Generating Web Simulations

Generating Web Simulations

e-SIM Ltd.
POB 45002
Jerusalem
91450
Israel

Tel: 972-2-5870770
Fax: 972-2-5870773

Written and produced by e-SIM Ltd.
Printed in Israel.

MAN-GenWeb-7.2

Contents

ABOUT THE GENERATING WEB SIMULATIONS MANUAL

This manual is both a methodology guide and reference resource. As a methodology guide, it presents instructions and guidelines for developing RapidPLUS Web Studio applications that will be optimized for Java code generation. As a reference guide, it delineates code generation processes and output so that you will have a clear understanding of what is involved in producing simulation applets.

This manual is for the team using RapidPLUS Web Studio to develop simulations for viewing and manipulating in a Web browser. Simulations are developed in the Rapid environment, generated as Java code, then integrated into a Web site. The skills required by the design/implementation team vary, depending on the amount of customization you require:

- Rapid application development—for implementing applications in RapidPLUS Web Studio.
- Web design—for manipulating Web graphics and the default HTML, XML, and JavaScript files.
- Graphic design—for developing the graphics that will appear in simulations and Web browsers.

The size and composition of the team will vary from organization to organization and from project to project; however, in all cases the team members need to share a common conceptual model of how a Rapid application for Java code generation evolves from an idea to a functional, interactive simulation that can be viewed in a Web browser by people with different computer skill levels.

This manual comprises the following chapters:

- Chapter 1: “Introducing RapidPLUS Web Studio” introduces the concepts, environment, and components for developing, generating, and distributing Web simulations.
- Chapter 2: “Application Design Guidelines” presents the issues to consider when building a Rapid application that will be generated into a Java-code simulation applet.
- Chapter 3: “Generating Code and the Simulation” presents instructions for generating Java code, a descriptions of the code generation process, and guidelines for optimizing the simulation applet’s performance.

- Chapter 4: "Finalizing Simulations for Distribution" presents instructions and suggestions for running and testing simulation applets, and for distributing and publishing simulations.
- Chapter 5: "Developing the Scenario Player Applet" presents instructions and guidelines for building a Rapid application that will be generated as a Scenario Player applet.
- Appendix A: "Code Generation Output" describes the generated folders and files.
- Appendix B: "Errors, Warnings and Messages" describes the errors, warnings, and informational messages that may appear during code generation.
- Appendix C: "JavaScript User Interface for Scenarios" describes the Scenario Player applet functions that are accessible to JavaScript and utility libraries for use with JavaScript functions.

Document Conventions

This manual uses the following conventions:

Menu Conventions

- "Choose File|Save" means to select the Save command from the File menu.

Typographic Conventions

- HTML, Java, and JavaScript code appear in monospaced characters:
  `rapid_applet, FRAMESET`
- File names and extensions appear in italic characters:
  make.bat file, *.html* files
- Names of Rapid properties, functions, and events appear in italic characters:
  cursorExited event, *changeBy:* function
- Complete phrases of Rapid logic appear in bold, sans serif characters:
  Lamp1 enable, Integer1 changeBy: 5

- An arrow icon in the margin indicates a recommendation for Rapid application development.

CHAPTER 1

Introducing RapidPLUS Web Studio

RapidPLUS Web Studio is a comprehensive software package for building prototype applications of consumer electronics. These applications can be generated as Java™ code to produce functional, interactive simulations that can be viewed in a Web browser.

The Java simulations can be distributed via Web, intranet, kiosk, and CD-ROM. They can be used on their own, or as part of multimedia presentations that target key product features and functions.

The entire RapidPLUS complement of tools is available for you—the Rapid simulation developer—to create and test simulations as virtual product prototypes. As a prototype becomes an actual product, you can refine your Rapid application and generate a Java simulation. With Java technology, your simulations are portable and can be easily accessed by others, making them ideal solutions for training, marketing, and customer support.

This overview presents:

- An introduction to the Java environment.
- The stages involved in creating a Java simulation and a multimedia presentation.
- Ways to use simulation applets.
- Choices for generating simulations and distribution packages.
- Considerations for the Java runtime environment.

THE JAVA ENVIRONMENT

The Java programming language gives you the ability to:

- Build simulations on one platform and run them on multiple platforms.
- Create simulations for the World Wide Web.

With RapidPLUS Web Studio, you have the tools to generate applets from Rapid applications. An **applet** is a Java language program that can be viewed and run through most Java-enabled Web browsers. Applets are relatively small in size and can stream data, making them ideal for viewing via the Web.

You don't need to know Java programming to produce applets in Rapid—the Rapid code generation process translates Rapid code to Java code, compiles applets, and creates distribution packages with all the files you need to put simulations on the Web.

RAPID–GENERATED APPLETS

Two types of applets are generated in Rapid:

- Simulation applets, which are used to simulate product appearance and functionality.
- Scenario Player applets, which are combined with simulation applets to create multimedia presentations about the product.

Applets for Product Simulations

A **simulation applet**, a Java language program generated from a Rapid application, is a virtual presentation of a product that is viewed in a Web browser. Users who view and run simulation applets (referred to as "simulation users") are able to "experience" the product, either through manipulating the product themselves (in free-play) or by viewing the product as part of a multimedia presentation (prerecorded sequences that illustrate specific product features).

A simulation applet can be manipulated, almost like the real product; the interactive pushbuttons, dials, switches, and displays appear and respond the same as on the product. Simulation users can see how the product operates, gaining more than just a "feel" for what to expect from the product.

Besides the product, simulation applets typically include icons to change views or icons that represent additional devices. These additional devices may be necessary to fully simulate your product's capabilities. For example, a telephone needs to simulate responses to incoming and outgoing calls, an MP3 player needs a computer from which to download songs, and an audio/video receiver needs input and output devices.

The extent of a simulation applet's functionality mostly depends on the size of its files (an important concern for the Web) and/or your own time constraints.

Simulation applets can be used in multimedia presentations by combining them with scenarios. Scenarios are described in the following section.

Applets for Multimedia Presentations

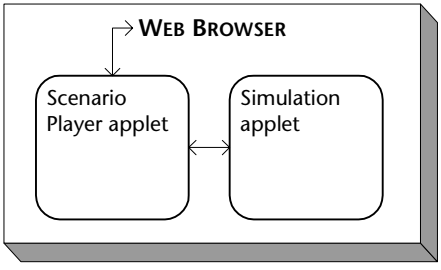
Companies can showcase their products by pairing simulation applets with scenarios. A **scenario** (also called a guided tour) is a multimedia demonstration that manipulates a simulation applet to:

- Showcase a product's features.
- Teach specific tasks.

Scenarios are created for a variety of uses, such as marketing, customer support, and training. Scenarios can present general features of a product, or they can present procedures for using the product.

The scenario combines prerecorded actions on the simulation applet (such as button presses and changing views) with text and sound. Visual aids that point out areas of interest on the simulation applet can also be added.

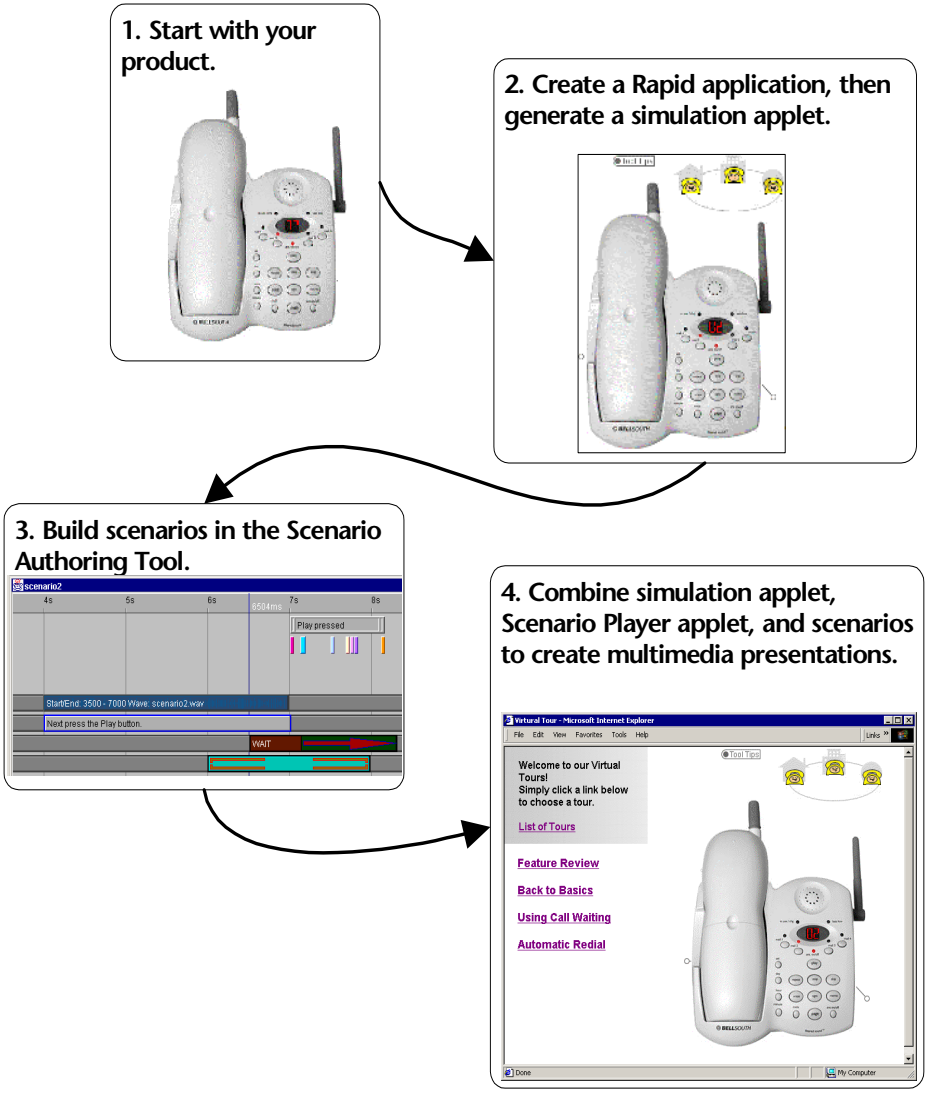
In the Web browser, scenarios run in the **Scenario Player applet**, a Java-language program that is generated from a specialized Rapid application. The Scenario Player applet communicates with the Web browser and the simulation applet (as illustrated to the right) via Java and JavaScript functions.



Scenarios are created in a separate application program called the **Scenario Authoring Tool (SAT).** For more information about scenarios, refer to the *Using the Scenario Authoring Tool* manual.

Workflow for Developing Multimedia Presentations

The following illustration presents the main phases in combining simulation applets with scenarios for presentation on a Web site:

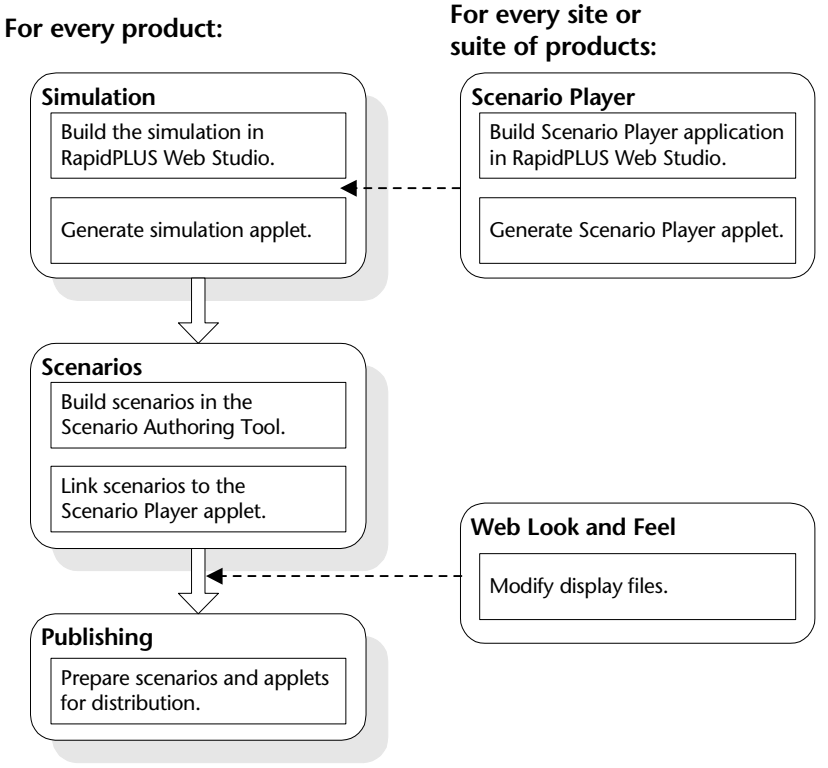


As you can infer from the illustration, the development of multimedia presentations requires coordination between you (the Rapid application

developer) and the scenario author. The simulation applet is the basis for the scenarios; its functionality determines the scope of each scenario. However, scenario requirements can direct and focus the functionality requirements for the simulation applet.

For every product, a simulation applet and a set of scenarios are developed. For every Web site or for a suite of products, a Scenario Player applet and several display files are developed.

The following illustration presents the typical workflow for developing multimedia presentations:



SIMULATIONS AND SIMULATION PACKAGES

Each simulation applet is generated to a simulation **package**, a folder with subfolders and files that comprise all the images and resources necessary for specific users or environments. This section describes the types of simulations and simulation packages that you can generate for free-play simulations or for scenario development.

For details on how to generate the required simulation type and package, see "Setting Configuration Options" on p. 44.

Choosing a Simulation Type

You can generate a Rapid application to two kinds of simulations: a stand-alone simulation and a simulation applet. Different tools are required to compile and view each kind of simulation. (For details about the required tools, refer to the Readme file, *Readme.wri,* in the Rapidx folder). Among the major differences between simulations are portability and file compression.

Stand-Alone Simulation

A stand-alone simulation runs in a stand-alone Prototyper window (i.e., a window that opens outside the Rapid environment). Stand-alone simulations are intended for the Rapid application developer to test simulation functionality quickly, without generating a simulation applet. It uses resources from the \\Rapidx\Java folder on the developer's computer.

- **Required tool for compilation:** JDK 1.3.
- **Viewer:** Stand-alone Prototyper.
- **Portability:** Runs only on a computer with RapidPLUS Web Studio installed (requires the \\Rapidx\Java folder).
- **File compression:** None.

❖ *NOTE: The stand-alone Prototyper contains menu options for recording user actions (such as button presses or cursor events) in an XML document called <MyApp>_UDC_<MyApp>_recording.txt. This document is used for building simple scenarios without the Scenario Authoring Tool.*

Simulation Applet

A simulation applet runs in the Java applet viewer and in Web browsers. The initial download of *.jar* or *.cab* files—the main body of the simulation—uses standard Web browser progress indicators.

- **Required tools for compilation:** JDK 1.3 and DashO-Pro.
- **Viewers:** Web browsers and Java applet viewer.
- **Portability:** Can be placed on a CD-ROM, on other computers, and on a Web server.
- **File compression:** Loaded files are optimized for size; Internet Explorer uses the *.cab* format, Netscape uses the *.jar* format (about 70% larger than the *.cab* format).
- **What the simulation user sees during simulation download:** While the *.cab* or *.jar* file loads, an animation appears in the browser window. (To change the animation, see "Customizing What the User Sees During Download" on p. 68.) Following the animation, a progress bar appears in the browser window while images load.

Choosing a Simulation Package

RapidPLUS Web Studio code generation produces simulation packages that comprise folders and files for different users and environments. You can generate a simulation package for use with the Scenario Authoring Tool, or you can generate simulation-only packages.

Scenario Authoring Package

Scenario authoring packages use simulation applets. The ScenPack subfolder in the code generation output folder contains all the files necessary for creating scenarios, for Web design modifications, and for distributing the scenario and simulation. The ScenPack folder includes:

- Simulation applet files.
- Scenario Player applet files.
- Utility file for linking scenarios to the Scenario Player and simulation applets.
- The complete site package folder, containing everything necessary for uploading to a Web server.

The ScenPack folder can be sent to the scenario author who creates scenarios and links them to the simulation. A Web designer can modify the Web page that holds the applets, and can upload the site package folder to a Web server.

Simulation-Only Packages

Simulation-only packages can be either stand-alone simulations or simulation applets.

Stand-Alone Simulation

A stand-alone simulation is generated in the StandAlone subfolder of the code generation output folder. Because this type of simulation is for testing during application development, it can only be viewed on a computer with the full RapidPLUS Web Studio installation.

Simulation-Only Applet

A simulation-only applet is generated in the SitePack subfolder of the code generation output folder. The simulation applet is called by the generated *demo_sim.html* file in the SitePack folder. The Web designer can modify and incorporate the simulation into a Web page.

Front End Processor CD Package

The Front End Processor (FEP) is an external object that can be added to a RapidPLUS Web Studio application; it enables users to enter Japanese text through a keyboard or keypad. Java code generation supports FEP use in browsers only when the simulation applet runs from a hard disk or a CD-ROM (not over the Internet). An FEP CD package is generated to the CdPack subfolder of the code generation output folder. It contains the files necessary to install and uninstall FEP browser support on the simulation user's machine.

PLANNING FOR THE JAVA RUNTIME ENVIRONMENT

When you plan a Rapid application for the Java runtime and Web environments, keep in mind that download time is a significant factor. After simulation users click a link to your simulation, it is important for them to see and use the simulation as soon as possible.

Below are general guidelines to consider when building your Rapid application. Chapter 2: "Application Design Guidelines" presents more specific details.

Controlling Download Time

You can control three factors that affect the download time:

- The number of files.
- The size of each file.
- The order in which files download.

The code generation process automatically optimizes files by removing unused code, combining small files into one, and dispersing files for download.

Conscientious planning while developing a Rapid application can lead to optimized simulations, especially with regard to the number of files and their size. Outside of the Rapid environment, you can adjust the sequence in which files download.

To promote simulation optimization, you should limit:

- Excess or redundant Rapid objects and logic.
- The number and size of image files.
- The number of color palettes.
- The number of user objects *(.udo)*.
- The simulation file size (recommended under 500 KB).

Planning for Behavioral Differences

You should anticipate differences between the computer's operating system and the Java virtual machine that can result in behavioral differences. These differences are explained in detail in "Anticipating Differences in Environments" on p. 32.

CHAPTER 2

Application Design Guidelines

With RapidPLUS Web Studio, you—the application developer—can generate a functional simulation applet without knowing the Java programming language. However, the Rapid application should anticipate the Java environment as much as possible.

