General Magic

SDK Quick Start Guide

for Magic Cap 3.1 (Rosemary)

June 19, 1997

License

Your use of the software discussed in this document shall be permitted only pursuant to the terms in a software license between you and General Magic.

Trademarks

The General Magic logo, the Magic Cap logo, the Telescript logo, Magic Cap, Telescript, and the A rabbit-from-a-hat logo are trademarks of General Magic, and may be registered in certain jurisdictions. All other trademarks and service marks are the property of their respective owners.

Limit of Liability/Disclaimer of Warranty

THIS DOCUMENT IS SOLD "AS IS." Even though General Magic has reviewed this document in detail, GENERAL MAGIC MAKES NO REPRESENTATION OR WARRANTIES, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THE ACCURACY OR COMPLETENESS OF THE CONTENTS OF THIS DOCUMENT. GENERAL MAGIC SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OR MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE AND SHALL IN NO EVENT BE LIABLE FOR ANY LOSS OF PROFIT OR ANY OTHER COMMERCIAL DAMAGE, INCLUDING BUT NOT LIMITED TO SPECIAL, INCIDENTAL, CONSEQUENTIAL, OR OTHER DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Some states do not allow for the exclusion or limitation of implied warranties or incidental or consequential damage. So, the exclusions in this paragraph might not apply to you. This warranty gives you specific legal rights. You may also have other rights which vary from state to state.

It's 4:30 a.m. Do you know where your children are?

Restricted Rights. For defense agencies: Use, duplication, or disclosure is subject to the restrictions set forth in subparagraph (c)(1)(ii) of DFAR section 252.227-7013 and its successors. For civilian agencies: Use, duplication, or disclosure is subject to the restrictions set forth in subparagraphs (a) through (d) of FAR section 52.227-19 and its successors. Unpublished—rights reserved under the copyright laws of the United States.

Patent Pending

Portions of the Magic Cap software and the Telescript software are patent pending in the United States and other countries.

General Magic

420 N. Mary Ave.
Sunnyvale, California 94086
USA

Telephone:	408 774 4000
Fax:	408 774 4030
E-mail:	dev-info@genmagic.com
URL:	http://www.genmagic.com/

Table of Contents

SDK Quick Start Guide

This guide introduces you to Magic Cap packages, gives an overview of the package development process, describes the Magic Cap development environment, and provides step-by-step instructions and examples for performing basic package development procedures. It contains the following sections:

- Introduction to Magic Cap Packages
- Overview of the Package Development Process
- Understanding the Development Environment
- Selecting, Building, and Running Packages
- Cloning Packages
- Constructing Packages in the Magic Cap Simulator
- Dumping a Package from the Magic Cap Simulator to Magic Developer
- Modifying the Source Code
- Using Magic Script
- Searching with Bowser Jo
- Localizing Packages

Using the information in this guide, you can quickly start developing your own Magic Cap packages. For more detailed information, refer to the *Guide to Magic Cap Development Tools.*

Introduction to Magic Cap Packages

Before you begin developing packages, you should understand what a Magic Cap package is and what it contains.

About Magic Cap Packages

Magic Cap software is distributed in packages that contain objects for performing tasks. A Magic Cap software package is a collection of objects organized to perform a specific set of user functions. Magic Cap provides several built-in packages—such as the datebook, notebook, name card file, and mail packages—that provide services usually handled by application programs on conventional computer systems. In addition, the Magic Cap platform provides a software development environment—called Magic Developer—for creating custom third-party packages.

A package's contents can range from a small set to a large, complex collection, providing users with a single stamp or an entire application. Some packages are like conventional applications, with specific purposes such as electronic mail and personal finance. Other packages perform tasks required by a variety of objects in Magic Cap; these include user-interface features like buttons and clocks.

In addition to differences in purpose, Magic Cap packages can also vary in their structure. Some special purpose packages can contain code that implements certain features, for example an inventory checker. Other packages might not contain any code and will simply add objects to Magic Cap. And other packages may fall somewhere in between these two categories.

Inside a Magic Cap Package

Inside a Magic Cap package, you will find the following four files:

Make File—An MPW build script containing a list of instructions for building a software module. The filename for the make file must be `<package name>.make`. MPW uses the `make` utility to organize the software build process, and Magic Developer also uses this system for building Magic Cap packages. The Makefile contains MPW commands that build your package. You do not have to create or modify the Makefile: it is created automatically when you either use the **Create Build Commands** option from the **Build** menu, or when you start your package by copying an existing package with the **Clone Package** command—the *Clone Package* script automatically generates a Makefile.

C++ Source File—A C++ source file that defines the code that implements the methods of any new classes in the package. The suffix of the filename is `.cpp`. Magic Developer requires that your package have at least one `.cpp` file; if your package has no code, this file may just be an empty file. If a package has only one source file, it usually has the same name as the package plus a `.cpp` suffix.

Class Definition File—A file that contains descriptions of classes that are unique to a given package. The suffix for the filename is `.cdef`. By convention, the filename is `<package name>.cdef`. Class definition files are compiled by CompileClasses during the package build process.

Class definition files are usually named after the classes defined within them. If a package does not define any new classes, it does not need a class definition file. Packages can have more than one class definition file.

Refer to Chapter 8: "Object Tools" in the *Guide to Magic Cap Development Tools* for further details about the class definition file.

Instance Definition File—A file that describes the objects used by a Magic Cap package. The suffix for the filename is `.odef`; by convention `Objects.odef`. Each package must have at least one instance definition file. If a package has more than one instance definition file, the additional files should have names that end with `Objects.odef`. As part of the process of building a Magic Cap package, CompileObjects compiles this instance definition file into a package.

The instance definition file contains textual representations of the package's *static* objects. These are the objects the package creates when it is loaded into the Magic Cap environment. The package usually creates other dynamic objects at runtime, but they are managed by the Magic Cap environment itself.

Refer to Chapter 8: "Object Tools" in the *Guide to Magic Cap Development Tools* for further information about the instance definition file.

Localization files—In addition to the `.make`, `.cpp`, `.cdef`, and `.odef` files, you will find the following files used for localizing the package:

```
<Locale>.Custom.Phrases
<Locale>.Package.Phrases
```

Overview of the Package Development Process

Using Magic Developer and the Magic Cap Simulator to create your package is the first step in the overall package development process. The entire series of steps can be summarized as follows:

1 Create your package in Magic Developer and the Magic Cap Simulator.

This process involves the following basic steps:

- In Magic Developer, clone, build, and run a sample package.
- Construct your package interface by editing the cloned package in the Magic Cap Simulator.
- Dump the package from the Magic Cap Simulator to Magic Developer.
- Make modifications, as necessary, to the source code files.
- Save the source code files.
- Build and run your package.

This guide provides step-by-step instructions and examples for performing the steps above. For further detail, refer to the *Guide to Magic Cap Development Tools.*

2 **Download your packages to your personal communicator.**

Refer to Chapter 3: “Building Software Packages” in the *Guide to Magic Cap Development Tools* for detailed instructions.

3 **Test your packages on the personal communicator and debug them with Magic Developer.**

Understanding the Development Environment

Magic Developer is a software development environment for creating packages for Magic Cap. It includes the following components:

MPW—Magic Developer is based on Apple’s Macintosh Programmer’s Workshop (MPW)—an application development environment for Macintosh software. MPW’s features include a shell for editing text files and launching other tools and a scripting language. Magic Developer provides a set of enhancements to MPW in the form of scripts and tools for building and debugging Magic Cap packages within the MPW environment. In addition, Magic Developer includes modified versions of GNU software tools. The most important of these is Perl, the text processing language used by the localization scripts. Refer to Chapter 4: “MPW Menus for Magic Developer” in the *Guide to Magic Cap Development Tools* for detailed descriptions of the extra MPW menus and menu options supplied by Magic Developer.

The Magic Cap Simulator—While developing Magic Cap packages, you will use this version of Magic Cap that runs on the Macintosh. Typically, you will build your package in Magic Developer, then run it in the Magic Cap Simulator. The Magic Cap Simulator is useful for two purposes:

- The Magic Cap Simulator simulates a personal communicator and allows you to develop software without having to move your package to an actual communicator every time you make a change to your package, thereby saving time in the development cycle.
- The Magic Cap Simulator allows you to graphically create, edit, and specify the behavior of live objects. Once you have modified objects in this way, you can dump them back to MPW as ASCII text in object definition files. Refer to the sections, "Constructing Packages in the Magic Cap Simulator" and "Dumping a Package from the Magic Cap Simulator to Magic Developer" for details.

Bowser Jo—The classes for developing Magic Cap packages are very large and powerful. To help you navigate through these classes, General Magic developed Bowser Jo—a class browser that runs on the Macintosh and allows you view the Magic Cap class hierarchy. Refer to the section "Searching with Bowser Jo" on page 39 for details about using Bowser Jo.

Object Tools—Object Tools are at the heart of Magic Developer. They are MPW tools provided as part of Magic Developer that translate text descriptions of Magic Cap objects into live representations of the objects. The Magic Cap Simulator also has the ability to reverse this operation, converting live objects back to their text representations.

Object Tools process two kinds of files, and each package includes at least one file of each kind (in addition to one or more source files that are processed by conventional compilers and assemblers). The first kind contains the descriptions of any classes defined by the package; this is called a **class definition file.** The second kind, called the *instance definition file*, contains descriptions of objects defined by the package.

