You can provide a delimited value for any cartridge HTML file parameter. For example, you could have the following line in your cartridge HTML file:

```
ARCHIVE="%Archive%"
```

You then must assign a value to %Archive% (either in the application’s cartridge settings, or in the application’s URL). Note that all variable parameters in your cartridge HTML file must receive values in this way at runtime. If a parameter does not receive a value, the Forms Cartridge Handler cannot build an HTML file to pass back to the user’s Web browser, resulting in an error. For information on checking for Forms Cartridge Handler errors, refer to Section 2.2.5, “Troubleshooting your Forms configuration.”
A.2.3 Examples of customized cartridge HTML files

The following examples display variations of the cartridge HTML file template.

A.2.3.1 Example 1

ACME has created its own cartridge HTML file template. It includes the following:
- a title for the Web browser window
- a virtual directory for the Forms Java codebase: /web_code/
- a Forms command-line parameter: Module

```
<!-- FILE: tempcart.html -->
<!-- ACME cartridge HTML file template -->
<!-- rename and modify as needed -->

<HEAD><TITLE>ACME Web Applications</TITLE></HEAD>

<BODY><CENTER>
  <BR>Welcome to ACME Web Applications!
  <BR>Please wait for the application to load...
  <BR></CENTER><P>

<APPLET CODEBASE="/web_code/"
    CODE="oracle.forms.uiClient.vl_4.engine.Main"
    HEIGHT=20
    WIDTH=20>
  <PARAM NAME="serverPort"
        VALUE="9000">

  <PARAM NAME="serverArgs"
        VALUE="Module=%Module%">

</APPLET>

</BODY>
</HTML>
```
A.2.3.2 Example 2
General Traders Inc.’s cartridge HTML file includes the following:

- a virtual directory for images: /imag/
- a background image and main image: gti_wall.jpg and gti_logo.jpg
- a virtual directory for JAR files: /web_jars/
- two JAR files: f45web.jar and gti.jar
- a virtual directory for the Forms Java codebase: /web_code/
- two Forms command-line parameters: Module and Userid
- one user-defined parameter: Dept_id
- an application class setting: gti_ac

```html
<html>
<!-- FILE: gti_cart.html -->
<!-- General Traders cartridge HTML file -->

<body background="/imag/gti_wall.jpg">
<center><img src="/imag/gti_logo.gif" height=170 width=120 alt="gti_logo.jpg">
</center>

<applet codebase="/web_code/"
    code="oracle.forms.uiClient.v1_4.engine.Main"
    archive="/web_jars/f45web.jar ,
    /web_jars/gti.jar"
    height=20 width=20>
    <param name="serverPort" value="7777">
    <param name="serverArgs" value="Module=%Module%
                  Userid=%Userid%
                  Dept_id=%Dept_id%">
    <param name="serverApp" value="gti_ac">
</applet>
</body>
</html>
```
A.3 Working with the Static (Non-Cartridge) HTML File Template

When you deploy a Forms application on the Web in a non-cartridge implementation, you must create a static HTML file for the application. The easiest way to do this is to modify static.html, the static HTML file template provided by Oracle:

At runtime, the static HTML file downloads to the end user’s Web browser.

For more information, refer to the following sections:

- Section 2.2.3.4, “Creating a static (non-cartridge) HTML file.”
- Section 2.2.3.5, “Providing application URLs to end users.”
A.3.1 Snapshot: the static (non-cartridge) HTML file template

```html
<HTML>
<!-- FILE: static.html -->
<!-- Oracle Static (Non-Cartridge) HTML File Template -->
<!-- Rename, and modify tags and parameter values as needed -->

<HEAD><TITLE>Developer/2000 for the Web</TITLE></HEAD>

<BODY><BR>Please wait while the Forms Client class files download and run.
  <BR>This will take a second or two...<P>

<!-- applet definition (start) -->
<APPLET CODEBASE="/codebase_vdir/"
        CODE="oracle.forms.uiClient.v1_4.engine.Main"
        ARCHIVE="/jars_vdir/f45web.jar"
        HEIGHT=20
        WIDTH=20>

<PARAM NAME="serverPort"
       VALUE="9000"/>

<PARAM NAME="serverArgs"
       VALUE="module=fmx_name
                 userid=user/password@datasource">

<PARAM NAME="serverApp"
       VALUE="default">

</APPLET>
<!-- applet definition (end) -->

</BODY>
</HTML>
```
## A.3.2 Modifying the static (non-cartridge) HTML file template