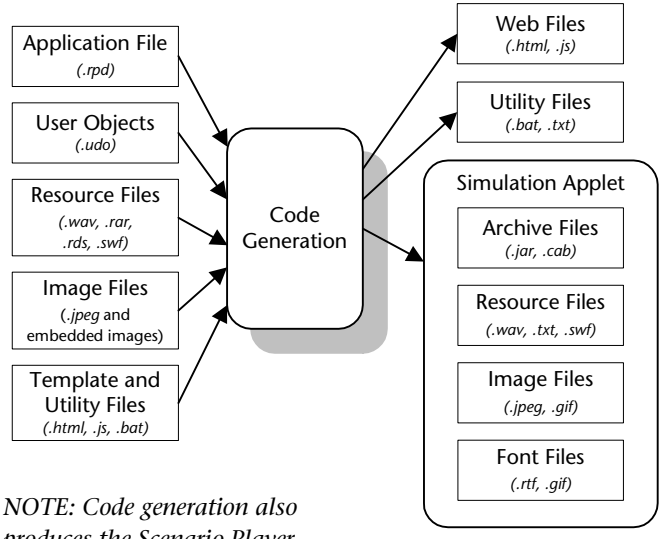
This chapter discusses design issues that you should consider when building a Rapid application for Java code generation.

This chapter presents guidelines for:

- Naming Rapid applications to avoid conflicts during code generation.
- Choosing objects for optimal simulation size and functionality.
- Referencing external files such as data store files, array files, audio files, and JavaBeans.
- Using graphics and sound files to enrich the look and feel of the simulation.
- Choosing fonts.
- Optimizing logic for smaller simulation size.
- Anticipating behavioral differences between the Rapid and Java environments.

CODE GENERATION INPUTS AND OUTPUTS

Before you design a Rapid application for generation to a simulation applet, you should be aware of the inputs and outputs involved. The illustration below presents the main file types:



❖ *NOTE: Code generation also produces the Scenario Player applet. For details about the Scenario Player applet, see p. 71.*

These inputs and outputs are described in detail throughout this manual. In addition, the code generation output is summarized in Appendix A: "Code Generation Output."

NAMING APPLICATIONS AND RESOURCES

You can avoid conflicts when compiling simulation applets by following these rules when naming your Rapid applications and resources:

- Do not give user objects or the parent application the same name as reserved Java names (e.g., int, float double, public).

- Do not give user objects or the parent application the same name as folders created by the code generation process (e.g., resources, images, SitePack).
- Do not give a user object the same name as the parent application. Otherwise, the files generated for the Rapid application *(.rpd)* will overwrite the files generated for the user object *(.udo)*.
- Do not use spaces in folder names for the Rapid application or the Java code generation output. For example, use "MyApp_java" instead of "MyApp java" for the name of the Java code generation output folder. This rule applies to the entire path of Rapid applications and simulation applets, as well as referenced folders and files.
- Use Latin-only characters to name applications and user objects.
- Use Latin-only characters to name objects in Rapid. (Certain non-Latin characters may generate an error in the Java compiler or DashO-Pro.)
- Enter folder and file names for applications, user objects, and referenced files accurately, including attention to case sensitivity. This is because the simulations may run from a case-sensitive platform (e.g., UNIX).

❖ ***CAUTION:*** *Not conforming to these rules may produce errors during code generation, compilation, or runtime.*

CHOOSING RAPID OBJECTS

Most Rapid objects are supported for Java code generation and are fully functional. When faced with a variety of similar objects, it is important to choose objects that fulfill application requirements, yet do not unnecessarily weigh down the simulation.

Generated and Nongenerated Elements

Most Rapid objects can be generated as Java code; however, some of these objects have functions and options that cannot be generated. Listed below are the objects that can be generated and the functions and options that cannot be generated. Following the lists are additional requirements and restrictions for the generated objects.

During the Java code generation process, any logic lines that include nongenerated objects or nongenerated functions are ignored and appropriate warnings are displayed in the Code Generation Status dialog box.

List of Generated Objects

This is an inclusive list of objects that can be generated for simulation applets.

- Primitive objects: all types of frame, circle, and line objects, and horizontal label objects.
- Bitmap and image objects.
- Data objects: all types.
- Constant objects: all types.
- Time objects: all types.
- Signal objects: event and sound objects.
- Display objects: font, graphic display, text display, and touch screen objects.
- Lamp objects: both types.
- Indicator objects: round and square dials, vertical and horizontal linear indicators, vertical and horizontal bar indicators, barrel and sector dial objects.
- Indicator objects: user single pointer objects of the following styles: Line, Needle, Two sided needle, Ellipse, Diamond, Arrow, Point in 1. (All other pointer styles are generated as Line.)
- Potentiometer objects: all types.
- Pushbutton objects: all types.
- Switch objects: all types.
- FEP objects.
- Wave audio objects.
- Animated objects.
- JavaBean objects.
- System objects: ASCII objects, mouse objects, SystemCursor objects, SystemDate objects, and SystemTime objects.
- Modes as objects (triggers).
- User objects *(.udo*).

Nongenerated Functions or Options

Of the objects that are generated, the following functions and options are not generated:

GENERATED OBJECTS	NONGENERATED FUNCTIONS OR OPTIONS
All active graphic objects	*name* function is not generated.
Data store, number array, string array, integer array objects	*saveToFile:* is not generated (a Java applet security limitation).
Graphic display objects	*dump:forBuffer:* (used to debug applications for C code generation) is not generated as Java code. For behavior differences, see "Graphic Display Objects" on p. 36.
Holder objects	Cannot be generated if they hold objects or object types that cannot be generated. *holdNew* and *holdCopyOf:* for user objects are not generated. ❖ *NOTE: Use the hold: function instead.*
Mouse objects	No functions are generated **except** for *setCursor:overObject:* and *setCursor:overSwitchPos:.*
Object arrays	Cannot be generated if they hold objects or object types that cannot be generated.
Pointer objects	Only scale angles from 0-360 degrees are generated.
Primitive objects	No user-defined properties are generated.
Round and square dials, vertical and horizontal linear indicators	Moving scale option is not generated.

GENERATED OBJECTS	NONGENERATED FUNCTIONS OR OPTIONS
String objects	*is formattedAs:* and *byteAt:* are not generated.
System cursor objects	The system cursor objects noEntry, startArrow, and upArrow are not generated. The default cursor shape is used instead.
User objects *(.udo)*	Interface messages (used in applications for C code generation) are not generated as Java code.
Wave audio objects	The *volume:* function can only accept one value, which affects both the left and right speaker channels. Do not set different left and right channel values.

JavaBean Object Considerations

The following is a list of considerations when using JavaBean objects in a Rapid application:

- **Browser compatibility:** For the widest range of Web browser compatibility, use JavaBean objects that are compatible with JDK 1.1.
- **Java applet security:** Do not use JavaBean objects that try to save files or do other privileged operations on the simulation user's computer (a Java applet security limitation).
- **Windowed and non-windowed objects:** If Java code is generated for an application, windowed and non-windowed JavaBean objects are generated according to their inherent Java definitions, regardless of settings in Rapid.
- **Font property:** For the JavaBean object's font property, seven functions are not generated—*boldGet, boldSet:, italicGet, italicSet:, nameGet, nameSet:, sizeGet.*

- **Image property:** For the JavaBean object's image property, two functions are not generated—*clear* and *loadFromFile:*.

  Instead of using the *loadFromFile:* function to load an image, you can assign a bitmap object to the image property.

  Example

  The following assignment decreases response time and should, therefore, be used sparingly:

  ```
  JavaBean1.image := Bitmap1
  ```

Choosing from Similar Objects

Choose objects that fulfill simulation functionality requirements, yet do not weigh down the application after code generation.

When possible, use:

- System fonts (see "Using Native and Nonnative Fonts" on p. 26) instead of nonstandard or customized fonts.
- Arrays instead of data stores.
- Local variables instead of data objects.
- Touch screen objects instead of many transparent pushbuttons.
- Icon system cursor with the *setCursor:overObject:* function of the mouse object. This becomes the hand cursor in the simulation applet.

REFERENCING FILES

A Rapid application often references files outside of the application. External files include: Rapid generated files *(.rds* and *.rar),* audio files *(.wav),* image files, and JavaBean resources.

The Java code generation process handles each of these file types somewhat differently.

Data Store Files *(.rds)* and Array Files *(.rar)*

By default, the Java code generation process looks for these files in a subfolder called "resources" (all lowercase letters). You must create this folder yourself next to the *.rpd* application. To use a subfolder other than one called "resources," see "[Resource Folders]" on p. 48.

Use a relative path in the application logic to load referenced files. For example, **Array1 loadFromFile: 'resources\MyArray.rar'** loads the file *MyArray.rar,* located in the resources folder.

During code generation, these files are converted to text format *(.txt).*

Audio Files *(.wav)*

Audio files should be placed in the same folder as *.rds* and *.rar* files.

During code generation, files are copied; they are **not** compressed, combined, or otherwise changed. For further details on working with audio files, see p. 22.

Image Files

Keep image files in a subfolder **other than** the "resources" folder.

During code generation:

- Linked *.jpg* files are copied and renamed.
- All supported bitmap formats—other than linked *.jpg* files—are converted to *.gif* files and combined when possible. For further details on working with graphic files, see p. 19.

JavaBean Resources

Keep JavaBean resources in a subfolder **other than** the "resources" folder.

Before code generation, adjust *make.config* file. For instructions, see "Specifying a JavaBean Resource Folder" on p. 49.

WORKING WITH GRAPHIC FILES

RapidPLUS Web Studio supports bitmaps of the following types: *.bmp, .dib, .png, .jpg, .ico, .msp, .pcx,* and *.psd*. When these bitmaps are generated as Java code, all of them—except for linked *.jpg* files—are converted to 256-color *.gif* files. The *.jpg* and *.gif* formats are the only graphic formats that all Web browsers support for Java technology.

Choosing Formats for Application Graphics

There are several issues to keep in mind when deciding which graphic format to use in a Rapid application:

- A *.gif* file uses a 256-color palette while a *.jpg* file uses a true-color palette.
- A *.gif* file supports transparency, a *.jpg* file does not.
- Bitmaps that share the same color palette, same transparent color, and are in the same download category (see "Image Download Categories" on p. 57), are automatically combined into a single *.gif*. The simulation download time is less if there are fewer *.gif* files.
- Simulations with fewer *.gif* files consume less system resources, which may be a significant concern for Windows 95 and 98 operating systems.

Recommendations

Use True-Color Display Settings

For high quality graphics, when possible work with the monitor display setting in true color.

Use the Same 256-Color Palettes for Bitmap/Image Objects

If your application uses 256-color graphics, limit the number of different palettes so that the graphics can be combined into as few *.gif* files as possible.

To use the same 256-color palette:

1. In your graphics editor, set the same 256-color (or less) palette for all bitmaps.
2. In the Rapid application, import the bitmaps (File|Import Bitmaps) or add bitmap/image objects in the Object Layout work area.

   ❖ *NOTE: If the monitor display setting is 256 colors, Click Yes to replace the application palette with the object palette. Only one palette of 256 colors can be used in an application when the monitor is set to 256 colors.*
3. If you paste images in the Object Editor window and the display setting is either true or high color, a message box opens in which you can choose to preserve the pasted bitmap's palette. Click Yes.

Use the Same Transparent Color

Multiple *.gif* files that use transparency must have the same transparent color so that the graphics can be combined. Therefore, limit the number of transparent colors.

To use the same transparent color:

1. In the Object Editor window, color the area that you want to be transparent with a color not used by the rest of the image. Use the same color for all images with the same palette. For example, color the background of all the images magenta.
2. In the Colors Edit dialog box, make that color transparent. Using the same example, make magenta transparent.

Use Linked *.jpg* files for Complex Color

Reserve linked *.jpg* files for images that require the complex color variations available in true color.

Sometimes you can reduce the number of colors in a *.jpg* without compromising the quality of the image. Reducing the number of colors in the image, and then saving it again as a *.jpg* (and thus, returning the true color palette) can reduce the size of the file.

Adjust the *.jpg* compression to balance file size with image quality (usually between 50% and 70% compression).

Integrating true-color graphics with Rapid's 256-color object palette

Rapid objects such as pushbuttons and switches generate 256-color bitmaps, even if the application displays them in true color. If you want the generated simulation to have true-color buttons or switches, use:

- Bitmap objects (i.e., bitmap or image objects), linked to true-color *.jpg* files, that show each position of the Rapid object;
- Transparent, flat pushbuttons; and
- Logic to show and hide the bitmap objects.

To integrate true-color graphics with Rapid objects:

1. Add a bitmap/image object linked to a *.jpg* that shows the default position of the device's button.

   For example, to display a remote control in true color, add a single bitmap object linked to a *.jpg* that shows all the buttons in the Out position.

2. Superimpose bitmap/image objects, linked to *.jpg* files, for **all** positions of the pushbutton or switch.

   In our example, superimpose *.jpg* files showing the In position of the buttons—using a separate bitmap object for each button.

3. On top of **each** position, add a flat pushbutton object that is fully transparent (i.e., in the Colors Edit dialog box, define **all** color elements as transparent).

4. Use logic that depends on the condition of the pushbutton to show and hide the bitmap/image objects.

Further graphics optimization

You can further reduce the size and number of files in the simulation applet by using a combination of *.jpg* and *.gif* files: a *.jpg* file for the default position of the device's buttons, and *.gif* files for the non-default positions of the device's buttons. No Rapid logic is required for this method.

To combine *.jpg* and *.gif* files for Rapid objects:

1. Add a bitmap/image object linked to a *.jpg* that shows the default position of the device's button.
2. On top of **each** position, add a flat pushbutton object that is fully transparent (i.e., in the Colors Edit dialog box, define **all** color elements as transparent).
3. In the Object Editor, paste a 256-color *.gif* into the pushbutton's In position. Be sure to use the same color palette and transparent color for all (or most) of the *.gif* files.

WORKING WITH AUDIO FILES

Many Rapid applications reference external audio files (*.wav* files) so that the simulated device will sound like the real thing. During code generation, referenced audio files are copied to the generated packages, maintaining the folder hierarchy for the simulation code (see also "Referencing Files" on p. 18). Audio files are **not** compressed, combined, or otherwise changed during code generation.

This sections describes:

- Formatting *.wav* files for Java compatibility.
- Adding *.wav* files to the Rapid application.
- Substituting *.wav* files with Macromedia® Flash™ audio files *(.swf)* for better sound quality.

Using *.wav* Audio Files

RapidPLUS Web Studio supports every type of *.wav* file; however, the early Java runtime environment (JRE 1.1) does not. Large, high-quality sound files should be converted for the widest range of compatibility for the Java environment, and compressed for better performance.

❖ *NOTE: Save a copy of the high-quality sound files for editing.*

For Java compatibility, a reasonable compression ratio, and reasonable sound quality, format *.wav* files as follows:

- 8-bit audio data
- Mono audio
- 8 kilohertz sampling rate
- Global System for Mobile communication (GSM) digital compression

If download time is not a factor, such as for distribution on a CD-ROM, raising the sampling rate from 8 kilohertz will produce more natural sound.

Adding Wave Audio Objects

When you add audio files to the Rapid application, you can designate them to download in the background, after the simulation applet opens. Downloading audio files after the applet opens helps to optimize the initial download time.

To designate *.wav* files to download in the background:

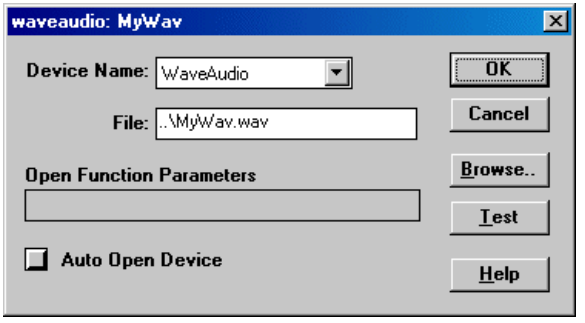
1. Add a wave audio object to your Rapid application and open its More dialog box.
2. In the File box, type or browse to the *.wav* file in the resource folder.

   Leave the Device Name and Open Function Parameters as is.

3 Clear the Auto Open Device check box.

Clearing the check box enables *.wav* files to download in the background. If the check box is not cleared, the simulation user must wait until the *.wav* file finishes downloading before the simulation can open.

The dialog box should appear similar to the following:



4 Click OK.

Make a separate *.wav* file for each sound in the simulation. The simulation performs better this way than if the sounds are all in one file.

Adding Logic for Wave Audio Objects

When entering logic for a wave audio object to load and play a *.wav* file, do NOT load the *.wav* file and play it in the same block of code. Instead, load the file, using the *load:* function, on the entry activity of the mode where the file can be played. If necessary, the file can be loaded in the entry activities of the root mode.

Substituting Flash Audio Files

❖ *NOTE: Only* ***.wav*** *files can be used in the Rapid application. Flash audio files can only be substituted while the simulation applet runs in a Web browser.*

If simulation users have the Macromedia Flash Player plug-in for their Web browsers, you can use Flash audio files (*.swf* files) for improved audio quality. Although Flash audio provides higher-quality sound than the compressed *.wav* files, *.swf* files use more bandwidth. At runtime, the simulation applet

will automatically substitute Flash audio files for *.wav* files if **all** of the following play conditions are met:

- There is an *.swf* file with the same name as the *.wav* file used in the simulation applet. For example, both *MySound1.wav* and *MySound1.swf* must be together in the same resource folder.
- The simulation applet is running in Internet Explorer or Netscape 6.x on Windows.
- The user has the Flash plug-in (5.0 or higher for Internet Explorer, 6.0 or higher for Netscape) or ActiveX installed.
- The download rate is at least 10,000 bytes/second (10 KB/second). To adjust the minimum download, see "The Rapid Application Library" on p. 123.

To prepare Flash audio files:

1. Copy the file *sound_template.fla* from the \\Rapidx\Java folder to the Rapid application's resource folder (i.e., where the application's *.wav* file is located).
2. Rename the file to <*filename*>*.fla*, where <filename> matches the name of the *.wav* file.
3. Open the *.fla* file in Flash.
4. In Flash, import the high-quality *.wav* file (not the GSM file) using File|Import to Library.
5. Select the imported *.wav* file in the Frame Properties panel. (Layer 2 must be selected in the Timeline panel.)
6. Extend the movie so its length matches the length of the original sound file.

   To do this, add a frame at the position where the sound will end. Since there are 12 frames per second, place the frame at

   <length of *.wav* file, in seconds> * 12
7. Export the file as a Movie (*.swf* file) to the same location as the *.wav* file it replaces.
8. Listen to the Flash audio file play in the simulation applet and complete editing the audio files. Only then can you delete the *.fla* file.

When you generate Java code, the *.wav* and *.swf* files are copied to the generated packages. When a user runs the simulation applet, the *.swf* file will be played—if all the play conditions were met.

USING NATIVE AND NONNATIVE FONTS

If you use native system fonts in your Rapid application, you can save download time for the simulation applet. This is because the simulation applet can use system fonts from the simulation user's computer. Nonnative fonts are generated as *.gif* files, which add to the simulation size and increase download time.