Refer to Chapter 8: "Object Tools" in the *Guide to Magic Cap Development Tools* for more information on how to use Object Tools to develop Magic Cap packages.

Magic Script—Magic Cap includes a simple but powerful scripting language called Magic Script that provides a high-level way of arranging objects and connecting them together. Magic Script uses a Java-based model of execution. When you create your own packages, you may write and edit Magic Script scripts in any instance definition file (any file ending in `.odef`). Magic Script is most useful in coordinating user interactions with viewable objects like buttons. It is easier to write small scripts for these than to create the equivalent object subclasses. Refer to the section "Using Scripts" on page 36 for details.

Localization Tools—General Magic provides internationalization support for Magic Cap ROM licensees to develop versions of Magic Cap for different languages. In addition, Magic Developer includes tools for localizing Magic Cap packages for different languages. Refer to the section "Localizing Packages" on page 45 for details about how to use these localization tools to develop localized versions of Magic Cap packages.

Other Tools in Magic Developer—In addition to Object Tools, Magic Developer includes the following other important development tools:

- Online documentation
- MPW scripts for finding text strings in source files
- Scripts for converting existing packages to the new Rosemary API

Selecting, Building, and Running Packages

Before you run a package in the Magic Cap Simulator, you must select and build the package in Magic Developer. To do this, follow these steps:

1. **Start MPW.**

   Double-click the MPW Shell icon.

2. **Select the package you want to build (either a sample package or a cloned package).**

   - To select a sample package:

From the **MagicDeveloper** menu, select **Samples**, then select the desired package.

- To select a package that you have created by cloning:

  From the **MagicDeveloper** menu, select **Packages**, then select the desired package.

3 Build and run the package.

- If you want the package to automatically open and run in the Magic Cap Simulator after a successful build:

  a From the **Build** menu, ensure that the **auto run** option is switched on.

  When **auto run** is checked, Magic Developer automatically runs the package in the Magic Cap Simulator after the package is built.

  b From the **Build** menu, select **Build**.

- If you want only to build the package and run it at a later time:

a From the **Build** menu, select **Build**.

Magic Developer builds the package but does not run it in the Magic Cap Simulator.

b When you are ready to run the package, select the package, then, from the **Build** menu, select **Run Current Package**.

A door labeled with the package name appears in the Magic Cap Simulator Hallway. When you click on the door, it opens and you see the package's scene.

Example 1—Building and Running a Sample Package

In this example, you will practice selecting, building, and running a package. You will select and build the *HelloWorld* sample package.

1 Start MPW.

2 Select the *HelloWorld* package.

From the **MagicDeveloper** menu, select **Samples**, then select **HelloWorld**.

3 Build and run the package.

a From the **Build** menu, ensure that the **auto run** option is switched on.

When **auto run** is checked, Magic Developer automatically runs the package in the Magic Cap Simulator after the package is built.

b From the **Build** menu, select **Build**.

Magic Developer builds the package. When the build is complete, a *HelloWorld* door appears in the Magic Cap Simulator hallway, as shown in Figure 1.

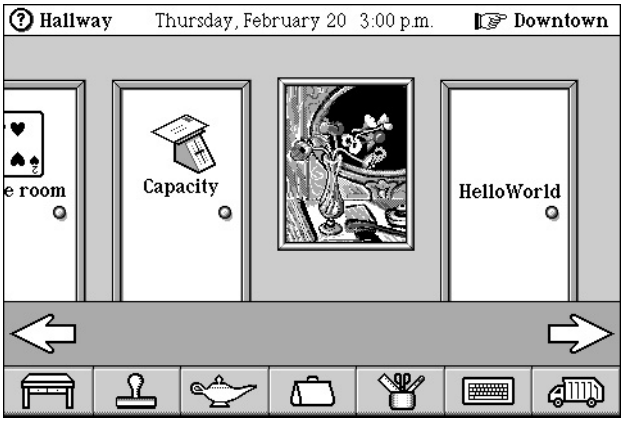

Figure 1 HelloWorld Door in the Magic Cap Simulator Hallway

When you click on the door, it opens and you see the *Hello World* package, as shown in Figure 2.

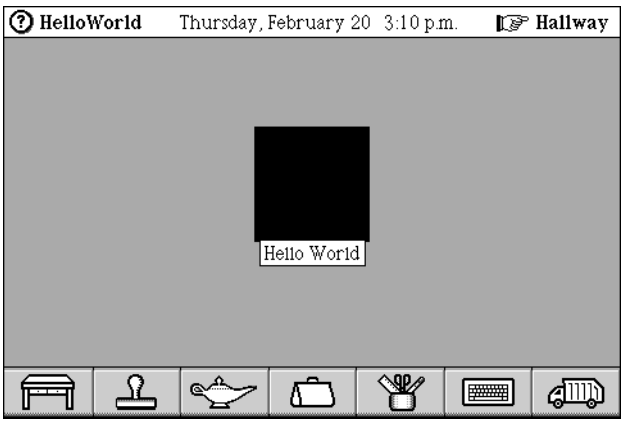

Figure 2 Hello World Package Running

Cloning Packages

You can create packages from scratch—by creating source files manually and using the **Create Builds Commands** option from the **Build** menu to create the make file—but the easiest way to create a Magic Cap package is to clone an existing package and modify it to suit your needs.

Note: When you clone a package, the new package automatically has a Makefile, based on a template Makefile. For more complicated packages, you may wish to develop your own Makefiles. Refer to the *Guide to Magic Cap Development Tools* for instructions.

To clone a package, follow these steps:

1 Start MPW.

2 Select the package you want to clone.

For example, from the **MagicDeveloper** menu, you would select **Samples**, and then select the desired package.

3 Clone the selected package.

a From the **Build** menu, select **Clone Package**.

You are prompted to enter a name for the new package, as shown in Figure 3.

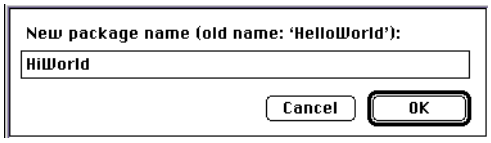

Figure 3 Entering the Name for the Cloned Package

b Enter the desired name, then click *OK*.

Note: Package names should be no longer than 19 ASCII characters and cannot contain spaces.

You are prompted to select the directory in which you want to save the new package, as shown in Figure 4.

Note: A package for the clone will be created in the directory shown in the pop-up menu, not in the directory highlighted in the list.

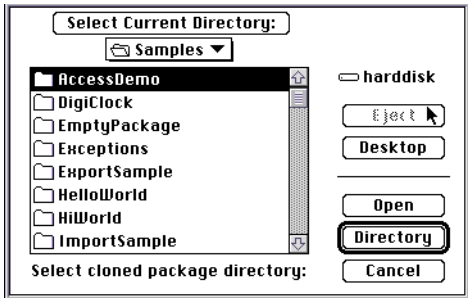

Figure 4 Selecting a Directory in which to Store the Cloned Package

c Select the desired directory, then click *Select Current Directory*.

Magic Developer makes a copy (clone) of the package you selected in step 2, gives the cloned package the name you entered in step 3b, and stores the cloned package in the directory you selected in step 3c.

4 Build and run the cloned package.

Note: If you want to run the package in the Magic Cap Simulator after it is built, select **auto run** from the *Build* menu. When **auto run** is selected, Magic Developer automatically starts and runs the package in the Magic Cap Simulator after a successful build.

From the **Build** menu, select **Build**.

Example 2: Cloning a Sample Package

In this example, you will practice cloning a package. You will clone the *HelloWorld* package you built in Example 1 and name it *HiWorld*.

1 Start MPW.

2 Select the *HelloWorld* package.

From the **MagicDeveloper** menu, select **Samples**, then select **HelloWorld**.

3 Clone the *HelloWorld* package:

a From the **Build** menu, select **Clone Package**.

You are prompted to enter a name for the new package.

b Enter `HiWorld`, then click *OK*.

You are prompted to select the directory in which you want to save the new package.

c Select the desired directory, then click *Select Current Directory*.

Magic Developer makes a copy (clone) of the *HelloWorld* package, gives the package the name *HiWorld*, and stores the cloned package in the directory you selected in step 3c.

4 Build and run the cloned package.

Refer to the section "Selecting, Building, and Running Packages" on page 5 for details.

Note: If you want to run the package in the Magic Cap Simulator after it is built, select **auto run** from the **Build** menu. When **auto run** is selected, Magic Developer automatically starts and runs the package in the Magic Cap Simulator after a successful build.

From the **Build** menu, select **Build**.

The *HiWorld* package appears in the Magic Cap Simulator as shown in Figure 5. Notice that the contents of the *HiWorld* package—including the label for the black box—are identical to those of the *HelloWorld* package from which it was cloned.