Modify the Oracle static HTML file template to fit your application as follows:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Parameter Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>codebase</td>
<td>Replace <code>/codebase_vdir/</code> with the virtual directory you defined to point to the physical directory <code>ORACLE_HOME/forms45/java</code>. For example: <code>/web_code/</code></td>
</tr>
<tr>
<td>archive</td>
<td>Replace or add to <code>/jars_vdir/f45web.jar</code> to provide the virtual directory path and filename of any JAR file(s) you want to be downloaded to end users’ Web browsers at application startup. For more information, refer to Section 2.2.4.2.1, “About JAR files.”</td>
</tr>
<tr>
<td>serverPort</td>
<td>Replace 9000 with the number of the port on which the Forms Server Listener process was started. In most cases, the port number will remain 9000 (the default). For more information, refer to Section 2.2.2, “Step 2: Starting and stopping the Forms Server Listener.”</td>
</tr>
</tbody>
</table>
| serverArgs  | Replace `fmx_name` with the name of the .FMX file you want to run on the Web. Replace `userid/password@datasource` with any valid connect string. You also can include other Forms command-line parameters, and user-defined parameters, just as if starting Forms from the command line. **Notes:**
  - You must provide a **physical** directory path for the .FMX file.
  - The .FMX suffix is optional. |
| serverApp   | Replace `default` with the name of your application class (if any). Use application classes for creating application-specific font mapping and icon path settings. For more information, refer to Section B, “Working with Application Classes.” |
A.3.3 Examples of customized static (non-cartridge) HTML files

The following examples provide variations of the static HTML file template.

A.3.3.1 Example 1
ACME’s static HTML file (for their ORDERS application) includes the following:

- a title for the Web browser window
- a brief welcoming message to users
- a virtual directory for the Forms Java codebase: /web_code/
- a virtual directory for JAR files: /web_jars/
- a JAR file: f45all.jar
- two Forms command-line parameters: module and userid

```html
<html>
<!-- FILE: order.html -->

<head><title>ACME: Web Applications</title></head>
<body>
  <center>
    <p>Welcome to ACME Web Applications!</p>
  </center>

  <!-- applet definition (start) -->
  <applet codebase="/web_code/"
    code="oracle.forms.uiClient.v1_4.engine.Main"
    archive="/web_jars/f45all.jar"
    height=20 width=20>
  <param name="serverPort" value="9000">
  <param name="serverArgs"
    value="module=c:\orant\bin\orders
      userid=clerk@inventory">
  </applet>

  <!-- applet definition (end) -->
</body></html>
```
A.3.3.2 Example 2

General Traders Inc. has their own static HTML file template that includes the following:

- a virtual directory for image: /imag/
- a background image and main image: gti_wall.jpg and gti_logo.jpg
- Forms command-line parameters that GTI changes for each application: module and userid

```html
<!-- FILE: gti_stat.html -->
<!-- GTI static HTML template -->
<!-- Rename and modify as needed -->

<BODY BACKGROUND="/imag/gti_wall.jpg">
<CENTER><IMG SRC="/imag/gti_logo.gif"
HEIGHT=170 WIDTH=120 ALT="gti_logo.jpg">
</CENTER>

<APPLET CODEBASE="/web_code/
CODE="oracle.forms.uiClient.v1_4.engine.Main"
ARCHIVE="/web_jars/gti.jar"
HEIGHT=20 WIDTH=20>
PARAM NAME="serverPort" VALUE="7777">
<PARAM NAME="serverArgs"
VALUE="module=\orant\bin\fmx_name
userid=clerk\@database">

</APPLET>
</BODY>
</HTML>
```
A.3 Working with the Static (Non-Cartridge) HTML File Template
This appendix covers these topics:

- About Application Classes
- Referencing an Application Class
- Setting Iconic Button Properties for an Application Class
- Creating Font Mappings for an Application Class

### B.1 About Application Classes

Application classes enable you to specify the following custom settings for your Web-enabled Forms applications:

- the directory path and image format (GIF or JPEG) of image files for iconic buttons
- font mappings (application fonts to Java fonts)

To create settings for an application class, you must add the appropriate settings to the `Registry.dat` file.