Native Fonts

In an effort to be platform independent, Java technology uses five font types, called **logical fonts.** These logical fonts are mapped to native system fonts, called **physical fonts.** The logical fonts recognized by the Java runtime environment are Serif, SansSerif, Monospaced, Dialog, and DialogInput. Physical fonts have names such as Times New Roman, Arial, and Courier New.

For example, Java code usually maps logical fonts to the following physical fonts on Windows and Macintosh operating systems:

- Serif → Times New Roman.
- SansSerif and Dialog → Arial.
- Monospaced and DialogInput → Courier New (Windows) or Courier (Macintosh).

Java technology also uses the following Japanese fonts in Japanese Windows operating systems:

- MS ゴシック
- MS P ゴシック
- MS 明朝
- MS P 明朝

Specifying Fonts Used by Java Code

Sometimes font mapping in Java code does not produce the results you expect. Through Rapid, you can specify physical fonts for the Java code to use directly, instead of by mapping. If the simulation user doesn't have the specified font in the standard location, the Java code will revert to mapping.

To specify the physical font that the Java code will use:

1. In a text editor, open the *Rapidx.ini* file.
2. Under the [JavaSystemFonts] group, add a semicolon at the beginning of the line with the font type you want to specify. This comments out the line and specifies the physical font.

   Example

   By commenting out the font type, the Java code uses Arial directly, instead of mapping to the font type SansSerif.

```
[JavaSystemFonts]

FontName1=Arial
;FontType1=SansSerif
```

Commented out

3. Save and close the file.

Changing Native Fonts to *.gif* files

All of the listed native system fonts are supported by Java technology, but implementation can vary slightly in different versions of the Java environment. You can prevent the variations by changing these fonts to *.gif* files.

To change native fonts to *.gif* files:

1. In a text editor, open the *Rapidx.ini* file.
2. Under the [JavaSystemFonts] group, add a semicolon at the beginning of the lines with the font **name and type** that you want to make into a *.gif*. This comments out the lines and the font will be generated as a *.gif*.

Example

By commenting out the font name and type, the code generation process will generate Arial as a *.gif*.

```
[JavaSystemFonts]

;FontName1=Arial
;FontType1=SansSerif
```

Commented out

3 Change the numbering of the remaining fonts so that they begin with number one (1) and continue sequentially.

4 Save and close the file.

When the simulation applet is generated, the commented out font will be incorporated into the simulation as a *.gif*.

Nonnative Fonts

Nonnative fonts must be installed on the operating system. Code generation groups all the nonnative fonts into a *.gif* file. The code generation process also creates an *.rtf* file for each nonnative font (in the \\<MyApp>\Fonts folder of the code generation output folder). The *.rtf* file contains the information necessary to locate and show the letters from the *.gif* file in the simulation.

Font Recommendations

Use native system fonts when possible, or change the Java code's mapping to a specific physical font.

However, if you must use nonnative fonts, reduce the size of the generated *.gif* by limiting the font's character range in the font object's Advanced dialog box.

OPTIMIZING RAPID LOGIC

As with any Rapid application, efficient and concise use of modes, transitions, triggers, activities, and actions leads to better simulation performance and smaller application size. Therefore, avoid excessive or redundant logic.

The following guidelines keep application logic concise:

- Use parent-to-child mode transitions instead of many transitions and triggers between modes.
- Use concurrent modes or user objects for parts of the system that are functionally independent of each other and can run simultaneously.
- Define user functions to invoke common blocks of logic in several places as a single function.
- Use For and While loops and If...else branches instead of multi-mode loops with conditions.

PLANNING FOR SCENARIOS

Your simulation applet might be combined with a scenario to produce a total multimedia presentation. Because scenarios often require specific functionality from the simulation applet, the overall production flow is more efficient when the scenario author is involved in the design phase of the Rapid application. Good communication with the scenario author can help you focus the application requirements on features, images, and other resources that are required for scenarios. For details about scenario objects and creating scenarios, refer to the manual *Using the Scenario Authoring Tool*.

You should consider the following issues before and during development of the Rapid application.

Download Time

If the simulation and scenarios will be distributed over the Web, you must consider download time. In some cases, you may decide not to simulate features that would require large image files (especially high-quality *.jpg* files).

Another option to reduce download time is by presenting aspects of the product in scenarios, without actually simulating them, and therefore save file

size by reducing the amount of logic. Such features would only be activated in the scenario with a transparent pushbutton clicked by the scenario author. In the application you should "hide" the pushbutton from the simulation user, i.e., place it where the simulation user is unlikely to click.

Determining the Scope of Simulation Functionality

Your Rapid application can accurately simulate most features and functions in a product. However, real-world time constraints and application size may place restrictions on the number (and detail) of the features to simulate.

Example

An electronic dictionary and thesaurus holds over 130,000 words, but only 20 words will be presented in the simulation.

Determining Default Settings

The simulation reverts to its default state at the beginning of each scenario. In general, the simulation defaults should mimic the default settings of the product itself. However, there may be times when a different default is preferred.

Example

A digital camera has a menu system that can only be used when the LCD is on. Even though the default setting for the LCD is off, the simulation will be more visually effective if the simulation's default setting shows the LCD on.

Referencing Objects During Runtime

Objects added or copied during runtime do not have object reference names. Therefore, the scenario author cannot record a user action on the dynamically generated object. Likewise, the object cannot be used as a target object for scenario indicators or simulation prompts.

Discuss these issues with the scenario author to avoid conflicts. You may need to hide and show some objects instead of dynamically generating them.

Adding Supplementary Devices

In some cases, your product by itself is not enough to create effective scenarios—a telephone needs incoming and outgoing calls, an MP3 player needs a computer, an audio/video receiver needs input and output devices. You may need supporting devices to fully represent your product's capabilities.

Supplementary devices do not need to be fully functional, nor do they need to look realistic. In fact, using cartoon-style *.gif* files instead of high-quality *.jpg* files will both reduce the overall file size and keep attention focused on the main device.

Example

The simulation of a cellular telephone is more effective with another telephone that can send and receive calls. If the cellular telephone has call waiting, caller ID, and conference-call capabilities, additional telephones are required to demonstrate these features.

Including Special Images or Audio Files

As with supplementary devices, supporting images and audio files can enhance the effectiveness of a scenario.

Example

The scenario author may require specific music files to showcase music enhancement features on an MP3 player. The scenario author may also require an image to help indicate the size of the player.

Activating the Mouse Object

As in all Rapid applications, you must activate the mouse object (*activate* function) to make use of mouse events and to assign cursor shapes over specified graphic objects.

For scenarios, the active mouse object enables the scenario author to record mouse events over any graphic object. This provides more flexibility when building scenarios.

Updating the Simulation

You can continue adjusting or updating the simulation after work on the scenarios has begun. For compatibility with scenarios that are in progress or already finished, **do not change Rapid object names** between versions. For details on how to update the simulation for the scenario author, see "Updating the Simulation While Scenarios are Being Developed" on p. 70.

ANTICIPATING DIFFERENCES IN ENVIRONMENTS

Your Rapid application and your simulation applet run in different environments:

- The Rapid application runs on your computer's operating system.
- The simulation applet runs on the Java virtual machine.

There are behavioral differences—mostly due to code incompatibilities between the two environments—that you should consider when you design Rapid applications for Java code generation. Behavioral differences between Rapid applications and simulation applets are presented in this section.

Condition-Only Transitions

RAPID BEHAVIOR	JAVA BEHAVIOR
When a transition is based solely on the condition of an object, the Rapid state machine checks the condition every time the object changes.	The Java state machine, which runs in a different thread than the object, may not check the condition of the object on every cycle. Therefore, the state machine may "miss" the exact value of an object.

RAPID BEHAVIOR	JAVA BEHAVIOR
Example Where **powerON_animation** is an animation object with 8 frames, the Rapid state machine evaluates the following condition-only transition every time the value of the animation object changes: **& powerON_animation.currentFrame = powerON_animation.lastFrame-1** Therefore, when the frame value changes from 6 to 7, the condition is triggered.	***Example/Solution*** The Java state machine might not evaluate the condition until after the animation object has already reached the last frame. Therefore, use conditions that are guaranteed to be triggered: **& powerON_animation.currentFrame >= powerON_animation.lastFrame-1**

Condition-Only Transition Recommendations

In general, try to avoid using condition-only transitions. But when one cannot be avoided, use the following recommendation:

- Use a condition that does not require an exact value so that the condition is guaranteed to be triggered (as in the example above).
- In the Rapid application, use a breakpoint at the condition. Run the application to be sure the condition is evaluated only once.

See also "Self-Changing Mode Activities" on p. 39.

Date Object *year* Property

RAPID BEHAVIOR	JAVA BEHAVIOR
Assigning a one- or two-digit integer to the date object's *year* property treats the year as if it is from the **1900s.**	Assigning a one- or two-digit integer to the date object's *year* property treats the year as if it is from the **current century.**
Example The following code calculates the date for the year 1903. **Integer1 := 3** **Date1.year := Integer1**	***Example*** The same code calculates the date for the year 2003.

Exported Events of User Objects Held by Holder Objects

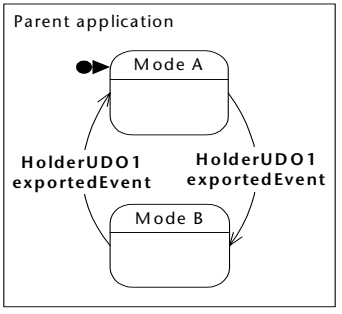
A user object uses exported events to trigger transitions in its parent application. Sometimes a user object is held by a holder object, in which case, the holder can use the user object's exported events to trigger transitions.

When an exported event (from the holder object) triggers a transition from one mode to another, and the **same exported event** triggers a transition back to the original mode, behavioral differences between the Rapid and Java environments occur. The following examples illustrate the differences in behavior and provide a solution when designing applications for Java code generation.

RAPID BEHAVIOR	JAVA BEHAVIOR
The transitions take place at the appropriate time: when the exported event occurs in Mode A, the transition to Mode B takes place; when the exported event occurs in Mode B, the transition to Mode A takes place. (See the following example.)	When the first transition takes place (e.g., the exported event occurs in Mode A thereby triggering the transition to Mode B), the second transition (from Mode B to Mode A) is *immediately* triggered. (See the following example.)

RAPID BEHAVIOR

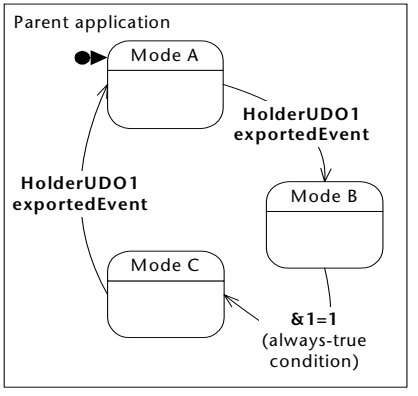
Example
The following diagram illustrates the two transitions:



JAVA BEHAVIOR

Example/Solution
To avoid the problem, add another mode (Mode C). Mode B should contain an always-true condition to create a transition to Mode C.

The following diagram illustrates the workaround in Rapid's logic for the transitions:



❖ *NOTE: You might need to move some or all activities from Mode B to Mode C in order to reproduce the same functionality as in the original application.*

Graphic Display Objects

RAPID BEHAVIOR	JAVA BEHAVIOR
When an image is brought into a graphic display object, its colors are "mapped" to those defined in the object's palette.	When an image is brought into a graphic display object, the image retains its original colors and is not changed (or limited) by the colors in the object's palette.
Example A 256-color image appears in two colors if the graphic display object's palette is set to two colors.	***Example*** A 256-color image appears in 256 colors, even if the graphic display object's palette is set to two colors.

Graphic Display Object Recommendation

To produce the same results in the simulation applet as in the Rapid application, images should use the same color palette as that defined in the graphic display object.

❖ *NOTE: If the color palette of the image does* ***not*** *match the graphic display object's palette, color manipulation functions (such as attributeSetReverse and attributeSetXOR) may not produce expected results.*

Holder Object *holdCopyOf:* Function

RAPID BEHAVIOR	JAVA BEHAVIOR
A copied object has the properties of the object's **current** state.	A copied object has the properties of the object's **default** state (due to size optimization in Java code).

Image Objects

Color and copy functions are slower in the Java environment than in the Windows environment, especially for cumulative changes on image objects.

Image Object Recommendations

Adjust colors using red, green, and blue (RGB) values. Avoid a sequence of cumulative color changes (e.g., avoid the *changeBlueBy:* function) on an image object. If you require a cumulative-type effect, set the color (e.g., with the *setBlue:* function) in different places in the logic.

Jog Dial Objects

RAPID BEHAVIOR	JAVA BEHAVIOR
Clicking any position on the jog dial makes the indicator jump to the "clicked" position.	The jog dial must be clicked and dragged to the required position.

Modulo Function Values

RAPID BEHAVIOR	JAVA BEHAVIOR
The value of the *modulo:* function **cannot** be negative. ***Example*** The function **Integer1 := -8 modulo: 3** assigns the value "1" to Integer1.	The value of the *modulo:* function **can** be negative. ***Example*** The function **Integer1 := -8 modulo: 3** assigns the value "-2" to Integer1.

Object *cursorEntered* and *cursorExited* Events

RAPID BEHAVIOR	JAVA BEHAVIOR
If an object with *cursorEntered* and/or *cursorExited* events is completely covered by another object, the events **are** generated.	If an object with *cursorEntered* and/or *cursorExited* events is completely covered by another object, the events **are not** generated. (This is due to the component model used by Java technology.)
When the mouse button is pressed, the *cursorExited* event is generated when the **cursor exits** the object's bounds.	When the mouse button is pressed, the *cursorExited* event is generated when the **mouse button is released** outside the object's bounds.

Saving Files

RAPID BEHAVIOR	JAVA BEHAVIOR
During runtime, files such as *.rar* and *.rds* **can** be saved in any folder with write permission.	During runtime, these files **cannot** be saved (due to Java security restrictions).

Selecting Multiple Pushbuttons

RAPID BEHAVIOR	JAVA BEHAVIOR
Holding the Shift key while clicking one or more pushbuttons holds them in the In position. After releasing the Shift key, clicking any pushbutton (selected or not) releases **all the buttons** (i.e., they all return to the Out position).	Holding the Shift key while clicking one or more pushbuttons holds them in the In position. After releasing the Shift key, clicking a selected button releases **only that button.**

Self-Changing Mode Activities

RAPID BEHAVIOR	JAVA BEHAVIOR
A condition that is dependent on a self-changing mode activity (e.g., *changeBy:*) is triggered when the mode activity reaches the condition-triggering state.	A condition that is dependent on a self-changing mode activity may or may not be triggered when the mode activity reaches the condition-triggering state. Why? Because the state of an object that is changed by a mode activity is not checked after each cycle (due to Java code optimization).
Example The condition **Integer1 = 2** is triggered when the mode activity **Integer1 changeBy: 1** reaches 2.	***Example*** The condition **Integer1 = 3** might be triggered when the mode activity **Integer 1 changeBy: 1** reaches 3, but the condition **Integer1 = 2** might not be triggered when the same mode activity reaches 2.

See also "Condition-Only Transitions" on p. 32.

Sorting Strings in Array or Data Store Objects

RAPID BEHAVIOR	JAVA BEHAVIOR
String comparisons are sorted byte-by-byte (because Rapid holds strings in double-byte character sets).	String comparisons are sorted character-by-character (because Java code holds strings in Unicode). Generated string array or data store objects may sort differently for non-Latin characters.

Transparent Color Assignment at Runtime

RAPID BEHAVIOR	JAVA BEHAVIOR
At runtime, changing the color of an object to the color that was designated as transparent (during design time), renders the object transparent.	At runtime, changing the color of an object to the color that was designated as transparent (during design time), shows the **color.** The object does not appear transparent.

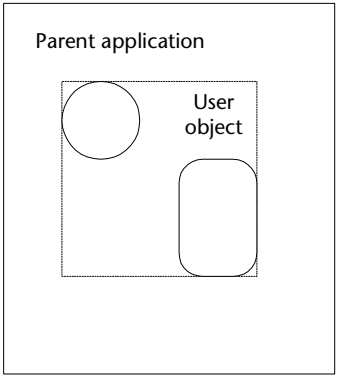
User Function Arguments

RAPID BEHAVIOR	JAVA BEHAVIOR
The property of an object used as the argument of an expression **can** be changed by calling it in a user function. ***Example*** The following user function is defined in *MyUDO1.udo* and exported: **getTime: <Integer:hourInteger> minuteInteger: <Integer:minuteInteger>** **<hourInteger> := SystemTime.hours** **<minuteInteger> := SystemTime.minutes** The function is called in the parent application using the *hour* and *minute* properties of the preformatted time object: **MyUDO1 getTime: Time1.hours minuteInteger: Time1.minutes** The function returns the current system time.	The property of an object used as the argument of an expression **cannot** be changed by calling it in a user function. ***Example*** The same function called in the parent application cannot use properties of an object (it generates an error during code generation). Instead, use integer objects in the parent application: **MyUDO1 getTime: hour_int minuteInteger: minute_int** where **Time1.hours := hour_int** **Time1.minutes := minute_int**

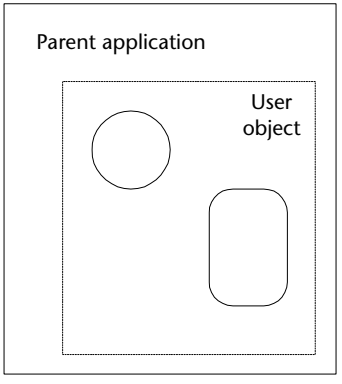
User Object Size and Position

RAPID BEHAVIOR	JAVA BEHAVIOR
A user object can change size during runtime. That is, if a graphic object inside a user object changes size during runtime, the bounds of the user object will change as well.	After Java code generation, the user object keeps the size of its original Object Layout work area.
A user object's Object Layout work area "shrinks to fit" around its graphic objects when added to a parent application.	

Example



Example



User Object Size and Position Recommendation

In each user object, position the upper-left graphic object in the upper-left corner of the Object Layout work area. This way you will not have to adjust for a shift of objects when the user object is added to the parent application (as seen in the previous example).

The placement of the lower-right graphic object is irrelevant, as long as there is room for it to change in size (if required).

CHAPTER 3

Generating Code and the Simulation

Generating a simulation applet has three phases:

i. Determining configuration settings.

ii. Generating Java code from Rapid code.

iii. Making the simulation—i.e., compiling and packaging the Java code.

Immediately after code generation, a batch file, *make.bat,* routinely runs to make the simulation. You can control which commands to run with *make.bat* by setting options in the *make.config* file.

This chapter presents guidelines for:

- Setting *make.config* to configure the "make" process.
- Generating Java code by setting preferences and running the RapidPLUS Web Studio code generation process.
- Optimizing simulation download for better initial simulation performance.