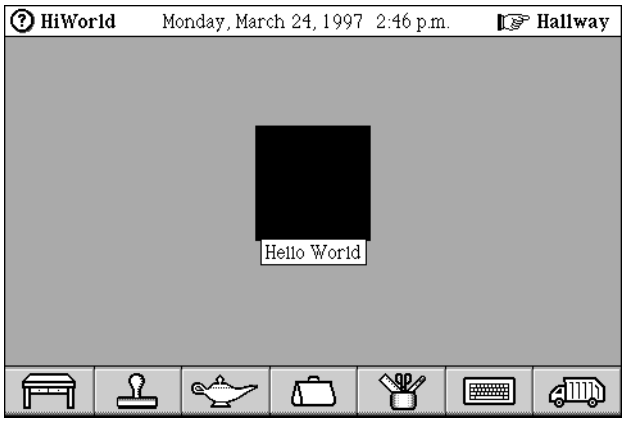

Figure 5 HiWorld Package after Cloning HelloWorld Package

Constructing Packages in the Magic Cap Simulator

In the Magic Cap Simulator, you can use various tools and techniques to develop your package. This process of adding to and changing your package is called construction. In construction mode, you can create new viewable objects by dropping them from the Magic Hat, then modify them with coupons dropped from the Magic Hat and with the Move, Copy, Stretch, and Tinker authoring tools, as described in the next two sections.

Enabling Construction Mode

To switch to construction mode, follow these steps:

1 **In the Magic Cap Simulator Hallway, click the** *Controls* **door.**

The Controls window opens, as shown in Figure 6.

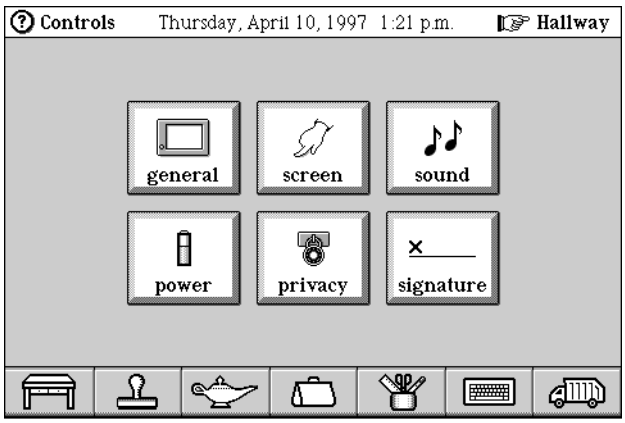

Figure 6 Controls Window

2 Click *general***.**

The General Controls window opens, as shown in Figure 7.

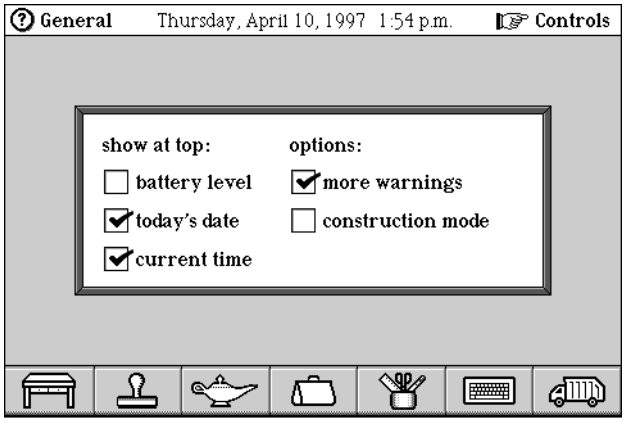

Figure 7 General Controls Window

3 Under options, click *construction mode* **to switch it on.**

The Stamper icon at the bottom of the screen turns into a Magic Hat, as shown in Figure 8.

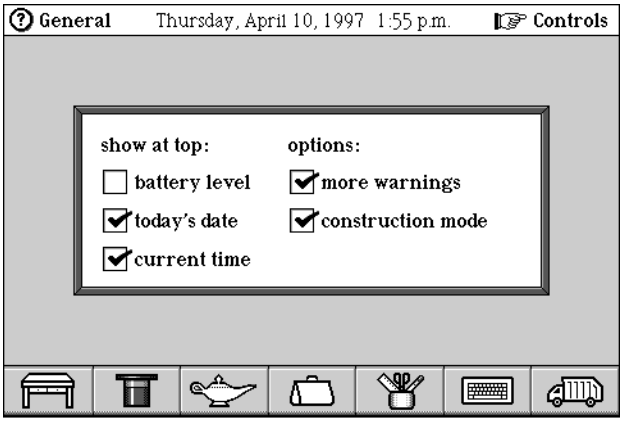

Figure 8 Construction Mode Selected in the General Controls Window

Adding New Viewable Objects

Objects that have a visual appearance are called viewable objects. They are the visible building blocks of the user interface. After you build and run your package, you can modify the package's appearance by adding objects, and then store the changes back into the package's source code.

You can drop the following viewable objects from the Magic Hat to your package:

Stamps—Small pictures used to decorate scenes. You can add stamps to any scene and position them anywhere you like. The Magic Hat contains a collection of different stamps that you can use as clip art in the scenes of your packages. You can also create your own stamps from Macintosh graphics. Refer to the *Guide to Magic Cap Development Tools* for details.

Components—Tools that allow you to build the parts of a package with which users interact. These objects have been developed to solve a variety of user interface problems while maintaining a consistent set of user expectations within the Magic Cap environment.

To add viewable objects to your package, follow these steps:

1. Start MPW.
2. Build and run a package.

   Refer to the section "Selecting, Building, and Running Packages" on page 5 for details.

Note: If you want to run the package in the Magic Cap Simulator after it is built, select **auto run** from the **Build** menu. When **auto run** is selected, Magic Developer automatically starts and runs the package in the Magic Cap Simulator after a successful build.

---

3 In the Magic Cap Simulator, switch on construction mode.

Refer to the section "Enabling Construction Mode" on page 10 for details.

4 Follow these steps to return to the package from the Controls panel (the scene where you switched on Construction mode.)

a Hold down the *option* key and click on the *step back pointer* at the top right corner of the screen.

A list of scenes you've visited recently appears.

b Click the name of the package in the list to return to the package's scene.

5 Click the *Magic Hat*.

The Magic Hat window opens, as shown in Figure 9.

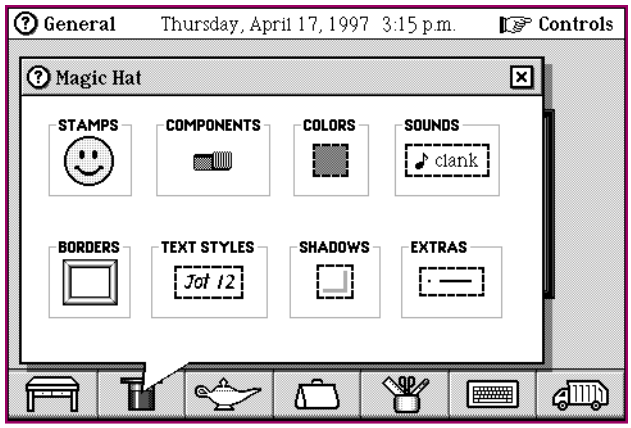

Figure 9 Magic Hat Window

6 Click *stamps* or *components*, depending on the type of object you want to add.

A window displaying the contents of the top drawer of the selected category opens. Figure 10 shows the Components window with the *buttons* drawer open. To open another drawer, click on it. To move to another stack of drawers, click the arrows below the stack to scroll through the stacks of drawers available.

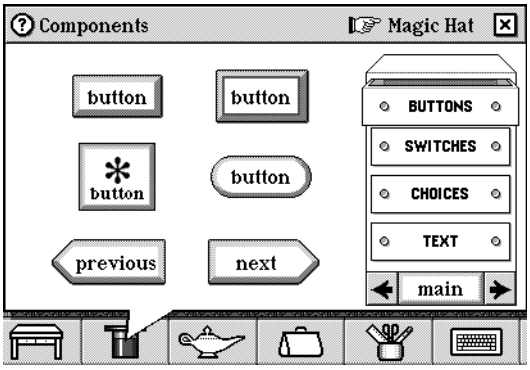

Figure 10 Components Window with BUTTONS Drawer Open

7 Click on the specific object you want to add.

The Stamps or Components window closes, and you can now slide the object to the desired location in your package.

Note: If you want to add more than one object from a particular category, hold down the Option key when you click the first object; the object will be copied from the Magic Hat window to the scene behind, but the window will remain open so you can select additional objects. After you have added all the objects you want, you can click and drag them to the desired locations in your package.

---

8 Dump the package and its objects back into Magic Developer.

Refer to the section "Dumping a Package from the Magic Cap Simulator to Magic Developer" on page 25 for details.

Example 3—Adding a Stamp to a Cloned Package

In this example, you will practice modifying a cloned package by adding a stamp. You will build the *HiWorld* package you created in Example 2, and add a smiley face stamp.

1 Start MPW.

2 Build and run the *HiWorld* package.

Refer to the section "Selecting, Building, and Running Packages" on page 5 for details.

Note: Before you build the package, be sure to select **auto run** from the **Build** menu so that Magic Cap will start and run the package in the Magic Cap Simulator after a successful build.

The *HiWorld* package appears in the Magic Cap Simulator as shown earlier in Figure 5. Notice that the contents of the *HiWorld* package—including the label for the black box—are still identical to those of the *HelloWorld* package.

3 Enter construction mode.

Refer to the section "Enabling Construction Mode" on page 10 for details.

4 Follow these steps to return to the package from the Controls panel (the scene where you switched on Construction mode.)

a Hold down the *option* key and click on the *step back pointer* at the top right corner of the screen.