Using application classes is optional, since Forms supplies default settings for font mappings and icon path. Use application classes if you wish to create a number of different settings that you can alternate between by switching a single reference in your application’s cartridge settings, cartridge HTML file, or static HTML file.

At runtime, Forms first looks for settings that correspond to the specified application class (if any). If no application class settings exist, Forms uses default settings.

---

1 `ORACLE_HOME/forms45/java/oracle/forms/uiClient/v1_4/util/Registry.dat`
B.2 Referencing an Application Class

To reference an application class for your application:

1. Include the serverApp parameter (and value) in your application’s cartridge settings, cartridge HTML file, or static HTML file.

   For example:

   ```html
   ...  
   <applet codebase="/web_code/" 
   code="oracle.forms.uiClient.v1_4.engine.Main" 
   height="20" width="20"> 
   <param name="serverPort" value="9000"> 
   <param name="serverArgs" 
   value="Module=inventory"> 
   <param name="serverApp" 
   value="inv_form"> 
   </applet> 
   ... 
   ```

2. Add font mapping and icon path settings (in the Registry.dat file) specific to the application.

   For more information refer to:
   - Section B.3, “Setting Iconic Button Properties for an Application Class.”
   - Section B.4, “Creating Font Mappings for an Application Class.”
B.3 Setting Iconic Button Properties for an Application Class

To specify directory path and image format for iconic button image files:

1. Open the `Registry.dat` file in a text editor.
2. Add the following lines to `Registry.dat`:
   ```
   <app_class>.icons.iconpath=<path>
   <app_class>.icons.iconextension=<format>
   ```
   Replace `<app_class>` with a valid application class referenced by your Web application.
   Replace `<path>` with one of the following:
   - a fully-qualified URL that points to any directory on any server. For example:
     ```
     inv_form.icons.iconpath=http://abc_dev.com/html/icons/
     ```
   - a virtual directory on the same server from which the HTML page downloaded. For example:
     ```
     inv_form.icons.iconpath=/icons/
     ```
   - a directory relative to the directory from which the HTML page was downloaded (non-cartridge implementations only). For example:
     ```
     inv_form.icons.iconpath=icons/
     ```

   Replace `<format>` with `gif` (for GIF files) or `jpg` (for JPEG files). For example:
   ```
   inv_form.icons.iconextension=gif
   ```

Default Settings  If you do not specify a directory path or image format for an application class, Forms uses the default settings in `Registry.dat`:

```
default.icons.iconpath=
default.icons.iconextension=gif
```

By default, Forms will search for iconic button image files in the directory from which the application’s HTML file was downloaded, and will append “.gif” to the filename provided in your form.

You can modify the default settings in the `Registry.dat` file.
B.4 Creating Font Mappings for an Application Class

To create font-mapping settings for an application class:

1. Open the Registry.dat file in a text editor.
2. Add lines to the file as follows:

   <class>.fontMap.<param>=<value>

   Replace <class> with a valid application class, replace <param> with the name of the font parameter, and replace <value> with an appropriate value.

To map fonts from your application to their Java equivalents, add lines similar to:

   inv_form.fontMap.appFontnames=Courier,Terminal,Arial
   inv_form.fontMap.javaFontnames=MonoSpaced,Dialog,SansSerif

To map unrecognized application fonts to a Java font, add lines similar to:

   inv_form.fontMap.defaultFontname=Dialog
   inv_form.fontMap.defaultSize=10
   inv_form.fontMap.defaultStyle=BOLD

Notes:

- You can modify the default Forms settings in the Registry.dat file:

    default.fontMap.<param>=<value>

- The mapping of Java fonts to platform-specific fonts is automatic, according to default Java settings; you cannot alter the mappings. For more information, refer to Section 3.2.4, “Selecting fonts.”

- For more information on setting font preferences, refer to the commented sections of the Registry.dat file.