❖ *NOTE: The information presented in the chapter does not apply to the Scenario Player applet. For information about code generation for the Scenario Player applet, see "Generating Code to Create the Scenario Player Applet" on p. 78.*

SETTING CONFIGURATION OPTIONS

The batch file *make.bat* starts the process of compiling Java code, generating selected simulation packages, and launching selected viewers. Setting options in the *make.config* file (formatted as a Windows configuration file) controls the commands that will run in *make.bat*.

To set configuration options for making simulations:

1. In a text editor, open *make.config*, located in the \\Rapidx\Java folder. The file is split into groups, designated by group titles in square brackets:

```
[Make Targets]
[Viewers]
[Make Options]
[Resource Folders]
[System]
[Versions]
```

2. Set options as described in the following tables. You can set Boolean options to yes/no or true/false.

   ❖ ***CAUTION:*** *Blank lines in make.config may cause the make process to fail. You can comment out options that don't apply by typing a semicolon (;) at the beginning of the line.*

3. Save and close the file.

4. (Optional) To determine (or override) the settings for the current simulation only, copy the *make.config* file in the \\Rapidx\Java folder and paste it into the code generation output folder. For easy identification, you can delete all the fields and sections except for those containing the changes you require (but do not leave blank lines between fields).

[Make Targets]

These options determine what kind of simulation will be created. See "Simulations and Simulation Packages" on p. 6 for details about the different simulation packages.

OPTION	DESCRIPTION
unsignedApplet	Compiles and generates applets for all Web browsers. Default setting: yes.
standAlone	Compiles and generates a stand-alone simulation for the stand-alone Prototyper. Default setting: yes.
import	Creates the folder necessary for importing files into the development environment. Used for debugging simulations in Java code. Default setting: no.
scenarios	Creates the folders necessary for building scenarios. Default setting: no.
fepCD	Creates the folders necessary to install (and uninstall) the FEP on a hard disk or CD-ROM. Default setting: yes, commented out.
installFep	Runs the installation program for the FEP automatically after code generation (i.e., on the Rapid application designer's computer). Default setting: yes, commented out.

[Viewers]

These options determine which viewers, if any, to open automatically after making the simulation.

OPTION	DESCRIPTION
runStandAlone	If a stand-alone simulation was generated, this option runs the simulation in the stand-alone Prototyper. Default setting: yes.
appletViewerHTML	If a simulation applet was generated, this option runs the designated file in the Java applet viewer. Default file: *demo_sim.html,* commented out.
defaultBrowserHTML	If a simulation applet was generated, this option runs the designated file in the default Web browser. Default file: *demo_player.html*.
viewer1	If a simulation applet was generated, these options point to additional Web browsers and run the designated files. Default settings: commented out.
viewer1HTML	
viewer2	
viewer2HTML	

[Make Options]

These options are used for locating a custom Scenario Player applet and for debugging simulations.

OPTION	DESCRIPTION
customScenarioPlayer Folder	Specifies the location of a custom Scenario Player applet folder. Default folder: C:\Rapid\Java\myPlayer, commented out.
silent	When set to **yes**, the make process command window only shows file operation headings. When set to **no**, the command window shows file operations, command lines, and file contents. Default setting: yes.
verbose	When set to yes, the make process command window shows all utilities and output files. Default setting: no.
forceExecution	Forces execution in case of errors. Used for debugging the make process. Default setting: no.
cleanUp	Removes temporary files created for the purpose of making the simulation. Default setting: yes.
displayErrorWindow	Opens a dialog box in case errors are detected during compilation. Default setting: yes.

[Resource Folders]

For Rapid applications that reference external files (e.g., *.rar, .rds,* and *.wav* files), this option directs the make process to the folder(s) holding these resources. See "Referencing Files" on p. 18 for more information on referenced files in the Rapid application. See "Specifying a JavaBean Resource Folder" on p. 49 to reference folders with JavaBeans.

OPTION	DESCRIPTION
folder1	Designates a subfolder (under the application folder) to use for referenced files. Default setting: resources.

In addition to a subfolder called "resources," you can create additional subfolders, place files in the application folder (i.e., not in a subfolder), or use a name other than "resources."

Example

To use more folders, type additional lines into the *make.config* file, as in the following example:

```
[Resource Folders]
;=================
; Additional folders for application resources
folder1=resources
folder2=MyFolderA
folder3=MyFolderB
folder4=.
;
```

A period indicates referenced files contained in the application folder (i.e., not in a subfolder)

[System]

These options point to the location of required tools on your system. **If you update or change the location of DashO-Pro or JDK you must update these options.**

OPTION	DESCRIPTION
dashOFolder	Specifies the location of DashO-Pro tool.
jdk13	Specifies the location of JDK 1.3.
satDir	Specifies the location of the Scenario Authoring Tool (SAT). If known, use the location on the scenario author's computer.

[Versions]

These parameters are for version control. **Do not modify these parameters.**

OPTION	DESCRIPTION
lookAndFeel	Version number to be written to the *.html* files for site compatibility issues.
WebStudioVersion	Version number of the Java libraries used to create the simulation applet.

Setting Options for JavaBean Objects

Specifying a JavaBean Resource Folder

For Rapid applications that use JavaBean objects, you can direct the make process to a folder holding the JavaBean resources.

To specify a folder with JavaBean resources:

1. Open the required *make.config* file (see "Setting Configuration Options" on p. 44 for details).
2. At the end of the file, add a group title and group options as in the following example:

Example

The sample subfolder, rapidbeans (shown in bold), sits under the application folder.

```
[External Resource Folders]
; Additional folders for JavaBean resources
folder1=rapidbeans
```

Bypassing JavaBean Objects in DashO-Pro

You might find that JavaBean objects added in the Rapid application function properly, but they do not function in the simulation applet. During code generation, the DashO-Pro optimization and obfuscation process could render the JavaBean objects inoperable. You can solve this problem by adding a group to the *make.config* file that directs DashO-Pro to bypass specific *.class* or *.jar* files.

To direct DashO-Pro to bypass specific *.class* or *.jar* files:

1. Open the required *make.config* file (see "Setting Configuration Options" on p. 44 for details).
2. At the end of the file, add a group title and group options as in the following examples:

Example for .class files

Sample file names are shown in bold.

```
[Preserved Classes]
class1=my.bean1.class
class2=my.bean2.class
```

Example for .jar files

Sample file names are shown in bold.

```
[Preserved JavaBeans]
bean1=MyBean1.jar
bean2=MyBean2.jar
```

UPDATING THE JDK FILE PATH

If you update or change the location of JDK, then you must update the file path that was automatically set during the RapidPLUS installation. The following files are affected:

- *make.bat* (see below)
- *make.config* (see "[System]" on p. 49)
- *dashogui.bat* (see below)

To update the path to JDK in *make.bat*:

1. In a text editor, open the *make.bat* file, located in the \\Rapidx\Java folder.
2. Edit the path to JDK.
3. Save and close the file.

Example

Edit the path as necessary for your system (sample shown in bold):

```
rem The following line should be changed to match system
   configuration
rem !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
SET JDK13=C:\Java\Jdk1.3
```

Path to JDK

To update the path to JDK in *dashogui.bat* (for DashO-Pro):

1. In a text editor, open the *dashogui.bat* file, located in the DashO-Pro folder.
2. Add the path to the Java virtual machine and edit, if necessary, the path to the Java class library.
3. Save and close the file.

Example

Edit the path as necessary for your system (sample shown in bold):

Path to Java virtual machine

```
C:\Java\Jdk1.3\Bin\Java -mx96000000 -classpath
DashoPro.jar;jh.jar;C:\Java\Jdk1.3\JRE\Lib\Rt.jar;DashoProGui
```

Path to Java class library

4 Run *dashogui.bat* to complete the registration and licensing. You must run *dashogui.bat* in order for DashO-Pro to function with the code generation process.

GENERATING JAVA CODE FROM RAPID CODE

After setting the configuration options for the make process, the next step is to set code generation preferences for the Rapid application, and then to finally generate the Java code.

Setting Code Generation Preferences

Set the code generation preferences to designate:

- The output folder for the generated folders and files.
- The batch file *(make.bat)* to run after code generation.
- The language (Java) for code generation.

To set code generation preferences:

1 In the Application Manager, choose Code Generator|Code Generation Preferences. The Code Generation Preferences dialog box opens at the General tab.

2 Under "Source output directory" type in or browse to a folder for the generated folders and files; use either an absolute path or a path relative to the location of the Rapid application.

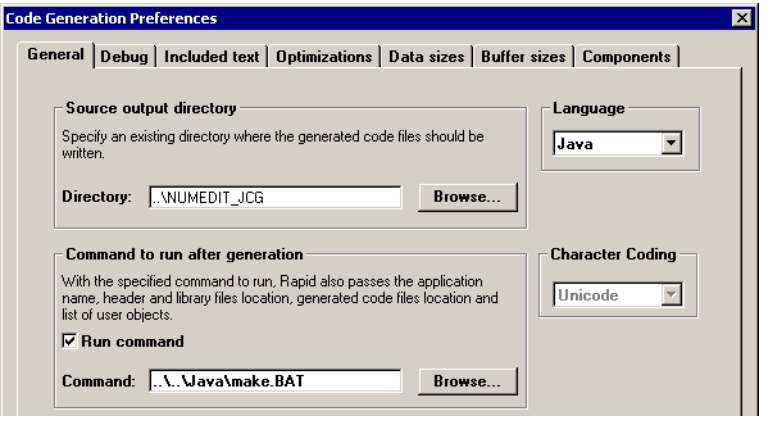
You can enter a folder that does not exist, but not a hierarchy of folders. If you enter a folder that does not exist, you are prompted to let Rapid create the folder.

3 Under "Command to run after generation" select the "Run command" check box and type in or browse to *make.bat,* located in the \\Rapidx\Java folder; use either an absolute path or a path relative to the location of the Rapid application. Be sure to specify the file name and extension.

The command file *make.bat* runs the Java compiler, builds the distribution package(s), and launches the viewer(s) according to configuration options set in *make.config* (see "Setting Configuration Options" on p. 44).

4 Under "Language," select Java from the list.

Your settings should look similar to the following example:



5 Click OK.

❖ *NOTE: You do not need to use the other boxes, the Make Default button, nor the other tabs in the dialog box; these options apply only to C code generation.*

Running the Code Generator

❖ *NOTE: Simulation applets use DashO-Pro to optimize code. DashO-Pro requires that you have write permission in the drive (local or network) where the Rapid application (.rpd) file is located. Without write permission, you will receive a compilation error.*

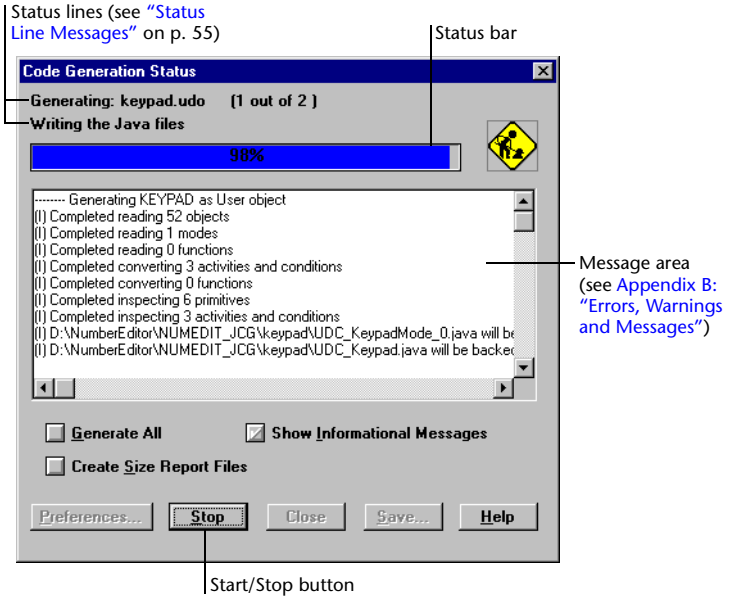
To generate a stand-alone simulation or simulation applet:

1 In the Application Manager, choose Code Generator|Generate Code. The Code Generation Status dialog box opens.

2 Select or clear the Show Informational Messages check box. Select to include informational messages in the message area log; otherwise, only errors and warnings will be reported in the message area log.

❖ *NOTE: The Generate All and Create Size Report Files check boxes apply only to C code generation.*

3 Click the Start button. The code generation process begins.



The status lines track the code generation process. If Show Informational Messages is selected, notices in the message area follow the progress, too. Upon successful completion of code generation, the *make.bat* file runs. A command window opens and shows the status of the make process as it compiles Java files, builds packages, and launches viewers. See "Setting Configuration Options" on p. 44 for details about the make process options.

❖ *NOTES: Do not close the command window until the line "Press any key to continue..." appears. Press any key to close the window—do not use the window's close button, which may cause problems on Windows 98 systems.*

If the stand-alone Prototyper window is running, close the Prototyper window before closing the command window.

4 (Optional) Click the Save button to save messages from the log in a text file.

Status Line Messages

The status line of the Code Generation Status dialog box displays the current code generation activity. It can display any of the following messages (depending on your user objects or application). The order in the table corresponds to that of the code generation process:

STEP	MESSAGE	COMMENTS
1	Processing images and fonts	—
2	Processing objects	—
3	Processing modes	—
4	Processing functions	—
5	Processing properties	—
6	Transforming the logic code	From native Rapid syntax into Java code.
7	Writing Java files	The Java source code files are written to the output folder specified in the Code Generation Preferences dialog box.
8	Issuing the command: *make.bat*	Executes the command specified in the Code Generation Preferences dialog box. ❖ *NOTE: This message appears only after the main application is generated, and only if there were no code generation errors.*

The Code Generation Process

The code generation process consists of three phases:

i. The Code Generator briefly activates the simulation in the Prototyper window to record the initial state of all graphics in the simulation. This information is for the image packager (see "Optimizing Image Download" on p. 57).

ii. The Code Generator opens and generates user objects, one after another; it then opens and generates the main application. The status line and status bar in the Code Generation Status dialog box track each generation process separately.

iii. After generating Java code from Rapid code, the batch file *make.bat* runs a process that compiles and packages the simulation code and resources, depending on settings in the *make.config* file (see "Setting Configuration Options" on p. 44). The process proceeds as follows:

   a. The Java utility *make.jar* runs.

   b. Configuration files are read and processed.

   c. Batch files are created in the output folder.

   d. The *.jlp* files are processed.

   e. Subfolders are created in the output folder.

   f. Web files are copied to the SitePack folder and template parameters (such as simulation name, height, and width, and the Java package name of the generated simulation) are replaced by actual values.

   g. Scenario files are copied to the SitePack folder. The *.html* files with <SCEN_FILEx> and <SCEN_NAMEx> parameters are updated.

   h. The resource folder(s) are copied from the application folder and *.rar* and *.rds* files are processed.

   i. Generated Java files are compiled.

   j. A script file *(.dop)* for DashO-Pro is automatically created. The script file contains class and function names to be preserved (according to a predefined list).

   k. DashO-Pro optimizes the code and creates the \\build\out output folder.

   l. The \\build\out folder is compressed by the CabArc utility to produce the *.cab* file.

   m. The simulation applet is run in the selected environment(s), according to configuration settings.

   n. Unnecessary files are removed.

   For details regarding the contents of the code generation output folder, see Appendix A: "Code Generation Output."

OPTIMIZING IMAGE DOWNLOAD

Code generation automatically sorts images into download categories. Images that appear in the initial state of the simulation download first so that the simulation applet can open and start running, while images that are hidden in the initial state of the simulation download in the background.

The automatic sorting is not always optimal for initial simulation performance because Rapid doesn't know what actions simulation users are expected to do first. Below is an explanation of the image download categories, followed by instructions for changing the categories to optimize the initial performance.

Image Download Categories

Code generation automatically sorts images into one of three download categories: LOCAL, REMOTE, and BACKGROUND. Each category appears differently to the simulation user.

LOCAL Download Category

LOCAL images are small (up to 2 KB) *.jpg* files that appear in the initial state of the simulation. Because multiple server connections are inefficient for small files, these images are included with the *.jar* or *.cab* file.

- **Loading order:** First.
- **What the simulation user sees:** While LOCAL images download, simulation users see an animation in their Web browser. (To change the animation, see "Customizing What the User Sees During Download" on p. 68.)

REMOTE Download Category

REMOTE images are larger *.jpg* files and all the *.gif* files that appear in the initial state of the simulation.

- **Loading order:** Second.
- **What the simulation user sees:** While REMOTE images download, simulation users see a progress bar and the message "Loading simulation, please wait."

After REMOTE images finish downloading, simulation users can see and operate the simulation.

BACKGROUND Download Category

BACKGROUND images are images that are hidden in the initial state of the simulation.

- **Loading order:** Third.
- **What the simulation user sees:** Standard browser indicators may appear (file names in the status bar, for example), but loading takes place in the background of the visible, running simulation.

Changing the Download Category

After REMOTE images finish downloading, the simulation user can see and operate the simulation while BACKGROUND images continue to download. If initial user actions require a BACKGROUND image that has not finished downloading, the image will not be visible until its download is complete. Changing the download category from BACKGROUND to REMOTE can alleviate this delay.

Example

The initial view of a VCR simulation (called VCR1) shows a large front panel of the device and a small, blinking remote control. Simulation users are expected to click the remote control to see an enlarged view of the remote control, *lg_remote.jpg,* and use it to operate the device.

During code generation, this image was automatically assigned to BACKGROUND (since it is initially hidden); you can change the category to REMOTE for better initial performance.

To change the download category of an image:

1. Open the file *<MyApp>_ImagePackager.txt,* located in the code generation output folder.

   This file contains three tables of images (LOCAL FILES, REMOTE FILES, and BACKGROUND FILES) followed by tables of color palette entries for the different 256-color (or less) *.gif* palettes in the application.

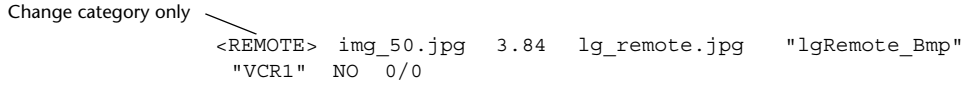
2. Locate the images that you want to change. You can search by parameters such as the original name of the image or the Rapid object name.

   Example

   ```
   <BACKGROUND>  img_50.jpg   3.84   lg_remote.jpg    "lgRemote_Bmp"
    "VCR1"  NO  0/0
   ```

3 Change **only** the category word(s) in the angle brackets. Do not move the image information to another category.

Example



4 Save and close the file.

5 Regenerate the application in Rapid. Changed files automatically move to their new download category.

❖ *NOTE: You can change the loading category of any image; however, most changes will probably be from BACKGROUND to REMOTE.*

SPLITTING LARGE FILES

When a Rapid application generates a *.jar* file over 1 MB, you may encounter difficulties compiling the applet or downloading it through Netscape browsers. One solution may be to split certain *.class* files within the *.jar* file.