A list of scenes you've visited recently appears.

b Click the name of the package in the list to return to the package's scene.

5 Add a smiley face stamp to the *HiWorld* package.

a Click the *Magic Hat*.

b Select *stamps*.

c Click the smiley face stamp and slide it to the top of the black box that represents the *HiWorld* package. Figure 11 shows the smiley face added to the *HiWorld* package cloned from the *HelloWorld* package.

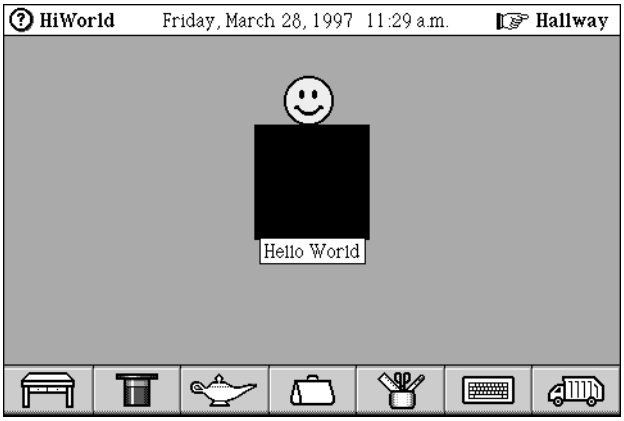

Figure 11 Smiley Face Added to the HiWorld Package

6 Dump the smiley face object to the log file and import it into the package's Objects.odef file.

Refer to the section "Dumping Single Objects" on page 25 for details.

Modifying Viewable Objects

Once you add objects to a package, you can modify these objects with the authoring tools and by dropping coupons from the Magic Hat. In addition, you can create your own custom modifications by copying and pasting graphics from other Macintosh applications.

Using the Authoring Tools

Magic Cap's **authoring tools** include tools that allow you to move, copy, and stretch viewable objects and a Tinker tool that allows you to display an object's label and specify the position for that label. The Tinker tool also allows you to set properties for the object, including whether or not it can be moved or copied.

To use the authoring tools to modify viewable objects in a package, follow these steps:

1 Build and run the package.

Refer to the section "Selecting, Building, and Running Packages" on page 5 for details.

2 In the Magic Cap Simulator, switch on construction mode.

Refer to the section "Enabling Construction Mode" on page 10 for details.

3. Follow these steps to return to the package from the Controls panel (the scene where you switched on Construction mode.)

   a. Hold down the *option* key and click on the *step back pointer* at the top right corner of the screen.

      A list of scenes you've visited recently appears.

   b. Click the name of the package in the list to return to the package's scene.

4. Click the *Tool holder* at the bottom of the screen.

   The Pencils Tools window opens.

5. Click the right or left arrow until you see the authoring tools in the Tools window, as shown in Figure 12.

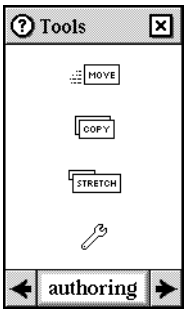

Figure 12 Authoring Tools

6. Select the tool you want to use: Move, Copy, Stretch, or Tinker (the wrench).

   The Tools window closes, and the authoring tool you selected becomes the current tool, with its icon replacing the Tool Holder icon at the bottom of the screen.

7. Use the selected tool to modify a viewable object.

   Using the Move and Stretch Tools

   To use the Move and Stretch tools to modify an object:

   Click and drag the object to move or stretch it.

   Using the Copy Tool

   To use the Copy tool to modify an object:

   Click the object to copy it.

Using the Tinker Tool

To use the Tinker tool to modify an object, follow these steps:

a Click the object you want to modify to open the Tinker window for that object. Figure 13 shows the Tinker window for the smiley face stamp.

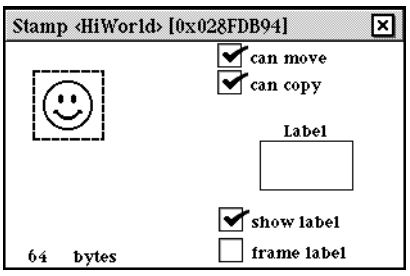

Figure 13 Tinker Window for the Smiley Face Stamp

b Use the Tinker window to set properties for the object and to select and position a label:

- Specify whether or not an object can be moved or copied and whether or not it will appear with a label by selecting or clearing the *can move*, *can copy*, and *show label* check boxes.
- Specify whether or not an object's label will appear with a frame by selecting or clearing the *frame label* check box.
- Select the location for the label by dragging the word *Label* to the desired location on the box that appears above the *show label* check box. This box represents the object you selected in step 5a. You can select one of 15 positions for the label by clicking the position you want. In the example shown in Figure 13, the label will appear above the smiley face.

c Close the Tinker window by clicking the close box in the top right corner of the window.

d If you added a label, use the coupon maker to add or change text for the label. Refer to the section "Creating Text Coupons" on page 21 for details.

Using Coupons from the Magic Hat

Coupons from the Magic Hat hold intangible attributes which you can apply to viewable objects. The Magic Hat includes coupons that allow you to make the following modifications to viewable objects:

Colors—You can change the color that fills an object by dropping a color coupon onto it. Most viewable objects accept color coupons. The Colors window also contains a color grid, which allows you add color coupons to multiple objects without reopening the Magic Hat each time.

Sounds—Drop a sound coupon onto an object to specify a digitized or synthesized sound that will play when the object is tapped. You can drop sound coupons onto most viewable objects. To hear the sound in a sound coupon, click the coupon. If you don't specify a sound, switches and buttons will play the touch sound when tapped.

Borders—Use a border coupon to add or change an object's border. To remove the border from an object, use the *no border* coupon. Boxes, fields, the Inspector, meters, and any class that inherits from "HasBorder" accept borders.

Text styles—Use text style coupons to change the text style in text fields and labels.

Shadows—Use shadow coupons to add or remove shadows from objects. Shadows can enhance the appearance of an object. You can drop shadows onto most viewable objects.

Extras—Use coupons from the extras category to modify objects in a variety of ways, including changing line styles.

A coupon has a thick dashed border around it. Each coupon is good for one change to a viewable object, and you can slide coupons without first getting the move tool. When you drag a coupon over another object, the object will highlight if it can accept the coupon. For example, you cannot drop a border onto a stamp, so the stamp does not highlight when you drag a border coupon over it.

Some objects have multiple parts, and you can drop different coupons onto each part. For example, boxes have content, border, and label parts. You can drop different color coupons onto the border, content, and label, thereby setting the different parts to different colors.

To modify a package's objects with coupons from the Magic Hat, follow these steps:

1. Launch MPW.
2. Build and run a package.

   Refer to the section "Selecting, Building, and Running Packages" on page 5 for details.

   Note: If you want to run the package in the Magic Cap Simulator after it is built, select **auto run** from the **Build** menu. When **auto run** is selected, Magic Developer automatically starts and runs the package in the Magic Cap Simulator after a successful build.

3. In the Magic Cap Simulator, switch on construction mode.

   Refer to the section "Constructing Packages in the Magic Cap Simulator" on page 10 for details.

4. Click the *Magic Hat*.

   The Magic Hat window opens, as shown earlier in Figure 9.

5. Click the type of coupon you want to use: colors, sounds, borders, text styles, shadows, or extras.

   A window displaying coupons in the selected category opens.

6 Select a coupon and use it to modify a viewable object.

Selecting and Using Color Coupons:

When you select *colors*, the Colors window opens, as shown in Figure 14. This window displays the available color coupons. You can either click one of these coupons or you can click the color grid in the lower right corner of the window.

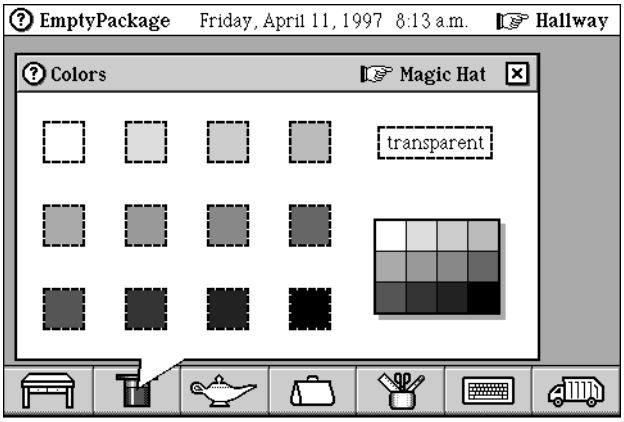

Figure 14 Colors Window

- If you select a single coupon, the Colors window closes and the color coupon you selected remains on the screen. Click and drag the coupon onto the object you want to modify. The coupon disappears and the object is filled with the coupon's color.
- If you select the color grid, the Colors window closes, and the color grid remains on the screen. Click and drag a coupon from the grid onto the object you want to modify. The object is filled with the coupon's color, but the grid remains on the screen, allowing you to add color coupons to other objects without opening the Magic Hat again. Use the move tool to slide the color grid into the Trash truck when you are done using it.

Selecting and Using Sounds, Borders, or Text Styles Coupons:

When you select *sounds*, *borders*, or *text styles*, a window displaying the contents of the top drawer of the selected category opens. Figure 15 shows the Sounds window with the *standard* drawer open. To open another drawer, click on it. To move to another stack of drawers, click the arrows below the stack.

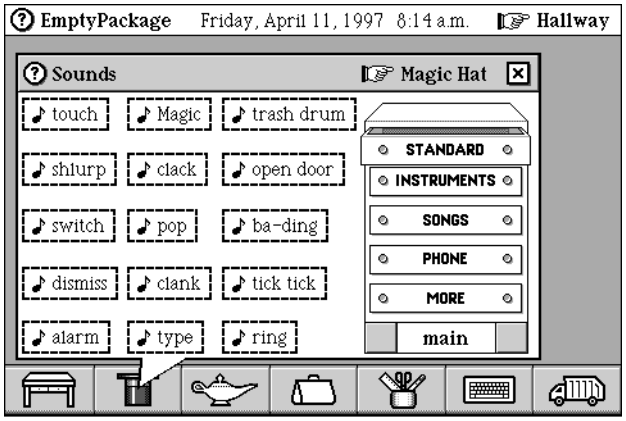

Figure 15 Sounds Window

To select a coupon, click on it. The window closes, and the coupon remains on the screen. Click and drag it onto the object you want to modify.

Selecting and Using Shadows and Extras Coupons:

When you select *shadows* or *extras*, a window displaying the available coupons in the selected category opens. Figure 16 shows the Shadows window. You can either click one of these coupons or you can click the coupon chooser in the lower right corner of the window.

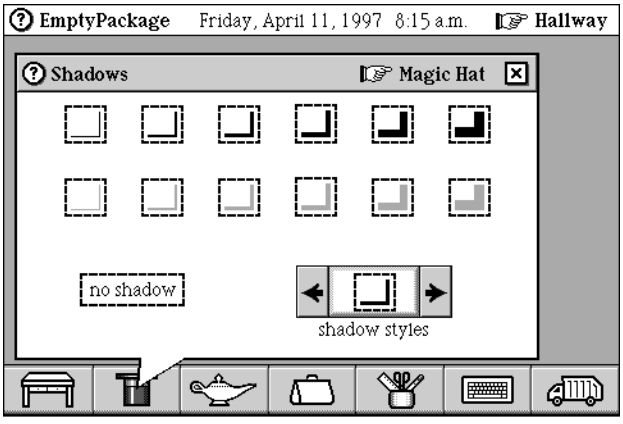

Figure 16 Shadows Window

- If you select a single coupon, the window closes and the coupon remains on the screen. Click and drag the coupon onto the object you want

modify. The coupon disappears and the object is modified based on the coupon you dropped.

- If you select the coupon chooser, the window closes, and the coupon chooser remains on the screen. Click the arrow keys on the coupon chooser until it displays the desired coupon. Then click and drag the coupon from the center of the coupon chooser onto the object you want to modify. The object is modified based on the coupon you dropped, but the chooser remains on the screen, allowing you to add coupons to other objects without opening the Magic Hat again. Use the move tool to slide the coupon chooser into the Trash truck when you are done using it.

Creating Text Coupons

When you drop a text coupon onto an object, the text displayed in the coupon replaces the object's label. You can drop a text coupon on any object that accepts one—a Telecard, a notebook page, or an name card, for example.

To create a text coupon, follow these steps:

1 Hold down the *option* key and click the *Keyboard*, located at the bottom of the screen.

The labelmaker appears. The labelmaker is simply the Keyboard with a labelmaker above it.

2 Click the Keyboard keys to type the desired text for the text coupon.

As you type, the labelmaker creates the text coupon, as shown in Figure 17.

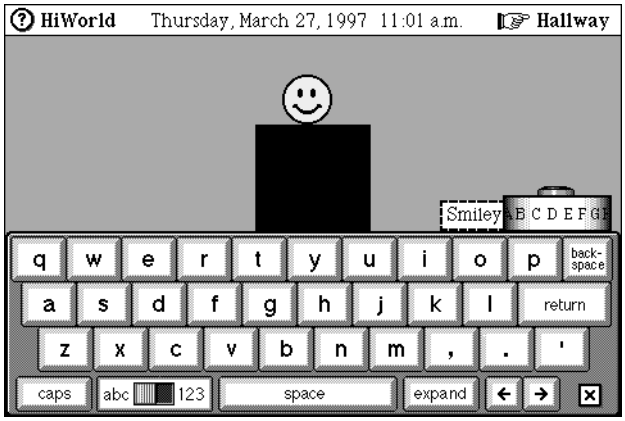

Figure 17 Creating a Text Coupon with the Labelmaker

3 When you are done typing, click the *text coupon*.

The labelmaker disappears, and the text coupon remains on the screen.

4 Slide the text coupon onto the desired object.

The text you typed in step 2 now replaces the object's label.

Example 4—Modifying Objects in the HiWorld Package

In this example, you will add a label to the smiley face stamp you added to the *HiWorld* package in Example 3, change the size of the package's black box, and change the label for the black box.

1 Build and run the modified *HiWorld* package you created in Example 2.

Refer to the section "Selecting, Building, and Running Packages" on page 5 for details.

Use the steps that follow to add a label to the smiley face.

2 In the Magic Cap Simulator, click the *Tool holder* at the bottom of the screen.

3 Click the right or left arrow until you see the authoring tools in the Tools window, shown earlier in Figure 12.

The authoring tools appear only when construction mode is switched on. Refer to section "Enabling Construction Mode" on page 10 for details.

4 From the authoring tools, select the Tinker tool (the wrench).

The Tools window closes, and the Tinker tool becomes the current tool, with the Tinker tool icon replacing the Tool holder icon at the bottom of the screen.

5 Click the *smiley face*.

The Tinker window for the smiley face opens, as shown in Figure 18.

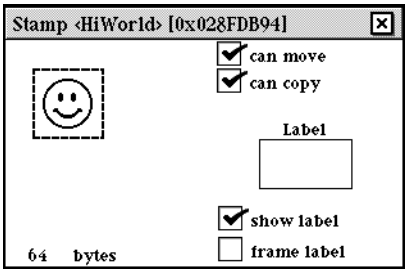

Figure 18 Tinker Window for the Smiley Face

6 Click the *show label* checkbox so that it is switched on.

A checkbox appears in the checkbox to show that the setting is switched on.

Note: If you want the label to appear with a frame, also select *frame label*. It is easier to determine where to drop the text coupon (step 9d) if your label has a frame.

---

7 Select the location for the label by sliding the word *Label* to the desired location on the box that appears above the show label check box.

This box represents the smiley face. In this example the label will appear above the smiley face.

8 Close the Tinker window.

9 Add text for the label.

a Hold down the *option* key and click the *Keyboard* at the bottom of the screen.

The labelmaker appears.

b Click the keyboard keys to type `smiley` on the labelmaker.

c When you are done typing, click the text coupon.

The labelmaker disappears, and the text coupon remains on the screen.

d Slide the text coupon onto the smiley face's label.

Note: You must drop the text coupon onto the label area—not the smiley face stamp itself. If you selected *frame label* in step 6, you will see the label frame into which you must drop the text coupon. If you did not select *frame label*, you will not see a label frame; you must estimate where the frame is (based on the position you selected in step 7) and drop the text coupon onto that location.

After you drop the coupon, the text you typed appears in the label, in the position you specified in step 7. Figure 19 shows a framed label added to the smiley face. Figure 20 shows an unframed label added to the smiley face.

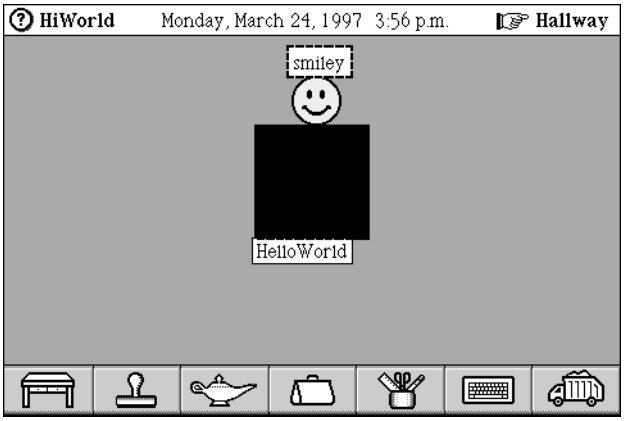

Figure 19 Text Coupon Added to the Smiley Face's Label (frame label selected)

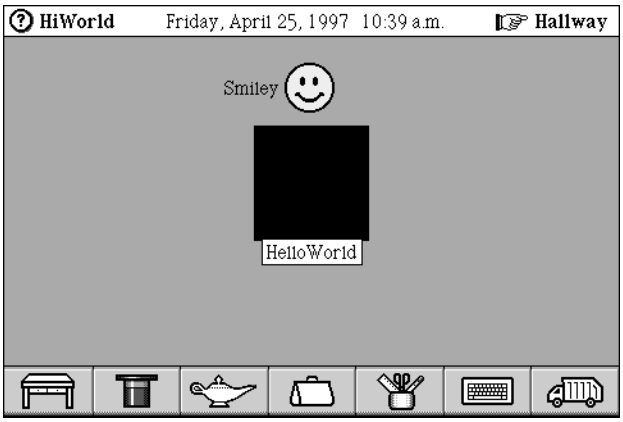

Figure 20 Text Coupon Added to the Smiley Face's Label (frame label not selected)

Use the steps that follow to change the size of the black box.