Code generation creates two user-defined component (UDC) files for the main Rapid application *(.rpd)* and for each of its user objects *(.udo)*. Each UDC is a *.class* file within the *.jar* file.

Example

The Rapid application, *ABC.rpd* contains the user objects *ABCu1.udo* and *ABCu2.udo*. After code generation, opening the archive *ABC.jar* with a *.zip* compatible application program, shows all the applet *.class* files, including the following:

UDC_ABC.class
UDC_ABCMode_0.class
UDC_ABCu1.class
UDC_ABCu1Mode_0.class
UDC_ABCu2.class
UDC_ABCu2Mode_0.class

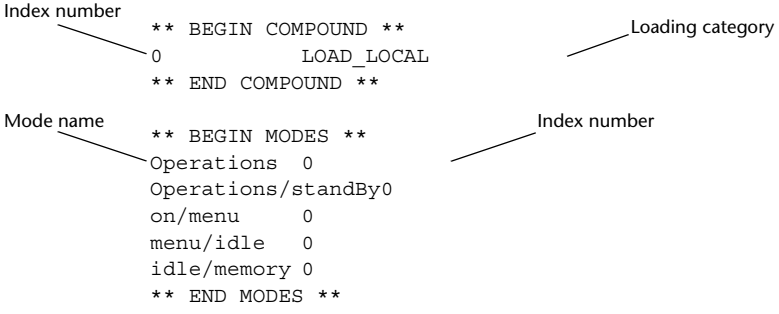
The UDC *.class* files hold the code and logic for the Rapid objects as follows:

- ***UDC_<MyApp>.class*** (or ***UDC_<MyUDO>.class***)
  These files hold the code for all the Rapid objects used. If one of these files is larger than 250 KB, you may need to create more user objects. (This is rarely necessary.)
- ***UDC_<MyApp>Mode_0.class*** (or ***UDC_<MyUDO>Mode_0.class***)
  These files hold logic for all the modes. If one of these files is large enough to cause a runtime error, you can split it into two or more files. To split this file, you must edit the packager file *UDC_<MyApp>.pkgr.*

To split the *UDC_<MyApp>Mode_0.class* file (modify the *UDC_<MyApp>.pkgr* file):

1 In a text editor, open the *UDC_<MyApp>.pkgr* file, located in the <MyApp> subfolder of the Java code generation output folder. It should appear similar to the following example:

Example




```
** BEGIN COMPOUND **
0           LOAD_LOCAL
** END COMPOUND **

** BEGIN MODES **
Operations  0
Operations/standBy0
on/menu     0
menu/idle   0
idle/memory 0
** END MODES **
```

2 Between the headings BEGIN COMPOUND and END COMPOUND, is an index number (0) and loading category (LOAD_LOCAL). Copy and paste this line so it appears as in the following example:

Example

```
** BEGIN COMPOUND **
0     LOAD_LOCAL
0     LOAD_LOCAL
** END COMPOUND **
```

3 Change the index number of the new line to one (1).

Example

```
** BEGIN COMPOUND **
0     LOAD_LOCAL
1     LOAD_LOCAL
** END COMPOUND **
```

4 Under the heading BEGIN MODES, is a list of modes in the simulation with the index number zero (0). Change the index number of approximately half the modes to one (1).

Example

```
** BEGIN MODES **
Operations0
Operations/standBy0
on/menu1
menu/idle1
idle/memory1
** END MODES **
```

5 Save and close *UDC_<MyApp>.pkgr.*

6 Regenerate the application in Rapid. The *UDC_<MyApp>Mode_0.class* file splits into two files. A file called *UDC_<MyApp>Mode_1.java* is created for the new index number. These files contain the generated Java code for the logic.

You can use the same procedure to split the *UDC_<MyApp>Mode_0.class* file into more than two files.

MAKING SIMULATIONS WITHOUT REGENERATING CODE

After the initial code generation and make process, you can run the make process again without regenerating code, i.e., outside of the RapidPLUS Web Studio environment. This is convenient for updates or changes that do not affect the functionality of the simulation, such as making simulation packages or linking scenarios to the Scenario Player applet.

Running the Make Process

The make process compiles Java source code, generates simulation packages, and launches viewers.

To run the make process:

1. Adjust the configuration settings in the *make.config* file to determine the type of simulation and package you want to generate (see "Setting Configuration Options" on p. 44).
2. Run the *make.bat* file located in the code generation output folder.

For a description of the processes that run through the *make.bat* file, see "The Code Generation Process" on p. 55.

Running the Link Process

The link process, which is part of the make process, creates and updates scenario information in the Scenario Player applet, copies the scenario files to the SitePack folder, and creates *.zip* files for scenarios. You can run the link process as part of the make process, or you can run it separately.

To run the link process through the make process:

1. Create a folder called Scenarios in the code generation output folder. Place scenario *.wav* files and the Scenario Authoring Tool's exported *.txt* files in the folder.
2. Run the *make.bat* file located in the code generation output folder.

To run the link process from the ScenPack folder, see "Distributing Simulations to Scenario Authors" on p. 69.

CHAPTER 4

Finalizing Simulations for Distribution

After you've generated and optimized the simulation applet, you should run and test it.

A simulation can run from a number of different files, with or without the Scenario Player applet. You can test the simulation applet performance on a local or network drive, and then test its download time and performance from a Web server.

After testing, you can distribute simulation packages to a scenario author or a Web designer simply by sending one of the generated folders.

This chapter presents information about:

- Running and testing generated simulations.
- Adjusting simulation download.
- Distributing and publishing simulations.

RUNNING GENERATED SIMULATIONS

The code generation process can produce different simulation packages with different types of simulations. The simulation packages, and the types of simulations they can contain, are:

- **Simulation-only packages**, which contains a stand-alone simulation or a simulation applet.
- **Scenario authoring package**, which contains a simulation applet with the Scenario Player applet.
- **FEP CD package**, which contains a simulation applet that requires FEP browser support.

For details about the different packages and types of simulations, see "Simulations and Simulation Packages" on p. 6. For details about setting the code generation options, see "[Make Targets]" on p. 45.

❖ *NOTE: Files for the Scenario Player applet are generated whenever a simulation applet is generated, however, the Scenario Player applet is hidden in simulation-only packages.*

You can run simulations for various reasons, including testing simulation functionality or viewing the simulation with scenarios. The package(s) you generate and your reasons for running a simulation determine which file you run. The following sections describe these files and explain how to run them.

Testing Functionality of Stand-Alone Simulations

To test simulation functionality in a Java or Java-like environment, there are two different *.bat* files that you can generate and launch relatively quickly:

- *runStandAlone.bat*
- runApplet.bat

Selecting the File to Run

You can run the simulation from the *runStandAlone.bat* file and/or the *runApplet.bat* file. The procedure for running the simulation follows the descriptions.

runStandAlone.bat

This file is for testing the application **without** generating a simulation applet. Because there is no applet, the make process finishes quickly.

LOCATION	VIEWER	DESCRIPTION
StandAlone folder of the code generation output.	Stand-alone Prototyper.	Runs on an application developer's computer (i.e., a computer with the \\Rapidx\Java folder).

runApplet.bat

This file is used for testing the simulation applet **without** using a Web browser.

LOCATION	VIEWER	DESCRIPTION
Root folder of the code generation output.	Java applet viewer.	Runs the simulation applet in the separate viewer window.

Running the Simulation

To run a simulation for functionality testing:

1 Locate the file for the type of simulation you want to run:

- The *runStandAlone.bat* file in the StandAlone folder of the code generation output.
- The *runApplet.bat* file in the root folder of the code generation output.

2 Run the selected file.

Viewing Simulation Applets in a Web Browser

You can open a simulation applet (and the Scenario Player applet) in a Web browser from one of four *.html* files. The file *demo_sim.html* is used for viewing only the simulation; the other files are used for viewing the simulation with different kinds of scenarios.

Installing FEP Browser Support

If the Rapid application includes an FEP object, and you generated an FEP CD package, simulation users must install FEP browser support on their machines.

Simply run the file *FepInstall.exe,* located in the \\CdPack\install subfolder of the code generation output folder.

Selecting an *.html* File to Run

All the *.html* files are all located in the SitePack subfolder (or \\ScenPack\SitePack subfolder) of the code generation output folder.

The following table describes the *.html* files. Procedures for running the simulation applets follow the descriptions.

FILE	DESCRIPTION
demo_sim.html	This file is used to run only the simulation applet. The Scenario Player applet is launched—but is hidden.
demo_player.html	This file is used to run the simulation applet and the Scenario Player applet. The Web page is arranged in an HTML `TABLE` element.

FILE	DESCRIPTION
demo_playerviewnew.html	This file is used to run the simulation applet and the Scenario Player applet with the JavaScript user interface (see Appendix C: "JavaScript User Interface for Scenarios"). The Web page is arranged using HTML `FRAMESET` elements. ❖ *NOTE: This file is for demonstration purposes; it is not a supported interface.*
demo_playerviewold.html	This file is used to run the simulation applet and the Scenario Player applet with the JavaScript user interface for scenarios that only use simulation prompts (see Appendix C: "JavaScript User Interface for Scenarios"). The Web page is arranged using HTML `FRAMESET` elements.

For more details about the content of each *.html* file, see Appendix A: "Code Generation Output."

Looking at the code: ways to run applets

Relevant *.html* files can run the simulation applet (and the Scenario Player applet) in one of two ways:

- By the `rapid_applet` and `player_applet` JavaScript functions.
- By the *.html* element `APPLET` (one for each applet).

The JavaScript functions (called from the *.html* files and defined in the *appletutilslf2.js* file) dynamically build the `APPLET` elements in a Web browser during runtime; these functions are used by default.

Commented-out `APPLET` elements appear in the *.html* files as samples, in case you want to launch the applets directly from the *.html* file.

❖ *NOTE: Both the JavaScript functions and the sample APPLET elements use parameters that are automatically updated during the make process. For a description of these simulation and scenario parameters, see Appendix A: "Code Generation Output."*

ADJUSTING SIMULATION DOWNLOAD

If the simulation will be distributed over the Web, it would be prudent to time the simulation download and test its performance over a dial-up modem connection. By adjusting certain aspects of the simulation or the generated files, you can optimize the simulation download, and thus enhance the simulation user's experience.

Changing the Download Rate

If, after testing the download rate, you want to further optimize the simulation, refer to the following sections:

- "Working with Graphic Files" on p. 19.
- "Working with Audio Files" on p. 22.
- "Using Native and Nonnative Fonts" on p. 26.
- "Optimizing Rapid Logic" on p. 29.
- "Optimizing Image Download" on p. 57.
- "Splitting Large Files" on p. 59.

Customizing What the User Sees During Download

You can customize what the simulation user sees during the initial download to suit the look and feel of the product or Web site. You can use an image, an animated image, text, or other Web medium (e.g., a Macromedia® Flash™ movie).

By default, the generated file *demo_player.html* uses *wait.gif,* an animated image that appears in the Web browser during the download of the *.jar* or *.cab* file.

To change what the user sees during download:

1 In a text editor, open *demo_player.html*.

To make a global change: open *demo_player.html* located in the \\Rapidx\Java\HTML folder. This change affects every simulation generated from Rapid.

To make a local change: open *demo_player.html* located in the generated SitePack folder. This change affects the current simulation only and will be

overwritten the next time the simulation is generated from the Rapid application.

2 Find the JavaScript that is labeled with the comment “Code for loading animation” and update as required.

DISTRIBUTING SIMULATIONS (PUBLISHING)

When you generate simulation applets, you can also generate simulation distribution packages for scenario authors and Web designers.

Distributing Simulations to Web Designers

When you generate a simulation applet, the make process automatically creates a SitePack subfolder in the code generation output folder. The SitePack folder contains only the files needed to run simulations from a Web server.

Send the SitePack folder to the Web designer. The Web designer should put the SitePack folder on the Web server, and link to (or otherwise incorporate) the appropriate *.html* file. For details regarding the different kinds of *.html* files, see “Running Generated Simulations” on p. 64.

Distributing Simulations to Scenario Authors

When the simulation applet is part of a multimedia presentation that includes scenarios, the scenario author will need to work with the simulation in the Scenario Authoring Tool.

Send the ScenPack folder to the scenario author. After creating scenarios, the scenario author will link them to the simulation (using the utility file *link.bat,* included in the ScenPack folder) and send the Web designer the \\ScenPack\SitePack folder. For details regarding creating and linking scenarios, refer to the manual *Using the Scenario Authoring Tool*.

About the SitePack and \\ScenPack\SitePack folders

The code generation output folder includes a SitePack subfolder (for Web designers) and a ScenPack subfolder (for scenario authors). The ScenPack folder includes a **copy** of the SitePack folder.

After creating scenarios and linking them to the simulation, the Web designer should receive this copy of the SitePack folder—which contains the scenarios—to put on the Web server.

Updating the Simulation While Scenarios are Being Developed

You can continue adjusting or updating the simulation after work on the scenarios has begun. In that case, you will have to update the scenario package and send a new copy of the ScenPack folder to the scenario author.

To update a simulation after work on scenarios has begun:

1. Regenerate the simulation applet with a ScenPack folder (see "[Make Targets]" on p. 45).
2. Send the ScenPack folder to the scenario author.

Before playing scenarios, the scenario author must copy their own \\ScenPack\Scenarios subfolder to the updated ScenPack folder. Then the scenario author must re-export and re-link scenarios.

Creating CD-ROMs for Macintosh Computers

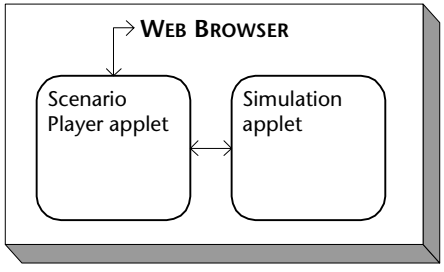
When using an IBM-compatible computer (PC) to create a CD-ROM for Macintosh computers, file names longer than eight characters are truncated, rendering the files unusable on a Macintosh computer. You must use Macintosh-compatible software to make CD-ROM images.

CHAPTER 5

Developing the Scenario Player Applet

A simulation applet can be combined with scenarios to create a multimedia presentation of a product. **Scenarios** integrate simulation actions (such as button presses and changing views) with text, sound, and visual aids. Scenarios are produced with e-SIM's Scenario Authoring Tool. They run in a separate applet called the Scenario Player applet.

The **Scenario Player applet** controls the appearance and behavior of scenarios. It communicates with the simulation applet and the Web browser or applet viewer via its API.



Similar to simulation applets, the Scenario Player applet is generated from a Rapid application. The main difference in the development cycles of these two applets is that the Scenario Player applet is usually developed once for an entire Web site or suite of products, while a new simulation applet must be developed for each product.

Creating the Scenario Player applet and integrating it with a simulation applet consists of three phases:

1 **Building the Scenario Player application in Rapid.**
The Scenario Player applet is generated from a specialized Rapid application that contains one or more user objects (*.udo* files) that are

supplied with RapidPLUS Web Studio. Each user object governs a specific aspect of the Scenario Player applet's functionality or appearance.

Because the Scenario Player applet is generated from a Rapid application, its user interface is as flexible and adaptable as any other Rapid application built for Java code generation.

2 **Generating Java code to create the Scenario Player applet.**
After the Scenario Player application has been built, Java code is generated. This code will be used to build the Scenario Player applet when the simulation applet is built.

3 **Linking a simulation applet to the Scenario Player applet.**
In order to link the simulation applet with the Scenario Player applet, the simulation's *make.config* file must be modified to specify a path to the Scenario Player applet. When the simulation applet is generated and compiled, the Scenario Player applet is included in the code generation output.

An additional file, *scenariotemplate.txt,* defines the appearance of visual aids that appear in the simulation applet while a scenario is running.

This chapter presents:

- Information about building the Scenario Player application.
- The user objects that are added to the application and their API.
- Steps for creating and modifying the Scenario Player applet's GUI.
- How to generate the Scenario Player applet.
- How to modify indicators and demonstrators.

❖ *NOTE: For details about scenario objects and creating scenarios, refer to the manual, Using the Scenario Authoring Tool.*

BUILDING THE SCENARIO PLAYER APPLICATION

You can customize the appearance of the Scenario Player applet to visually complement the style of your product or company. You can also design the functionality of the Scenario Player applet in order to control the degree of interaction with the user, with the Web browser or applet viewer, and with the simulation applet.

For example, you can use custom fonts, arrange the presentation of scenario lists and text prompts, add images or rich media content, and control user interaction with the scenarios.

What you see in the Rapid Prototyper while building the Scenario Player application is different, though, from what you will see in the Web browser. This is because the content—such as scenario titles and prompt text—is not available until the Scenario Player applet is combined with a simulation applet, linked to scenarios, and opened in a Web browser. Therefore, you must keep the final product—the Scenario Player applet—in mind as you build its application.

This section presents a brief description of the building blocks for the Scenario Player application and contains usage examples of the Scenario Player applet.

Scenario Player Building Blocks

A Scenario Player application must contain certain elements:

- **Graphic objects such as text displays, buttons, and images**
  These objects allow users to navigate scenarios, view scenario content, control playback, and determine volume level. They also present the "look" of the Scenario Player applet.
- ***ScenarioPlayer_Manager.udo***
  Through this user object, the Scenario Player applet controls the functionality of the scenarios (e.g., play, pause, stop) and receives information from the simulation applet.
- ***ScenarioPlayerUI_Agent.udo***
  Through this user object, the Scenario Player applet processes information about scenario prompts and Supplemental Content files.
- **Nongraphic objects such as arrays, data stores, and strings**
  These objects hold scenario information that is used by the main application and the user objects.

Optionally, the application can contain additional elements:

- *AppletContext_Manager.udo*
  Through this user object, the Scenario Player applet communicates with the Web browser/applet viewer.
- *Simulation_Agent.udo*
  Through this user object, the Scenario Player applet manipulates the simulation applet and obtains information about actions performed by the user on the simulation applet.
- Players for Supplemental Content files
  Supplemental Content files are external media, such as a Flash movie or an animated *.gif*, that can be used to enhance scenarios. In order for them to play in the scenarios, their players must be added to the application.

Scenario Player Usage Examples

The following examples present a typical Scenario Player applet, and give a brief description of Rapid objects that control the applet's appearance and functionality.

The scenarios in these examples use a hierarchy of group headings and steps. A **group heading** is a high-level topic, under which a number of scenarios can be organized. A **step** is a division of a scenario, such as one part of a procedure or a specific feature of the device. For more information on organizing scenarios under group headings and by steps, refer to the *Using the Scenario Authoring Tool* manual.

Example 1: Scenarios running in the navigation area

After opening this sample Scenario Player applet in a Web browser, the user sees the following elements:

❶ The navigation area displays the available scenarios. In this example, there are five visible group headings. Related scenarios are arranged under each heading.