10 Click the *Tool holder* at the bottom of the screen.

11 Click the right or left arrow until you see the authoring tools, shown earlier in Figure 12.

12 From the authoring tools, select the *stretch tool*.

The Tools window closes, and the stretch tool becomes the current tool, with the stretch tool icon replacing the Tool holder icon at the bottom of the screen.

13 Click and drag the black box to resize it.

14 When the box is the size you want, click the *stretch tool* icon at the bottom of the screen to get out of stretch mode.

Note: If you want to move the black box after you have resized it, use the move tool. (Follow the steps above, but select the move tool instead of the stretch tool in step 12.)

Change the label for the black box to match the package name:

15 Using the labelmaker, as described in step 9, create a text coupon with the text `HiWorld`.

16 Slide the text coupon onto the label area for the black box.

Figure 21 shows the label for the black box changed to “HiWorld.”

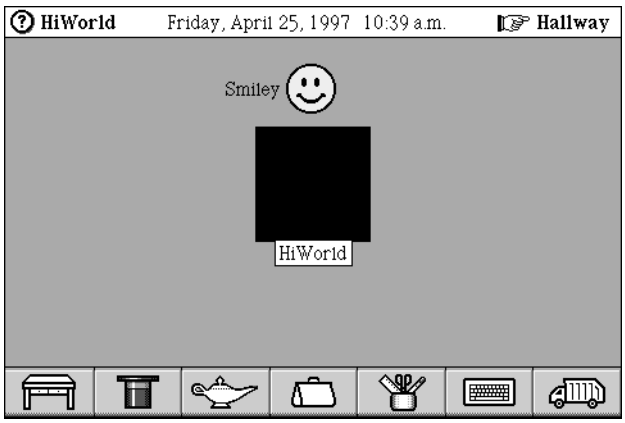

Figure 21 Label for the Black Box Changed from HelloWorld to HiWorld

17 Dump the modified objects to the log file.

Refer to the section "Dumping Single Objects" on page 25 for details.

Dumping a Package from the Magic Cap Simulator to Magic Developer

Once you have constructed a package in the Magic Cap Simulator, you then dump it back into Magic Developer where you can make any necessary modifications to the package’s source files. You can then build and run your package.

The procedure you use to dump a package from the Magic Cap Simulator to Magic Developer varies, depending on whether you want to dump a single object or the entire package.

- **Object dumping** is useful when you are changing or adding single objects at a time to your package.
- **Package dumping** is useful when you have made wholesale changes to your package’s interface.

Dumping Single Objects

To dump a single object from the Magic Cap Simulator to Magic Developer, follow these steps:

Note: These instructions assume that you are in the Magic Cap Simulator, with the package running.

---

1 Dump any objects that you added or modified to the log file.

a From the **Examine** menu, select **Show Inspector**.

The Inspector window opens, as shown in Figure 22. This window displays a list of the current hierarchy of viewable objects, called the view list.

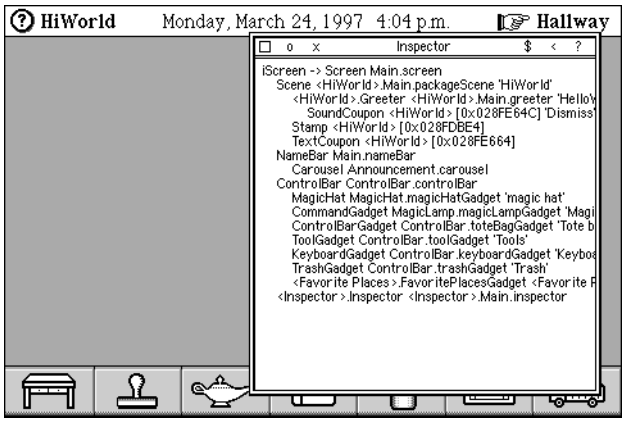

Figure 22 Inspector Window

b Click the question mark in the top corner of the Inspector window.

c Click the first object that you want to dump.

The object that you clicked becomes the Inspector's target object, and the Inspector window now displays the fields for this object. Figure 23 shows the Inspector window displaying the fields for the smiley face stamp that you added to the *HiWorld* package in Example 3.

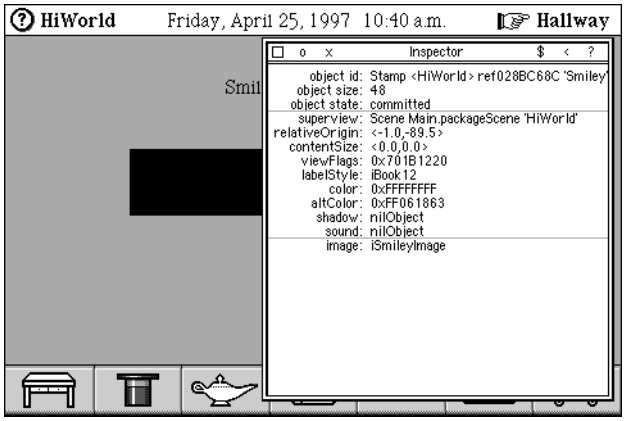

Figure 23 Inspector Window Listing the Fields for the Smiley Face

d From the **Examine** menu, select **Dump Inspector Target**.

The selected object is converted to its text representation (in CompileObjects; syntax), and the text representation is dumped to the log file.

Note: If you want to dump the object to the clipboard, hold down the Shift key when you select the **Dump Inspector Target** option.

---

e Repeat steps a-d for any other objects you added or modified and want to dump, clicking the appropriate object in step c.

2 Paste the dumped objects into the `Objects.odef` file.

a Go back to MPW.

b From the **Open** menu, open the `Objects.odef` file. If the file is already open, just click its window to make it the current window.

c From the **Utils** menu, select **Open Log** to open the log file. In this file, you will find the objects you dumped in step 1.

Note: If you pasted the object to the clipboard in step 1d, you can skip this step and go directly to step 2d.

---

d Use the **Copy** and **Paste** commands from the **File** menu to copy the objects you dumped from the log file and paste them into the Objects.odef file.

Note: If you pasted the object to the clipboard in step 1d, you can simply use the **Paste** command.

---

3 Save the `Objects.odef` file.

4 Build and run the modified package.

Dumping an Entire Package

To dump an entire package from the Magic Cap Simulator to Magic Developer, follow these steps:

Note: These instructions assume that you are in the Magic Cap Simulator, with the package running.

---

1 Dump the objects to the package's log file.

From the **Examine** menu, select **Dump Package**.

You are prompted to enter an instance definition file name and to select a directory in which to save the file. You can replace the original instance definition file in your package directory.

2. Enter a filename and select the directory in which you want to save the new instance definition file.
3. Build and run the modified package.

Modifying the Source Code

Magic Cap software is constructed with conventional software development tools and some unique tools developed by General Magic. These tools implement the Magic Cap Object model by preparing source files for the C/C++ compiler and by binding together the compiled package.

The planning and design process usually involves building a Magic Cap package from existing classes. The actual C programming you will do occurs when you have to develop new classes or override the methods of an existing class, as described in the following sections.

Adding an Instance Definition to the Objects.odef File

Your package must include at least one instance definition file to be compiled by CompileObjects. For the purposes of grouping or managing large numbers of objects, you may need to add instance definitions to your package.

Figure 24 shows an example of an instance definition.

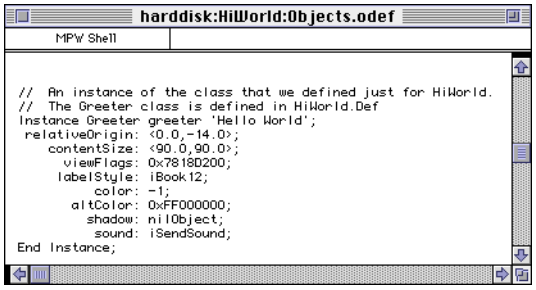

Figure 24 Instance Definition in an Objects.odef File

Each instance definition includes three parts:

Instance header—A single line that consists of the keyword `instance`, followed by the name of the object's class, followed by a symbolic tag that must be unique for this package, followed by an optional object name which may contain spaces and is enclosed by single quotes, ending with a semicolon. The unique tag for `SoftwarePackageContents` must be `contents`.

Body—One or more lines that list the object's fields and their values.

Instance footer—A single line that consists of the keywords `end instance` followed by a semicolon.

To add a new instance definition, follow these steps:

1. Build the package to which you want to add the instance definition.

   Refer to the section "Selecting, Building, and Running Packages" on page 5 for details.

2. From the **File** menu, open the `Objects.odef` file.
3. Add the new instance definition, using the syntax described above.
4. Save the `Objects.odef` file and close it.
5. Build and run the modified package.

Adding a New Class to the .cdef File

If your package includes any new classes, as most packages do, you must define them in a class definition file to be compiled by CompileClasses. Figure 25 shows an example of a class definition.