Scenario links are hyperlinks that display scenario titles and allow users to select the various scenarios. Clicking a link selects a scenario and displays its steps (see Example 2 for a sample of a selected scenario).

- The group headings and scenario links are displayed in separate display objects. In this example, text widget JavaBeans are used, but text display and graphic display objects could be used.
- Text for group headings and scenario links come from the exported file *scenarios.txt*.
- The *scenarios.txt* file is loaded to the application using the *loadScenariosStructure:* function of *ScenarioPlayer_Manager.udo*.

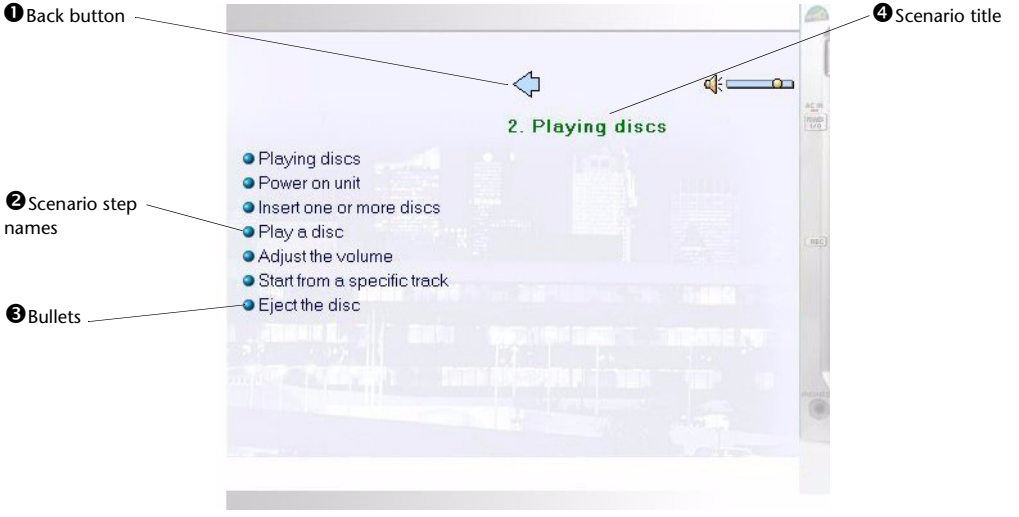
❷ Add scroll buttons or a scroll bar to allow users to move through the scenario list.

❸ The scenario volume can be changed by users. The *setVolume:* function of *ScenarioPlayer_Manager.udo* controls this feature.

❹ A background image can be added to the main application to provide visual interest, a company logo, or other visual information.

Example 2: A scenario's steps displayed in the navigation area

After clicking a scenario link, the user sees the following elements:



❶ Back button, which when clicked, returns to the list of scenarios.

❷ Step names for the selected scenario. The text for step names is entered in the Scenario Authoring Tool. Each step name is a hyperlink that starts playback at a specific point in the scenario.

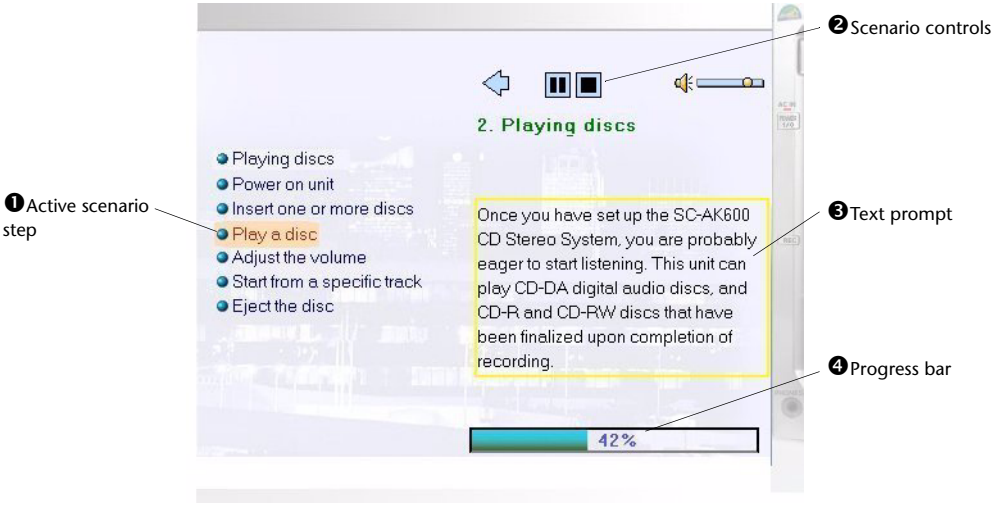
- The step name comes from the exported scenario file *<MyScenario>_prompt.txt*.
- The scenario step names are loaded using the *getScenarioSteps:* function of *ScenarioPlayer_Manager.udo*.
- The *playStep:* function of *ScenarioPlayer_Manager.udo* is used to set the scenario conditions for that step and to start playing the scenario.

❸ A bitmap is used to provide the image for bullets.

❹ The scenario title is displayed in a display object. In this example, a text widget JavaBean is used.

Example 3: Scenario playback in the navigation area

While the scenario plays, the user sees these elements:



❶ The step that is currently playing is highlighted.

- A filled frame is used to highlight the active step name.
- A user function determines the location of the frame.

❷ The scenario controls (buttons) allow the user to pause or stop the scenario, using the *pause* and *stop* functions of *ScenarioPlayer_Manager.udo*.

❸ Text prompts provide an onscreen narration for the actions that occur on the simulation.

- Text prompts are added in the Scenario Authoring Tool (in this case, as list prompts) and their contents are added to the exported scenario file *<MyScenario>_prompt.txt*.
- Text is obtained via the *promptText* property of *ScenarioPlayerUI_Agent.udo*.
- The text for the first prompt is displayed when the event *firstPrompt-Event* occurs. The text changes each time the event *normalPromptEvent* occurs. The text for the last prompt is displayed when the event *lastPromptEvent* occurs. All of the events are available in *ScenarioPlayerUI_Agent.udo*.

❹ The progress bar informs the user how much of the scenario has already played.

- In this example, a user object controls the display of the progress bar.
- The *getScenarioProgress* function of *ScenarioPlayer_Manager.udo* is used to obtain the progress value.

GENERATING CODE TO CREATE THE SCENARIO PLAYER APPLET

Generating Java code for the Scenario Player applet is similar to generating code for simulation applets. For details about the code generation dialog boxes, see "Generating Java Code from Rapid Code" on p. 52.

To generate code from the Scenario Player application:

1 In the Code Generation Preferences dialog box:

- Under "Source output directory" type in or browse to a folder for the generated folders and files; use either an absolute path or a path relative to the location of the application.

  You can enter one folder that does not exist, but not a hierarchy of folders. If you enter a folder that does not exist, you are prompted to let Rapid create the folder.
- Under "Command to run after generation" select the "Run command" check box and type in or browse to *makePlayer.bat,* located in the \\Rapidx\Java folder; use either an absolute path or a path relative to the location of the application. Be sure to specify the file name and extension.
- Under "Language," select Java from the list.

❖ *NOTE: You do not need to use the other boxes, the Make Default button, nor the other tabs in the dialog box; they apply to C code generation.*

2 In the Code Generation Status dialog box, click the Start button.

The status lines track the code generation process. If Show Informational Messages is selected, notices in the message area follow the progress, too. For an explanation about the status line messages, see p. 55.

3 (Optional) Click the Save button to save messages from the log in a text file.

Once the Scenario Player applet is complete, you must package it with the simulation applet.

To package the Scenario Player applet with the simulation applet:

1 Generate the Scenario Player applet.

2 Edit the *make.config* file to include the path of the generated Scenario Player applet. See "[Make Options]" on p. 47 for details.

3 Generate the simulation applet as usual. Be sure to specify *make.bat* as the file to run in the Code Generation Preferences dialog box. See "Generating Java Code from Rapid Code" on p. 52 for details.

THE USER OBJECTS' API

This section presents the API for the Scenario Player application's user objects:

- *ScenarioPlayer_Manager.udo*
- *ScenarioPlayerUI_Agent.udo*
- *AppletContext_Manager.udo*
- *Simulation_Agent.udo*

ScenarioPlayer_Manager.udo Reference

This user object controls scenario operations, including:

- Playback control.
- Volume control.
- Scenario structure display information.
- Simulation status during playback.

The available API is described in the following tables.

Functions of ScenarioPlayer_Manager.udo

This user object has one property, *self*.

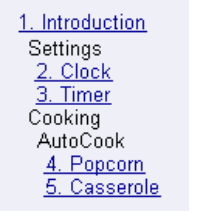
FUNCTION	DESCRIPTION
allowFreePlayWhile Paused: <Integer>	Enables (1) or disables (0) user interaction with the simulation when scenario playback is paused. When scenario playback resumes, the simulation returns to the state at which it was paused. **Parameters:** 0 (by default) or 1.
disableSimulation	Disables user interaction with the simulation. This function is particularly useful when you want the simulation to remain disabled after the scenario ends. ***Example*** The end of one scenario uses the *disableSimulation* function. The simulation user is asked to play another scenario that continues where the first one left off. The next scenario continues, without a need to account for user actions.
enableSimulation	Enables user interaction with the simulation.
getBufferingProgress	Returns the percentage of the voice-over data buffer that is already filled. **Return value:** a number in the range of 0–100.
getScenarioProgress	Returns the percentage of the total length of time that the scenario has already played. **Return value:** a number in the range of 0–100.
getScenarioSteps: '<String>'	Returns a list of step names from the named scenario. **Parameter:** a string; the name of the scenario file. **Return value:** a one-dimensional string array.
getVolume	Returns the level of the voice-over volume. **Return value:** a number in the range of 0–1.

FUNCTION	DESCRIPTION
loadScenario: '<String>'	Loads the specified scenario, ready for playback. **Parameter:** a string; the name of the scenario file.
loadScenariosStructure: <Data_Store>	Loads the scenario structure display information for group headings. **Parameters:** a data store with the following fields (read at runtime from the file *scenarios.txt,* located in the SitePack subfolder). • name: a string record that contains the name of the scenario or folder. • level: an integer record that specifies the nesting level of the scenario or folder (1=top level). • folder: an integer record that specifies whether this record describes a scenario name (0) or a folder name (1). • file: a string record that contains the name of the scenario script file (only meaningful when the value of "folder" is 0). • numbered: an integer record that specifies whether the scenario is numbered (1) or bulleted (0) (only meaningful when the value of "folder" is 0). See the following example.

FUNCTION	DESCRIPTION
	Example The content of **scenarioStructure_DS** is dynamically filled in at runtime (as shown in the Inspector):

	Name	name	level	folder	file	numbered
1		'Introduction'	1	0	'scenario1.zip'	1
2		'Settings'	1	1	''	0
3		'Clock'	2	0	'scenario2.txt'	1
4		'Timer'	2	0	'scenario3.zip'	1
5		'Cooking'	1	1	''	0
6		'AutoCook'	2	1	''	0
7		'Popcorn'	3	0	'scenario4.zip'	1
8		'Casserole'	3	0	'scenario5.zip'	1

The scenario structure would appear in the Web browser as follows:



muteSimulationSound: <Integer>	Mutes (1) or unmutes (0) the simulation sound. **Parameters:** 0 (by default) or 1.
pause	Pauses the scenario playback.
pauseAfterEachStep: <Integer>	Pauses (1) the scenario at the completion of each scenario step, or continues (0) and plays straight through the scenario. **Parameters:** 0 (by default) or 1.
play: '<String>'	Restarts the simulation applet and plays the specified scenario from the beginning. **Parameter:** a string; the name of the scenario file.

FUNCTION	DESCRIPTION
playStep: <Integer>	Plays the specified step of the loaded scenario. See "Playing a Scenario from a Specific Step" on p. 84. **Parameters:** an integer; a step number greater than or equal to zero (0).
playWithoutRestart: '<String>'	Plays the specified scenario without restarting the simulation. **Parameter:** a string; the name of the scenario file.
restartSimulation	Restarts the simulation applet.
resume	Resumes the scenario playback after it was paused.
setSimulationVolume: <Number>	Sets the simulation applet volume to the specified level. **Parameter:** a number in the range of 0–1.
setVolume: <Number>	Sets the voice-over volume to the specified level. **Parameter:** a number in the range of 0–1.
setWaitCursor: <Cursor>	Sets the wait cursor image to the specified cursor. **Parameter:** a system cursor. ***Example*** **ScenarioPlayer_Manager setWaitCursor: Arrow**
showPromptsWhile Skipping: <Integer>	Shows (1) or hides (0) list prompts between steps when steps are skipped. When set to 1, text prompts appear quickly until arriving at the selected step. When set to 0, the text prompts, narration, and simulation all start together at the selected step. **Parameters:** 0 (by default) or 1.
stop	Stops scenario playback.

Playing a Scenario from a Specific Step

The *playStep:* function, from the *ScenarioPlayer_Manager.udo,* controls the Scenario Player applet in the following way:

- Suspends the simulation graphics.
- Sets the internal clock of the simulation (the clock that sends timer ticks and other timing events) to run 50 times faster than normal.
- Starts the scenario, which triggers the event *scenarioStartEvent*.
- Plays the simulation actions in fast-forward speed while keeping the simulation graphics and voice-over suspended.

Upon reaching the selected step, the simulation clock is set back to normal, the graphics are enabled, and the voice-over begins to play.

Timing issues with scenario steps

Changing the simulation applet's internal clock to run 50 times faster causes the clock resolution to change: every one millisecond is now equivalent to 50 milliseconds in the normal

This means that time periods shorter than 50 milliseconds are less accurate. The effect is compounded by the fact that the resolution of the system clock is between 10 and 50 milliseconds, so that after multiplying by a factor of 50, the clock resolution is equivalent to 500–2500 milliseconds.

Different platforms, browsers, and connection rates can present difficulties with the fast simulation rate. Inconsistent timing issues can usually be addressed in the Scenario Authoring Tool. Refer to the manual *Using the Scenario Authoring Tool* for more information about scenario timing.

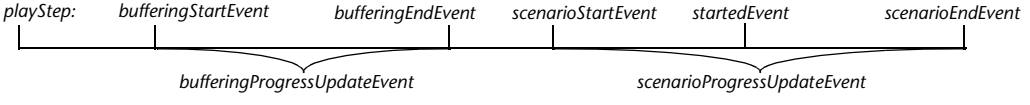
Events of ScenarioPlayer_Manager.udo

EVENT	DESCRIPTION
bufferingEndEvent	Triggered when the scenario voice-over buffering process ends.
bufferingProgress UpdateEvent	Triggered when the scenario voice-over buffering progress has been changed.
bufferingStartEvent	Triggered when the scenario voice-over buffering process begins.
pausedEvent	Triggered when scenario playback is paused.
resumedEvent	Triggered when scenario playback is resumed.
scenarioEndEvent	Triggered when the final scenario command is run.
scenarioProgressUpdate Event	Triggered when the scenario playback progress (a percentage of the total time) has been changes.
scenarioStartEvent	Triggered when the first scenario command is run.
startedEvent	Triggered when playback begins (anywhere in the scenario).
stoppedEvent	Triggered when playback stops (anywhere in the scenario).

Example

The schematic diagram below shows the sequence in which seven events are triggered when a scenario is started from an intermediate step with the *playStep:* function. Note that:

- The event *scenarioStartEvent* triggers actions such as restarting the simulation applet and, in this case, freezing the simulation image while simulation actions begin to fast forward.
- The event *startedEvent* triggers playback to begin at the selected step.



ScenarioPlayerUI_Agent.udo Reference

This user object manages the timing and content of (List) Prompts and Supplemental Content. List Prompts present the onscreen narration that guides simulation users through a scenario. They appear in the Scenario Player applet. Each List Prompt has text, a step number, and a sub-step number.

Supplemental Content segments can be used to enhance scenarios. In general, the supplemental content will be some external media, such as a Flash movie or an animated *.gif* file. A portion of the file will play at a specified point in the scenario.

The timing for playing the file is defined in the Scenario Authoring Tool. However, the Supplemental Content does not always require an external file. For example, the enhancement could be a change to the Scenario Player applet display that occurs for a specified amount of time.

The available API is described in the following tables.

Properties of ScenarioPlayerUI_Agent.udo

PROPERTY	DESCRIPTION
promptStep	Holds the step number of the prompt that is displayed when any one of the prompt events is triggered. **Parameter:** an integer; a step number greater than or equal to 0.
promptStyle	Holds the style name of the prompt that is displayed when any one of the prompt events is triggered. The prompt style must correspond to a style name exported by the Scenario Authoring Tool. A prompt style generally refers to a group of characteristics such as background color, border color, and font. **Parameter:** a string; a prompt style name.

PROPERTY	DESCRIPTION
promptSubStep	Holds the sub-step number of the prompt that is displayed when any one of the prompt events is triggered. **Parameter:** an integer; a sub-step number greater than or equal to 1.
promptText	Holds the text string of the prompt that is displayed when any one of the prompt events is triggered. **Parameter:** a string; the prompt text.
supplementalContentEnd	Holds the value at which to end the Supplemental Content segment, usually before it reaches the end of the file. The value is an integer that is meaningful to the content type, such as milliseconds, frame number, array index, etc. This property's value is only accurate when or after the *supplementalContentStartEvent* is triggered.
supplementalContentFile	Holds the name of the Supplemental Content file to be played when the *supplementalContentLoadEvent* is triggered. **Parameter:** a string; a file name. ***Example*** **ScenarioPlayerUI_Agent.supplementalContentFile := 'music.swf'**
supplementalContentStart	Holds the value to start the Supplemental Content segment, usually after the beginning of the file. The value is an integer that is meaningful to the content type, such as the time, frame number, array index, etc. This property's value is only accurate when or after the *supplementalContentStartEvent* is triggered.

PROPERTY	DESCRIPTION
supplementalContent Type	Holds the file type of the Supplemental Content file to be played when the *supplementalContentLoadEvent* is triggered. ***Example*** In this example, MyPlayer is the name of the Scenario Player application. *PlayFlashMovie:* and *PlayAnimatedGIF:* are user functions. **if ScenarioPlayerUI_Agent.supplementalContentType = 'FLASH'** **MyPlayer PlayFlashMovie: scenarioPlayerUI_agent.supplementalContentFile := 'logo.swf'** **else if scenarioPlayerUI_agent.supplementalContentType = 'GIF'** **MyPlayer PlayAnimatedGIF: scenarioPlayerUI_agent.supplementalContentFile := 'logo.gif'**

Events of ScenarioPlayerUI_Agent.udo

EVENT	DESCRIPTION
errorPromptEvent	Triggered when an error occurs. The *promptText* property contains the error message to be displayed.
firstPromptEvent	Triggered when the first prompt should be displayed. The *promptText* property contains the text that will be displayed. ❖ *NOTE: The step number for the first prompt is always 0. The first prompt is often used as an introduction to the scenario.*
lastPromptEvent	Triggered when the last prompt in the scenario should be displayed. The *promptText* property contains the text that will be displayed.
normalPromptEvent	Triggered when a prompt should be displayed. The *promptText* property contains the text that will be displayed. This event does not apply to the first and last prompts.
supplementalContent EndEvent	Triggered when a Supplemental Content segment is ended.
supplementalContent LoadEvent	Triggered when the Supplemental Content file is loaded and ready to play.
supplementalContent StartEvent	Triggered when a Supplemental Content segment is started.

AppletContext_Manager.udo Reference

This user object implements communication between the Scenario Player applet and the Web browser/applet viewer through JavaScript. With these functions, events, and properties, information that is entered through the *.html* files can be read dynamically by the Scenario Player applet. The original *.html* files used for code generation are located in the \\Rapidx\Java\HTML folder.

The available API is described in the following tables.

Events of AppletContext_Manager.udo

EVENT	DESCRIPTION
externalCall	Triggered when an external JavaScript function calls the simulation applet. (An example can be found in the `callPlayer` function of the *appletutilslf2.js* file.) The JavaScript call should be formatted as: `rapid_applet.javascriptNotify ('Player', functionName, params);` where `rapid_applet` is the name of the simulation applet in JavaScript and `functionName` and `params` are two string arguments to be passed to the Scenario Player applet.

Properties of AppletContext_Manager.udo

These properties use information received from the user object's *externalCall* event.

PROPERTY	DESCRIPTION
externalCallFunction	Holds the value of the "functionName" parameter.
externalCallParams	Holds the value of the "params" parameter.

Functions of AppletContext_Manager.udo

FUNCTION	DESCRIPTION
callJavaScriptFunction: '<String>' with: <String_Array>	Calls the specified JavaScript function with the specified parameters. **Parameters:** <**String**> the name of a JavaScript function. <**String_Array**> an array of strings that contains parameters for the JavaScript function. ***Example*** **AppletContext_Manager callJavaScriptFunction: 'showMessage' with: MessageInfo_Array** where **MessageInfo_Array** is a string array with two strings: the message title and message text.

getParameter: '<String>'	Returns the `value` attribute of a `PARAM` element (which is in the content of an HTML `APPLET` element).
	Parameter: a string; the `name` attribute in a `PARAM` element.
	Return value: a string; the `value` attribute.
	Example
	AppletContext_Manager getParameter 'bkgrdColor'
	returns the value "00FF00" (shown in bold below) where the markup appears similar to the following:
	`<APPLET name="player_MyPlayer" ... >`
	`  <PARAM name="CABBASE"`
	`  value="MyPlayer_player.cab">`
	`  <PARAM name="bkgrdColor"`
	`  value="00FF00"> ...`
	`</APPLET>`
	❖ *NOTE: You can add PARAM elements to the APPLET element in the .html files, and then incorporate their attribute values into the design of the Scenario Player applet. This enables control over specified characteristics outside the Scenario Player applet, so that changes can be made for a Web site or for a single Scenario Player applet.*

FUNCTION	DESCRIPTION
showDocument: '<String>' in: '<String>'	Used to display the specified document in the specified target window or frame. **Parameters:** First string: a URL. Second string: a target window or frame. The target should be one of the following strings: • _self: show the document in the window and frame that contains the Scenario Player applet. • _parent: show the document in the applet's parent frame. If the applet's frame has no parent frame, acts like "_self." • _top: show the document in the top-level frame of the applet's window. If the applet's frame is the top-level frame, acts like "_self." • _blank: show the document in a new, unnamed top-level window. • name: show the document in a frame or window named "name." If a target named "name" does not already exist, a new top-level window with the specified name is created and the document is shown there. ***Example*** **AppletContext_Manager showDocument: 'www.e-sim.com' in: '_parent'**
writeInStatusBar: '<String>'	Writes the specified string in the status bar of the Web browser/applet viewer. **Parameter:** a string. ***Example*** **AppletContext_Manager writeInStatusBar: 'Loading...'**

Simulation_Agent.udo Reference

This user object implements interface for the Scenario Player applet to obtain information from the simulation applet, and to send actions to the simulation applet. The Scenario Player applet can exchange information about simulation actions with the simulation applet. (In the Scenario Authoring Tool these actions are referred to as "sim-events.") This user object is particularly useful for computer-based training (CBT).

The available API is described in the following tables.

Properties of Simulation_Agent.udo

The properties' values are obtained from the scenario *.zip* files.

PROPERTY	DESCRIPTION
actionObject	Holds the name of the simulation object on which a user action was performed.
actionType	Holds the type of action that was performed. **Parameters:** a string that uses one of the following values: 'action': indicates that the simulation object's state was changed, e.g., *mousePressedEvent*. 'object': indicates that the simulation object's state was not changed, e.g., *mouseEnteredEvent*.
actionValue	Holds a string containing information about the user action, such as mouse coordinates or button pressed or released.

Functions of Simulation_Agent.udo

FUNCTION	DESCRIPTION
perform: '<String>'	Performs the specified event/command on the simulation applet. **Parameters:** a string; a command (as from a scenario script). **Return values:** target and value. ***Example*** **Simulation_Agent perform: '<SIMEVENT target='obj_onOff_pb' value='Pressed' />'**
setMonitorMode	Sets action routing to normal mode in which simulation-user actions are sent from the simulation applet to the Scenario Player applet and the actions are performed. This mode is the default mode.
setRerouteMode	Sets action routing to reroute mode in which events are sent from the simulation applet to the Scenario Player applet but are not performed.

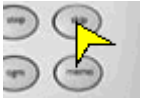
Events of Simulation_Agent.udo

EVENT	DESCRIPTION
actionOutEvent	Triggered each time a user action is performed on the simulation applet.

WORKING WITH INDICATORS AND DEMONSTRATORS

Indicators and demonstrators are visual aids that focus a scenario user's attention on objects and actions in the simulation applet. The appearances of these visual aids are defined in the *scenariotemplate.txt* file, an XML-content file that is read by the Scenario Player applet while a scenario runs.

Indicators frame an area of interest during a scenario. In the Scenario Authoring Tool, the scenario author defines whether the frame zooms in on the object, blinks, or remains steady.



Demonstrators bring attention to actions taking place during a scenario. In the Scenario Authoring Tool, the scenario author uses a single demonstrator to highlight a single action, or links several demonstrators together to trace a path of actions.

You can modify the *scenariotemplate.txt* file so that the appearance of indicators and demonstrators will suit the style of your product and Web site.

❖ *NOTE: The scenariotemplate.txt file may contain elements unrelated to indicators and demonstrators for the sake of backwards compatibility.*

Making Global or Local Changes

You can change the appearance of the default indicators and demonstrators by modifying *scenariotemplate.txt* and its referenced image files. Global changes ensure a uniform look among the indicators and demonstrators of an entire group of Scenario Player applets; a local change provides a distinctive look for a specific Scenario Player applet.

Making Global Changes Before Code Generation

A *scenariotemplate.txt* file and referenced image files are located in the \\Rapidx\Java\HTML folder. These files determine the appearance of indicators and demonstrators for every generated scenario authoring package. To change the look globally, modify this *scenariotemplate.txt* file and these images. The changes will apply to all newly generated scenario authoring packages.

❖ *NOTE: Save a backup copy of the \\Rapidx\Java\HTML folder before changing any part of it.*

Making Global Changes for a Web Site

You can direct many Scenario Player applets to read from the same *scenariotemplate.txt* file if it is located in a common folder on the Web server. (See "The Rapid Application Library" on p. 123 for an example on how to direct the Scenario Player applet to a common URL.) In this case, a single *scenariotemplate.txt* file can be easily updated for current use.

❖ *NOTE: If you use a common URL,* ***remove*** *scenariotemplate.txt and its referenced image files from the SitePack folder.*

Example

A new marketing strategy changes the color scheme on your company's Web site. You can easily coordinate the colors for indicators and demonstrators in the single *scenariotemplate.txt* file and its referenced image files.

Making Local Changes

You can set the appearance of indicators and demonstrators for a single scenario authoring package. Modify the *scenariotemplate.txt* file (and its referenced image files) in the \\ScenPack\HTML folder. After the scenario author links the scenarios to the simulation and Scenario Player applets (see "Distributing Simulations to Scenario Authors" on p. 69), the applied changes appear when the scenarios are run from the SitePack folder.

The Scenario Player applet first looks for *scenariotemplate.txt* in the SitePack folder. If the file is not found in this local folder, the Scenario Player applet looks for *scenariotemplate.txt* in the designated common folder.

Example

A line of cellular telephones uses a common folder for the *scenariotemplate.txt* file and referenced images, but a new product in the line is geared toward younger clients. The *scenariotemplate.txt* file and image files can be modified to adapt to this specific product's style.

Modifying Indicators

Currently, there is one kind of indicator defined in the *scenariotemplate.txt* file. Indicators are defined under the `Indicators` element, using the `ScenarioObject` element. The `ScenarioObject` element defines the fill and border colors, corner length, and blink timing of the indicators. The default attributes are as follows:

```
<Indicators>
   <ScenarioObject type='highlight' className='RectangleIndicator'
    MaxFrameLength='18' CornerLength='10' FillColor='#FFFF00'
    BorderColor='#000000' Blinking='on' BlinkingInterval='50'
    BlinkingTime='9000'/>
</Indicators>
```

To modify the appearance of indicators:

1 In a text editor, open the appropriate *scenariotemplate.txt* file (see "Making Global or Local Changes" on p. 95).

2 Locate the attribute you want to modify and change its value as needed. The attributes are described in the following table.

   ❖ *NOTE: **XML is case-sensitive**, so enter and reference values precisely. Also, make sure that there are no spaces between the attribute value and the quotation marks around it.*

3 Save and close the file.

You can edit most of the default values for the `ScenarioObject` element, but you cannot create a new one. The following table describes the attributes.

ATTRIBUTE	DESCRIPTION
type	Identifies the indicator. Corresponds to the indicator Type property in the Scenario Authoring Tool. **Do not change.**
className	The type of indicator. **Do not change.**
CornerLength	The length, in pixels, of each corner.
FillColor	The color of the area enclosed by the indicator's border. Color values can be defined as follows: • A six-digit hexadecimal code, prefaced by a number sign (for example, use #FFFFFF for white). • One of the color names defined in Java code: black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow.
BorderColor	The color of the indicator's outline. Color values are defined the same way as for `FillColor`.

ATTRIBUTE	DESCRIPTION
Blinking	Determines whether or not the indicator blinks or remains in a steady state. Can be set to on or off.
BlinkingInterval	Determines the rate at which the indicator blinks or zooms. The the blink or zoom property must be selected for the indicator in the Scenario Authoring Tool.
BlinkingTime	The time, in milliseconds, for which an indicator will blink before changing to steady state.

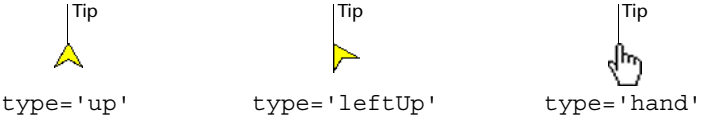
Modifying Demonstrators

There are nine demonstrators defined in the *scenariotemplate.txt* file. The `Demonstrator` elements have attributes that define the image files, and they have x,y coordinates on the image for the pointer's tip—the part of the image that enters the simulation object's hotspot or active area. The default attributes are as follows:

```
<Demonstrators>
   <Demonstrator type='up' image='images/demo_n.gif' tip_x='0'
    tip_y='2'/>
   <Demonstrator type='down' image='images/demo_s.gif' tip_x='14'
    tip_y='15'/>
   <Demonstrator type='left' image='images/demo_w.gif' tip_x='0'
    tip_y='14'/>
   <Demonstrator type='right' image='images/demo_e.gif' tip_x='15'
    tip_y='3'/>
   <Demonstrator type='upRight' image='images/demo_ne.gif'
    tip_x='19' tip_y='0'/>
   <Demonstrator type='rightDown' image='images/demo_se.gif'
    tip_x='20' tip_y='20'/>
   <Demonstrator type='downLeft' image='images/demo_sw.gif'
    tip_x='0' tip_y='20'/>
   <Demonstrator type='leftUp' image='images/demo_nw.gif' tip_x='0'
    tip_y='0'/>
   <Demonstrator type='hand' image='images/hand.gif' tip_x='8'
    tip_y='1'/>
</Demonstrators>
```

Example

In the following demonstrators, the tip coordinates are (8, 0), (0, 0), and (8, 1), respectively:



To modify the appearance of demonstrators:

1 In a text editor, open the appropriate *scenariotemplate.txt* file (see "Making Global or Local Changes" on p. 95).

2 Locate each attribute you want to modify and change its value as needed. The attributes are described in the following table.

   ❖ *NOTE: **XML is case-sensitive**, so enter and reference values precisely. Also, make sure there are no spaces between the attribute value and the quotation marks around it.*

3 Save and close the file.

You can edit an existing `Demonstrator` element, but you cannot create a new one. The following table describes the attributes.

ATTRIBUTE	DESCRIPTION
type	Name used to specify the demonstrator. Corresponds to the Type property of demonstrators in the Scenario Authoring Tool. The value reflects the direction in which the demonstrator is pointing. Do not change.
image	Relative path and file name of the image used for the demonstrator.
tip_x	The x coordinate on the image, in pixels, where the image enters the object's hotspot.
tip_y	The y coordinate on the image, in pixels, where the image enters the object's hotspot.

Example

You can change the Up arrow to a company logo. First place the file *mylogo.gif* in the images folder. Next, find the code for the Up arrow:

Original Demonstrator element

```
<Demonstrators>
  <Demonstrator type='up' image='images/demo_n.gif' tip_x='11'
   tip_y='0'/>
</Demonstrators>
```

Change the value of the `image` attribute and update the values for `tip_x` and `tip_y`, as shown in bold:

```
<Demonstrator type='up' image='images/mylogo.gif' tip_x='8'
 tip_y='4'/>
```

Now every time a demonstrator of **Type** Up is used in the Scenario Authoring Tool, the new demonstrator image appears in the scenario.

APPENDIX A

Code Generation Output

This appendix presents information about:

- Parameters that are updated automatically during the code generation and make processes.
- Files in the root folder of the code generation output folder.
- Generated subfolders and their contents.
- Generated *.html* files and the way they fit together.

UPDATED SIMULATION AND SCENARIO PARAMETERS

During the make process, *.html* and *.js* template files from the \\Rapidx\Java\HTML folder are copied to the SitePack and \\ScenPack\SitePack output folders and updated.

The following parameters are replaced and/or updated during Java code generation:

PARAMETER	DESCRIPTION
<APP_CLASS_NAME>	The simulation class name. This is generally the same as the application name, but uses the Java convention of all lowercase letters for package names.
<APP_HEIGHT>	The simulation applet height (same as the original application height).
<APP_NAME>	The application name.
<APP_WIDTH>	The simulation applet width (same as the original application width).
<PACK_NAME>	The simulation package name. This is generally the same as the application name, but uses the Java convention of all lowercase letters for package names.
<SCEN_FILEx>	The scenario file names exported from the Scenario Authoring Tool. For example, *MyScen1.txt, MyScen2.txt, MyScen3.txt,* etc.
<SCEN_NAMEx>	The scenario title, as entered through the Scenario Authoring Tool (for example, “Setting the Clock”). ❖ *NOTE: If there are fewer scenarios than scenario parameters, the excess parameters are deleted. If there are more scenarios than parameters, you can add parameters to the .html files.*

CODE GENERATION OUTPUT FILES IN THE ROOT FOLDER

The following files are in the root folder of the code generation output folder:

FILE	DESCRIPTION
<MyApp>.cgr	This file is for internal program use.
<MyApp>.ids	This file contains the name of each object in the simulation and an identification number for each object. These IDs are used by the Scenario Authoring Tool to indicate objects in the simulation.
<MyApp>.jlp	This file contains user-defined simulation parameters such as application name, height, and width. It is used by Rapid to pass information to the make process.
<MyApp>_Image-Packager.txt	This file contains the image download category information for every image, and the color palette entries for *.gif* files. For more information on image download categories, see "Optimizing Image Download" on p. 57.
make.bat	This file runs the make process as described in "Making Simulations Without Regenerating Code" on p. 62.
Utility batch files	These files are for internal program use.

GENERATED SUBFOLDERS

The code generation output folder contains subfolders and files to run simulations and build scenarios. The folders that are created depend on your settings in *make.config*. Some folders are temporary and are automatically removed if the *cleanUp* option in the *make.config* file was set to yes (default). (See "Setting Configuration Options" on pp. 44-51 for details.)

The generated subfolders are described in this section.

<MyApp> Folder

The <MyApp> folder contains the generated Java files. These files are used for the make process.

Build Folder

The Build folder contains the configuration file for the code optimization tool, DashO-Pro, and the report files generated by DashO-Pro. The Out subfolder contains the optimized classes generated by DashO-Pro.

This folder is removed automatically when *cleanUp* is set to **yes.**

CdPack Folder

The CdPack folder contains all the files necessary to install Front End Processor (FEP) support for Web browsers on the simulation user's computer:

- The *FepInstall.exe* file to install FEP support for Web browsers on the simulation user's computer. If the *make.config* file contains the command `installFep=yes`, the *FepInstall.exe* file runs automatically on the Rapid application designer's computer immediately after code generation.
- The *FepUninstall.exe* file to uninstall FEP support for Web browsers.
- Other files and subfolders required by the simulation applet.

After FEP installation, users can run Java simulations that use FEP from the SitePack folder (p. 105). Java simulations that use FEP can only run from a CD-ROM or a local hard disk.

Classes Folder

The Classes folder contains the compiled generated code in subfolders that correspond to package names.

This folder is removed automatically when *cleanUp* is set to **yes.**

Localres Folder

The Localres folder contains generated application resources to be included in the *.jar* or *.cab* files.

This folder is removed automatically when *cleanUp* is set to **yes.**

ScenPack Folder

The ScenPack folder contains the entire work environment for use with the Scenario Authoring Tool, including utility files to run the link process and **copies** of the \\Rapidx\Java\HTML folder, the simulation folder, and the SitePack folder (see below).

SitePack Folder

The SitePack folder contains only the files needed to run simulations and scenarios from a Web server.

❖ ***CAUTION: Do not modify the contents of SitePack.*** *The contents are overwritten every time the make process is run. To make changes, modify source files and then run the make process again.*

The SitePack folder includes the following subfolders and files:

Subfolders

The subfolders in the SitePack folder include:

- The <myapp> folder, containing the necessary files to run the simulation applet.
- The control_icons folder, containing the images for the Scenario Player applet's JavaScript interface (see Appendix C: "JavaScript User Interface for Scenarios"). This folder can be deleted if the JavaScript interface is not used.
- The images folder, containing the image files for the Scenario Player applet.
- The resources folder, containing the *.wav* files and *.txt* files that are called by the simulation applet.