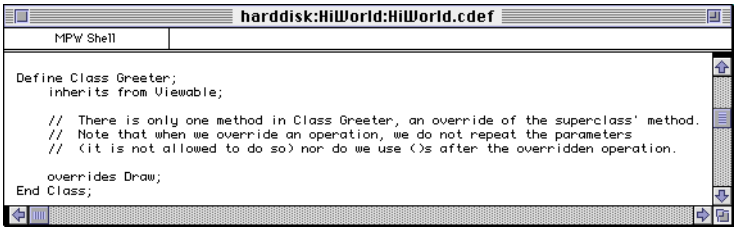

Figure 25 Class Definition in a .cdef File

Each class definition includes four parts:

Class header—One or more lines that consist of the keywords `define class`, followed by the name of the new class, optionally followed by other information about the class.

Superclass designation—The keywords `inherits from`, followed by the name of the class's immediate superclass.

Body—One or more lines that list the class's fields, operations, and attributes. Overridden methods are specified with `overrides`.

Footer—A single line that consists of the keywords `end class` followed by a semicolon.

Refer to Chapter 8: "Object Tools" in the *Guide to Magic Cap Development Tools* for further information about class definition files and their syntax.

To add a new class, follow these steps:

1. Build the package to which you want to add the class.

   Refer to the section "Selecting, Building, and Running Packages" on page 5 for details.

2. From the **File** menu, open the `.cdef` file.
3. Add the new class, using the syntax described above.
4. Save the `.cdef` file and close it.

Note: When you create an instance of a class, the .odef file must contain a read statement that reads in the file that contains the definition of that class. For example, `read "MagicCap.cdef"` allows class definitions to be read in for all system classes, while `read "HelloWorld.cdef"` allows Greeter objects to be created in the *HelloWorld* sample package.

---

5. Build and run the modified package.

Example 5—Adding a New Class to the Source Code

In this example, you will add a new class "Face" to the source code for the *HiWorld* package.

1. Build the modified *HiWorld* package you created in Example 4.

   Refer to the section "Selecting, Building, and Running Packages" on page 5 for details.

2. From the **File** menu, open the `Hiworld.cdef` file.
3. Add a new class "Face" that inherits from class "Stamp," as shown in Figure 26.

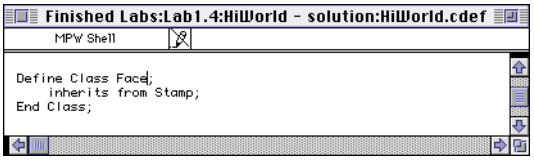

Figure 26 Adding a New Class "Face" to the Hiworld.cdef File

4. Save the `Hiworld.cdef` file and close it.
5. Modify the objects in the `Objects.odef` file to use the new classes.

   a. From the **File** menu, open the `Objects.odef` file.

b In the `Objects.odef` file, modify the instance definition for the Smiley face to use class “Face” instead of class “Stamp,” as shown in Figure 27.

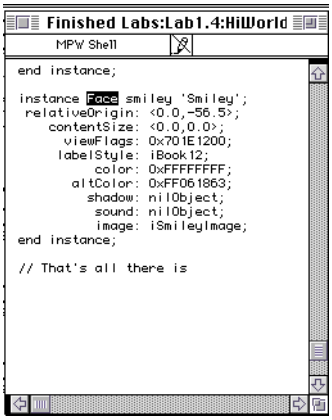

Figure 27 Changing the Smiley Face Object Instance Definition

6 Save the `Objects.odef` file and close it.

7 Build and run the modified package.

Defining Code to Implement Methods of New Classes

If your package includes any new classes, you must define the code that implements the methods of the new classes in C++ files. You can arrange your source code in any collection of files. MPW and Magic Developer require the `.cpp` suffix.

Although you can write most of a `.cpp` file in standard C, you must be aware of these important special elements:

- `#include` statements are used to include a number of files provided with Magic Developer; see the sample packages for the exact list of files to be included. You might also have your own header files that would be referenced with an include statement.
- The file must include a statement that specifies the class, and the class name in this statement must match your class name exactly, including case. If the class names do not match, your package may not compile. If you define more than one class, make sure that you set the class name correctly for each method. You can have as many of these class specifiers as you need, switching from one class to another before each method if necessary.
- Each function declaration for a method begins with the keyword `Method`.
- In Magic Cap, each method’s name is the name of the class, followed by an underscore, followed by the operation name (as defined in the operation statement in the class definition file). CompileClasses uses this convention to match operations to C++ functions. Of course, your source code can have functions that are not methods, and there are no naming restrictions on them.
- The first parameter to every method is an object reference number. For convenience, this parameter is never declared in the class definition file. However,

it must be included when you declare the methods in the source files, or the compiler will not know about it.

Example 6—Building a Package that Performs a Specific Function

In this example you will practice cloning and modifying a package. You will build a temperature converter that converts from Fahrenheit to Celsius and vice versa. If you change a number in either system (Fahrenheit or Celsius), the converter will automatically convert it to the other system.

1. **Start MPW.**
2. **Clone the** *EmptyPackage* **package, giving the new package the name TemperatureExample.**
3. **Build and run the** *TemperatureExample* **package.**

Note: Before you build the package, be sure to select **auto run** from the **Build** menu so that Magic Cap will start and run the package in the Magic Cap Simulator after a successful build.

---

4. **In the Magic Cap Simulator, enter construction mode.**

   Refer to the section "Constructing Packages in the Magic Cap Simulator" on page 10 for details.

5. **Add two meters to the TemperatureExample package (drag and drop the meter on the TemperatureExample package twice).**

   The meter is located in the *choices* drawer in the Components window. Refer to the section "Adding New Viewable Objects" on page 12 for details.

6. **Dump the package to the log file.**

   a. From the **Examine** menu, select **Dump Package**.

   b. Go to MPW.

   c. From the **Utils** menu, select **Import Dumped Object**.

   Refer to the section "Dumping a Package from the Magic Cap Simulator to Magic Developer" on page 25 for details.

7 Tie both meters together.

a Define a converter class (**“**TemperatureConverter”) that converts back and forth from Fahrenheit to Celsius and add this class to the TemperatureExample.cdef file, as shown in Figure 28.

- The class should be defined with two fixed fields: *Fahrenheit* and *Celsius*.
- Flag “getter” generates the auto-getter methods “Fahrenheit” and “Celsius” for both fields.
- The methods return the value of the fields.

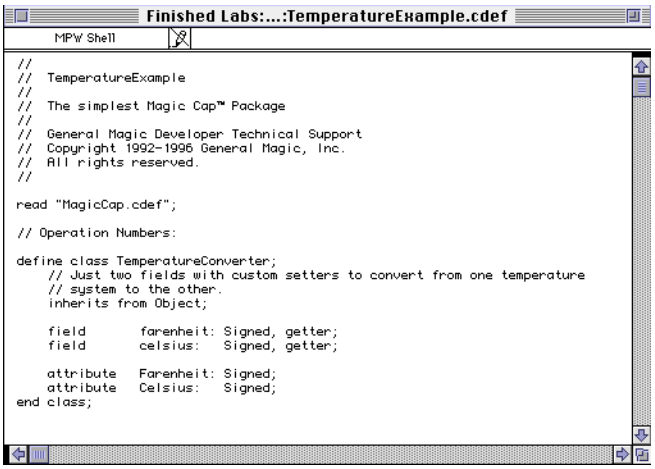

Figure 28 Adding class “TemperatureConverter” to the .cdef File

b Create instance definitions for the two meters and the converter and add them to the Objects.odef file, as shown in Figure 29.

- Use the converter class as the target for both meters. In other words, link the meter to its target.
- Add the temperature conversion with the target attributes “operation_Fahrenheit” and “operation_Celsius.”

The definition and implementation of the class “TemperatureConverter” are described in the following substeps.

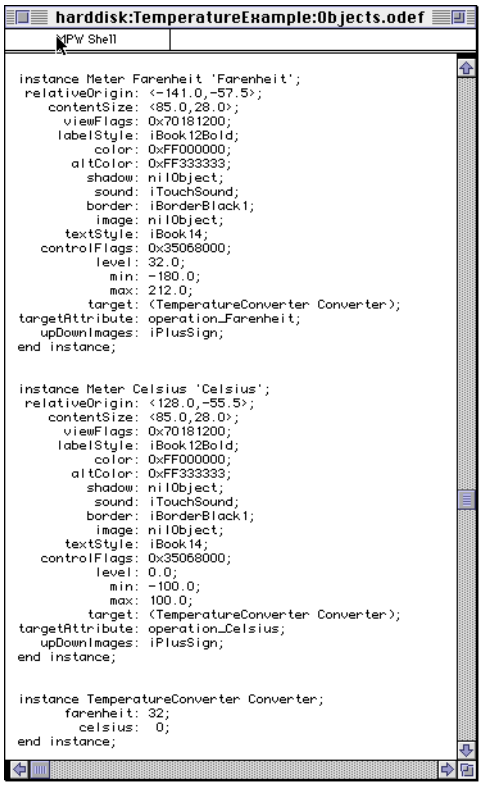

Figure 29 Adding Object Instance Definitions for the Fahrenheit and Celsius Meters and the TemperatureConverter Converter

c Implement the methods of the new class “TemperatureConverter” in a C++ source file, as shown in Figure 30. In this file, you will implement the custom setters that do the conversions (f=(9/5)c+32 and c=(5/9)(f-32)) and set the value into the fields. These setters are named “Set<FieldName>”—Method

"SetFahrenheit" converts from Fahrenheit to Celsius. Method "SetCelsius" converts from Celsius to Fahrenheit.