Files

The files in the SitePack folder include:

- The *.html* files to view the simulation applet (and Scenario Player applet) in a Web browser.

  The applets are run from one of three files: *demo_player.html, demo_playerviewnew.html,* and *demo_playerviewold.html*. You can also run the simulation applet (without the scenarios) from *demo_sim.html*.

The other *.html* files are used with the HTML `FRAMESET` elements for *demo_playerviewnew.html* and *demo_playerviewold.html,* as illustrated on p. 106.

These files are examples. The Web designer should modify them as necessary for the Web site.

- The *.js* files, which hold functions and parameters to display and run the simulation applet and the Scenario Player applet, and hold the version number for the Scenario Player applet and its user interface.
- The *scenariotemplate.txt* file to modify the appearance of the Scenario Player applet (see Chapter 5: "Developing the Scenario Player Applet").
- Image files, *grad.gif* and *wait.gif,* used for displaying the *.html* pages.
- Scenario *.txt, .zip,* and *.wav* files with references (links to start them) in the relevant *.html* files.
- Simulation *.jar* and *.cab* archive files.

StandAlone Folder

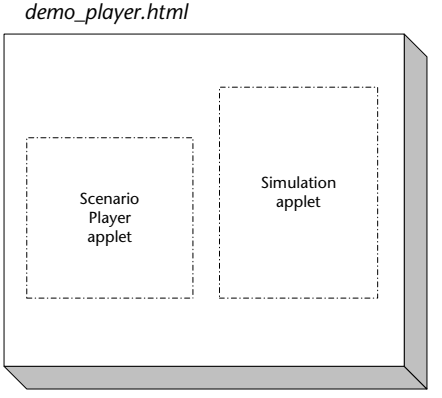
The StandAlone folder contains the files necessary to run a stand-alone simulation for the stand-alone Prototyper. This type of simulation is for testing during application development and can only be viewed on a computer with the full RapidPLUS Web Studio installation.

HTML FILES

The automatically generated *.html* files are examples—the Web designer should modify them as necessary for the Web site. The main files used to run the simulation with scenarios are *demo_player.html, demo_playerviewnew.html,* and *demo_playerviewold.html*. The diagrams in this section illustrate how these files arrange applets and *.html* files in Web browsers.

The *demo_player.html* File

In the *demo_player.html* file, the Scenario Player applet and simulation applet are arranged in an HTML `TABLE` element:

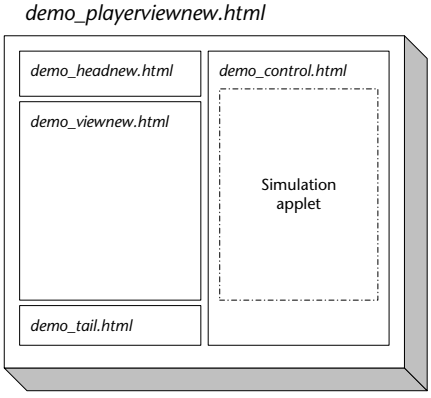


A graphic appears above and below the Scenario Player applet. The Web designer can edit the graphics as necessary in the *.html* file, and can adjust the *scenariotemplate.txt* file (see Chapter 5: "Developing the Scenario Player Applet").

The *demo_playerviewnew.html* File

The *demo_playerviewnew.html* file is intended for scenarios that run with the JavaScript user interface (see Appendix C: "JavaScript User Interface for Scenarios"), therefore, the Scenario Player applet is launched, but hidden.

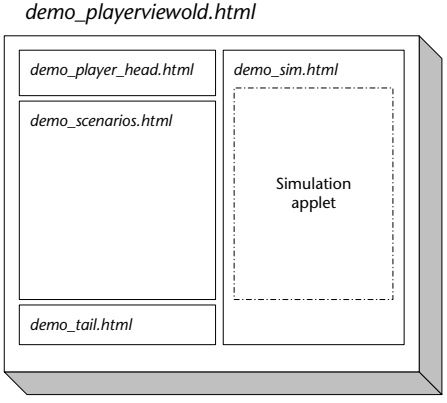
The file calls other *.html* files through HTML `FRAMESET` elements, and arranges them as follows:



❖ *NOTE: This file is for demonstration purposes; it is not a supported Scenario Player applet interface.*

The *demo_playerviewold.html* File

The *demo_playerviewold.html* file is intended for scenarios that run with the JavaScript user interface and only use simulation prompts (see Appendix C: "JavaScript User Interface for Scenarios"). The Scenario Player applet is launched, but hidden. The file calls other *.html* files through HTML `FRAMESET` elements, and arranges them as follows:



APPENDIX B

Errors, Warnings and Messages

As code is generated, a running log of the code generation process is displayed in the Code Generation Status window. The log includes notification regarding code generation problems (or potential problems) encountered. These notices are classified as follows:

- Errors (E): If the code generation encounters one of these conditions, the generated code might not compile.
- Warnings (W): These conditions may or may not result in improper application performance and they should be investigated before compiling the code and linking scenarios.
- Informational messages (I): These notices are primarily informational, although some of them may indicate application bugs that should be corrected.

This appendix presents errors, warnings, and informational messages that may appear when generating code.

ERRORS (E)

ERROR MESSAGE	REMARKS
You may not pass a property by reference in Java	The Rapid application contains a call to a user-defined function, whose parameter(s) must be an assignable data object. Change the Rapid application to make it suitable for Java code generation.
Corrupted application	The Rapid application is corrupted. Re-verify the logic before code generation.
Error recognizing the line: <Rapid source code>	The Rapid application contains an invalid logic line. Re-verify the logic before code generation.
Problem generating object: <object name>	Implementation error. Report the error to the Rapid technical support.
Object Array: <array name> has a non-generated object	The Rapid application contains an object array with an initial object, that is not supported in the Java simulation. Change the initial object in the array before code generation.
User Object name: <user object name> is the main application name. This will cause conflicts while trying to compile this file.	The application source code files have overwritten the user object's source code files. The application will not compile. Save the user object under a different name, then use it to replace the original user object.
File: <file name> is Read Only	A previously generated file has been re-defined as read-only, so the Code Generator cannot overwrite it. Change the file attribute before code generation.

ERROR MESSAGE	REMARKS
Font <font name> is not available	The Rapid application contains a font that is not available in the current Windows font set. Change the font in the Rapid application, or install the missing font.
Linked file <file name> is not found.	Bitmap or image object cannot be generated because its linked file is not found. Verify the location of the linked file.
<Application name> is the name of a Java reserved word. This will cause conflicts while trying to compile this file.	Save the application under another name before code generation.
There must be at least one local compound mode	The *UDC_<MyApp>.pkgr* file should contain at least one index number LOAD_LOCAL mode between "BEGIN COMPOUND" and "END COMPOUND" statements, and at least one mode belonging to this loading category between "BEGIN MODES" and "END MODES." Fix the *.pkgr* file, or delete it to use the default settings before code generation.
A transition without destination mode has been found	This condition is usually the result of pasting modes between applications. Re-verify logic before code generation.

WARNINGS (W)

WARNING MESSAGE	REMARKS
Else without If statement	The Rapid application contains an Else without an If statement. Fix the Rapid application logic before code generation.
For statement is empty	The Rapid application contains an empty For statement. Fix the Rapid application logic before regenerating code.
Application contains blank lines of logic	The Rapid application contains a blank line of logic. Delete the line from the Rapid application before code generation.
Subroutine: <function name> contains an invalid parameter	The Rapid application contains a user-defined function with a parameter type that is not supported by the Java Code Generator. As a result, the generated application behaves differently from the Rapid application. Fix the user-defined function in the Rapid application before code generation.
Function:<function name> is not supported *or* *Object: <object name> is not supported* *or* *Property: <property name> is not supported*	The Rapid application contains an object, object property, or object function that is not supported by the Java Code Generator. As a result, the generated application behaves differently from the Rapid application. Remove the unsupported item from the Rapid application before code generation.

WARNING MESSAGE	REMARKS
Mode <transition source mode name> contains a mouse-based transition.	The Rapid application contains a transition that is based on a mouse object event or property. Since the mouse object is implemented differently in Java code, the generated application may behave differently from the Rapid application. Change the Rapid application so as not to use a mouse-based transition.
Removing unnecessary user function <function name>	A function is not generated if: • It is both unexported **and** unreferenced by the application or user object. • It contains a parameter that cannot be generated (such as an unsupported object).
Mouse cursor: <mouse cursor shape name> is generated with default shape.	The Rapid application contains cursor shapes that are not supported by the Java Code Generator. These shapes are replaced with the default arrow shape in the generated application. To avoid arbitrary replacement, use cursor shapes that are compatible with Java technology in the Rapid application.
Function: <function name> is not necessary and therefore, is not generated	Rapid application functions that are not required in Java code (for example, “Mouse activate”) are omitted from the generated code without affecting the behavior of the Java application.

INFORMATIONAL MESSAGES (I)

MESSAGE	REMARKS
Removing the unnecessary <entry/exit/mode/transition> activity from mode <mode name>	A logic line which was commented out in the Logic Editor is not being generated.
Issued the command: <command>	The command specified in the Code Generation Preferences dialog box has been executed.
<fileName.java> will be backed up as <fileName.java$$>	When generating code in a folder that already contains output files of the same name, the existing output files are backed up by adding two dollar signs to the file extension.

APPENDIX C

JavaScript User Interface for Scenarios

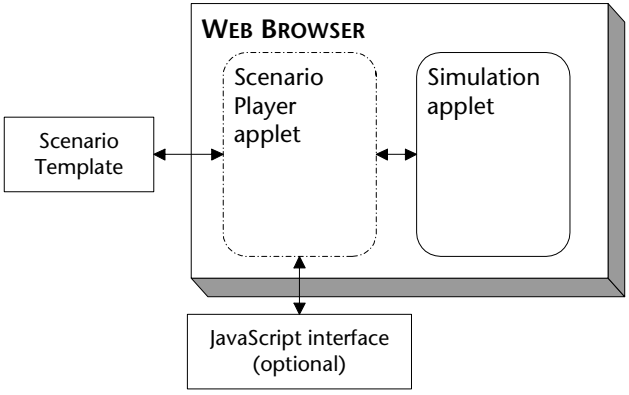
The Scenario Player applet controls the scenario timing, displays indicators and demonstrators, and provides a user interface for displaying scenario links and list prompts. This is the default method for running scenarios. However, in some circumstances you may prefer to hide the Scenario Player applet and use a JavaScript user interface instead.

The JavaScript user interface enables control over the appearance of scenarios in the Web browser—more control than the *scenariotemplate.txt* file provides (see Chapter 5: "Developing the Scenario Player Applet"). The hidden Scenario Player applet still manages the synchronization of the scenario, and it still uses the *scenariotemplate.txt* file to display indicators and demonstrators. However, control of the scenario playback (e.g., playing and stopping scenarios), the appearance of the list of scenario links, and the appearance of list prompts are now governed by JavaScript.

It is important to note that **there is not universal support for JavaScript-Java communication.** The following limitations are:

- **Netscape 4.6:** Java-to-JavaScript communication is not supported, but JavaScript-to-Java communication is supported.
- **Netscape 6:** Java-to-JavaScript communication is not supported, but JavaScript-to-Java communication is supported.
- **Internet Explorer for Macintosh with Apple Java:** Neither Java-to-JavaScript communication, nor JavaScript-to-Java communication are supported.

The following illustration shows the flow of information:



All the files necessary for the JavaScript user interface are generated automatically with the Scenario Player applet to the SitePack subfolder (and \\ScenPack\SitePack folder) of the code generation output folder.

The *demo_playerviewnew.html* file and the *demo_playerviewold.html* file are automatically generated with the Scenario Player applet. In these files, the Scenario Player applet is hidden and the JavaScript user interface is used instead.

Before making changes to the JavaScript user interface, review the following sections. They provide an explanation of the available functions of the Scenario Player applet and the simulation applet, as well as a description of the basic functions in the JavaScript utilities library, *appletutilslf2.js*.

SCENARIO PLAYER APPLET FUNCTIONS

The following Scenario Player applet functions are accessible to JavaScript. Although you can call the functions directly, we recommend that you call them through the functions provided in the JavaScript utilities library (see "The JavaScript Utilities Library" on p. 119).

playScript

Loads a scenario script and starts playing it (same as calling the `loadScenario` and `playClicked` functions). Used when the scenario only has simulation prompts.

Parameter

A string containing the scenario file name.

Syntax

```
player_applet.playScript("myscenario1.txt");
```

Called From

Called from `PlayScenario(appName, caseName)` in *appletutilslf2.js*.

loadScenario

Loads a scenario script without playing it.

Parameter

String containing the scenario file name.

Syntax

```
player_appletp.loadScenario("myscenario1.txt");
```

Called From

Called from `LoadScenario(appName,caseName)` in *demo_control.html*.

playClicked

Starts or resumes playback of a scenario which has already been loaded.

Parameter

None.

Syntax

```
player_applet.playClicked();
```

Called From

Called from `PlayClicked(appName)` in *appletutilslf2.js*.

stopClicked

Stops a playing scenario and resets the simulation to its initial state.

Parameter

None.

Syntax

```
player_applet.stopClicked();
```

Called From

Called from `StopClicked(appName)` in *appletutilslf2.js*.

pauseClicked

Pauses the scenario.

Parameter

None.

Syntax

player_applet.pauseClicked();

Called From

Called from `PauseClicked(appname)` in *appletutilslf2.js*.

THE JAVASCRIPT UTILITIES LIBRARY

A JavaScript utilities library, *appletutilslf2.js,* is copied to the \\ScenPack\HTML folder during code generation. Some of the functions defined in this file, such as `rapid_applet()`, are called from the *.html* pages. The remaining functions are used by the JavaScript files.

The functions described in this section provide a high-level interface between the simulation and the Scenario Player applet.

❖ *NOTE: Some of these functions have the same basic functionality (and even names) as the functions in the Scenario Player and simulation applets. We recommend that you call these functions through the JavaScript library, since the JavaScript functions perform additional checks to prevent timing problems.*

rapid_applet

Dynamically inserts the `APPLET` tags for the simulation applet into the referring *.html* file. The values for the parameters are supplied by *rapidapp.js*.

Syntax

```
rapid_applet(appName, codebase, width, height, backcolor);
```

Parameters

appName	Name of your simulation applet.
codebase	Search path for Java classes.
width	Width of the applet.
height	Height of the applet.
backcolor	Background color of the simulation applet. The value must be a hexadecimal number in RRGGBB format. For example, white is FFFFFF, black is 000000, and red is FF0000.

Called From

Called from *demo_player.html, demo_control.html,* and *demo_sim.html*.

Example

To display a simulation applet, the following script is added to the *.html* files:

```
rapid_applet(rapid_AppName, ".", rapid_Width, rapid_Height,
 rapid_Background
   );
```

player_applet

Dynamically inserts the `APPLET` tags for the Scenario Player applet into the referring *.html* file. The value for the parameters is supplied by *rapidapp.js*.

Syntax

```
player_applet(appName, codebase, width, height, backcolor, visible,
 commonFiles)
```

Parameters

appName	Name of your simulation applet.
codebase	Search path for Java classes.
width	Width of the applet.
height	Height of the applet.
backcolor	Background color of the applet. The value must be a hexadecimal number in RRGGBB format. For example, white is FFFFFF, black is 000000, and red is FF0000.
visible	True if Scenario Player applet is visible.
commonFiles	Relative or absolute path to the common files directory, which contains the *scenariotemplate.txt* file and related image files.

Called From

Called from *demo_player.html, demo_control.html,* and *demo_sim.html*.

Example

To display a Scenario Player applet, with the *scenariotemplate.txt* file stored in the images folder, the following script is added to the *.html* files:

```
player_applet(rapid_AppName, ".", rapid_PlayerWidth,
rapid_PlayerHeight, rapid_PlayerBackground,
   true,                 // Visible
   "images"     // Common files
   );
```

PlayScenario

Verifies that the simulation has loaded, then loads the scenario and starts playback. Calls the Scenario Player applet function `playScript`.

Syntax

```
PlayScenario(<appName>, <caseName>)
```

Parameters

appName	The name of your simulation applet. This must be the same as the <appName> parameter in the `rapid_applet` function.
caseName	The name of the scenario *.txt* file, such as "*scenario1.txt*".

Called From

Called from *demo_scenarios.html*.

Example

```
<A HREF="javascript:PlayScenario('myApp', 'scenario1.txt')">Click
 here to view the first scenario.</A>
```

StopClicked

Stops playing a scenario. Calls the Scenario Player applet function `stopClicked`.

Syntax

```
StopClicked(<appName>);
```

Parameters

appName	The name of your simulation applet. This must be the same as the <appName> parameter in the `rapid_applet` function.

Called From

Called from *demo_scenarios.html* and *demo_control.html*.

Example

```
<A HREF="javascript:StopClicked('myApp')">Stop</A>
```

PlayClicked

Starts or resumes scenario playback. Calls the Scenario Player applet function `playClicked`.

Syntax

```
PlayClicked(<appName>);
```

Parameters

appName	The name of your simulation applet. This must be the same as the <appName> parameter in the `rapid_applet` function.

Called From

Called from *demo_scenarios.html* and *demo_control.html*.

Example

```
<A HREF="javascript:PlayClicked('myApp')">Play</A>
```

PauseClicked

Pauses a scenario. Calls the Scenario Player applet function `stopScript`.

Syntax

```
PauseScenario(<appName>);
```

Parameters

appName	The name of your simulation applet. This must be the same as the <appName> parameter in the `rapid_applet` function.

Called From

Called from the *demo_scenarios.html*.

Example

```
<A HREF="javascript:PauseClicked('myApp')">Pause</A>
```

THE RAPID APPLICATION LIBRARY

The file *rapidapp.js* stores information about the simulation and player applets such as name and size. These data are referenced by JavaScript functions in other libraries. Their definitions are supplied during code generation.

PARAMETER (WITH EXAMPLE)	DESCRIPTION
rapid_AppName = "myapp";	Name of the simulation applet.
rapid_Width = "200";	Width of the simulation.
rapid_Height = "300";	Height of the simulation.
rapid_Background = "FFFFFF";	Background color of the simulation applet.
rapid_PlayerWidth = "340";	Width of the Scenario Player applet.
rapid_PlayerHeight = "365";	Height of the Scenario Player applet.

PARAMETER (WITH EXAMPLE)	DESCRIPTION
rapid_PlayerBackground = "FFFFFF";	Background color of the Scenario Player applet.
rapid_PlayerCommonFiles = "http://www.example.com/common_files";	Common URL for the *scenariotemplate.txt* file and Scenario Applet images.
rapid_minFlashRate = 10000	Minimum acceptable download rate (in bytes/second) for Flash audio. If the download rate is less than this value, *.wav* files will be used, even if Flash audio files are available.

Index

T