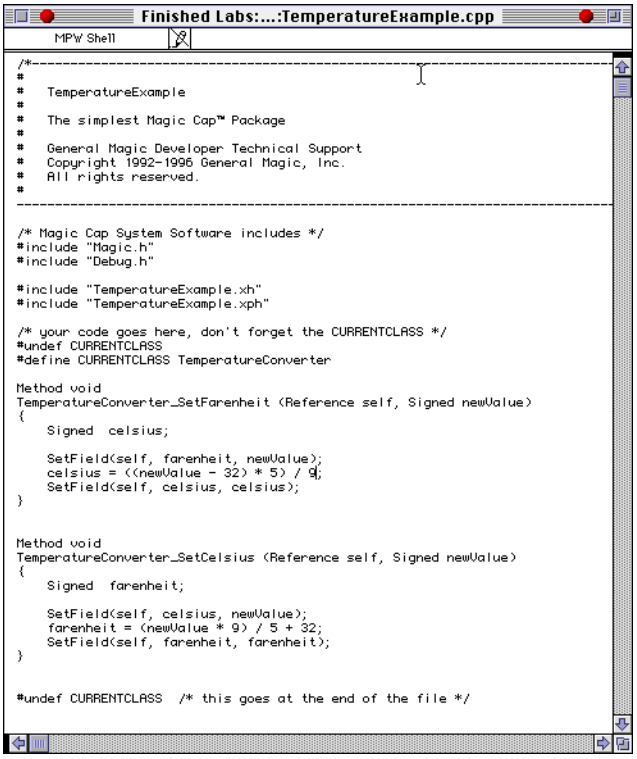

Figure 30 C++ Source File for TemperatureExample Package

d From the File menu, select Save to save the changes to the files.

8 Build and run the modified package.

Figure 31 shows the finished *TemperatureExample* package. When you change the temperature on either meter (by clicking - or +), the other meter will change accordingly.

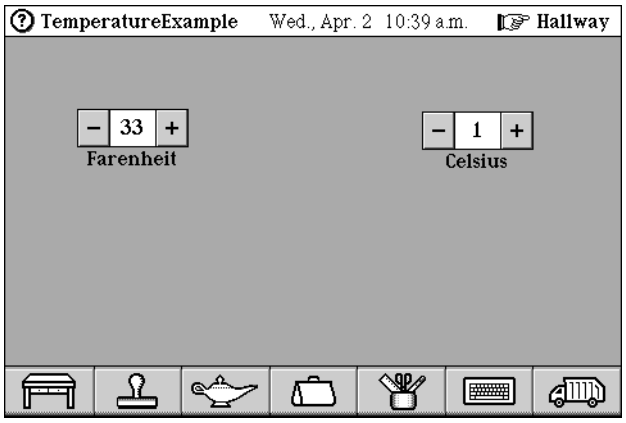

Figure 31 TemperatureExample Temperature Conversion Package

Note: Every control (including a meter) has a controlFlags word. The lower nibble of the upper word (mask: 0x000F 0000) denotes the type of data this meter tracks. The meter in the Magic Hat has this as 0 (object). More common values would be 4 (fixed), 5 (unsigned short), or 6 (signed short). If the meter displays fixed values, you must set the TemperatureConverter object to use fixed-point math also. The fixed-point methods are in `Math.cdef`.

Using Scripts

Magic Cap scripting for objects uses a script language that is compiled when a package is built. You can write and edit Magic Cap scripts in instance definition files; they will be assembled as part of the build process. If your script has syntax errors, you will receive error messages when your instance definition file is compiled.

To attach a script to an object, follow these steps:

1 From the **File** menu, Open the `Objects.odef` file.

2 Attach the script to the object's instance definition.

Use either the term `script` or the term `ScriptedMethod`. The following two lines have the same effect:

```
Instance Button makeLikeThis 'Sounds like' Action (script
scriptTag)
```

```
Instance Button makeLikeThis 'Sounds like' Action
(ScriptedMethod scriptTag)
```

3 Add the script itself to the `Objects.odef` file.

The skeleton for a script is as follows:

```
script scriptTag
        script prototype is [<prototype>]

        script statements
end script;
```

Note the following:

- If a script omits its prototype statement, it is assumed to have the same prototype as method Action, which takes a Reference and returns void. Scripts must specify their prototypes if the method to which they are attached has a different prototype.
- If a script needs, to return something, it must include a return statement. Scripts can have more than one return statement.
- Script statements must end with semicolons.

Refer to Chapter 8: “Object Tools” in the *Guide to Magic Cap Development Tools* for further details about scripting for objects and Magic Script, including a description of the script language, the Java virtual machine byte codes its script interpreter uses, and the underlying object format its scripts use.

Example 7—Adding User Interface Components to an Empty Package

In this example, you will practice adding user interface components to an empty package. You will add a button and a slider—a user interface component that allows users to control continuously adjustable levels—to the *EmptyPackage* sample package.

1 Start MPW.

2 Clone the *EmptyPackage* package, giving the new package the name *SimpleControl*.

3 Build and run the *SimpleControl* package.

Refer to the section "Selecting, Building, and Running Packages" on page 5 for details.

Note: Before you build the package, be sure to select **auto run** from the **Build** menu. When auto run is selected, Magic Developer starts and runs the package in the Magic Cap Simulator after a successful build.

---

4 In the Magic Cap Simulator, enter construction mode.

Refer to the section "Constructing Packages in the Magic Cap Simulator" on page 10 for details.

5 Add a button and a slider to the *SimpleControl* package.

a Click the *Magic Hat*.

b Select *components*.

The Components window opens, as shown earlier in Figure 10.

c Hold down the Option key and click a button to drop it on the empty package.

d Click the *choices* drawer to open it, as shown in Figure 32.

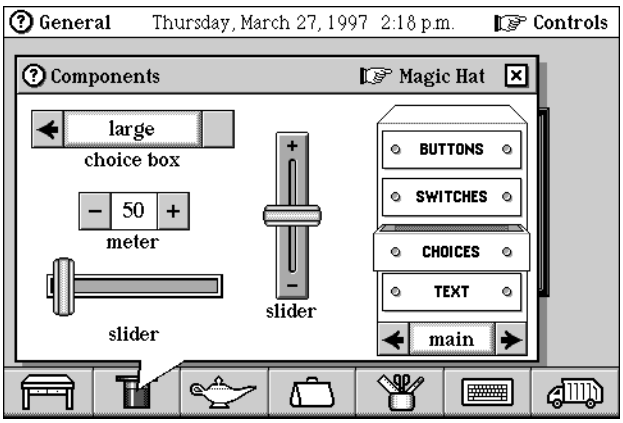

Figure 32 Contents of the Choices Drawer

e Click a slider.

The Components window disappears.

f Drag the button and slider to the desired locations in the package.

g Create a text coupon with the text `Slide to the end` and drop it on the button. Refer to the section "Creating Text Coupons" on page 21 for details.

Figure 33 shows the button and slider added to the *SimpleControl* package.

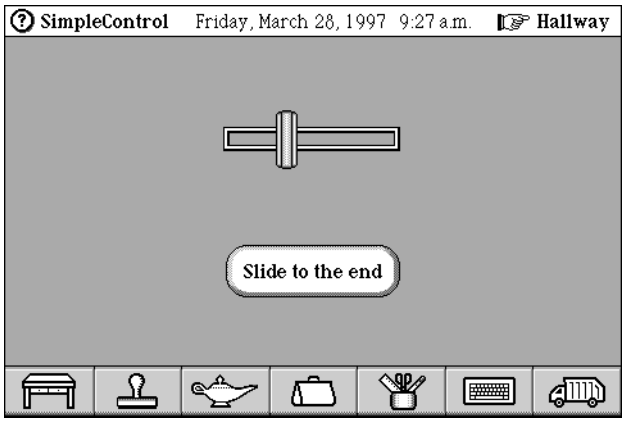

Figure 33 Button and Slider Added to the SimpleControl Package

6 Dump the package back into Magic Developer.

Refer to the section "Dumping an Entire Package" on page 27 for details.

7 Add an action to the button in the *SimpleControl* package.

a From the **File** menu, open the Objects.odef file.

b Attach the *maxOut* script to the Button object and add the script itself in the Objects.odef file, as shown in Figure 34. This script adds an action affecting

the slider to the button—the slider will slide to the right end when the button is pushed.

Figure 34 MaxOut script Added to the Objects.odef File

8 Build and run the modified *SimpleControl* **package.**

Searching with Bowser Jo

Bowser Jo is an HTML-based class browser for Magic Cap classes. With it you can look up a class and access information about that class, including its superclasses, subclasses, fields, operations, and attributes. You can also specify a search string and search for classes, methods, and fields associated with that string.

Note: In order to use Bowser Jo, you must have a Java-enabled web browser (such as Netscape Navigator or Internet Explorer) installed on your system.

To search with Bowser Jo, follow these steps:

1 Start MPW.

2 From the Utils menu, select Bowser Jo.

Magic Developer finds and launches your web browser and loads an HTML page for Bowser Jo that displays “Magic Cap Alphabetical Class Index ’A’,” as shown in Figure 35.

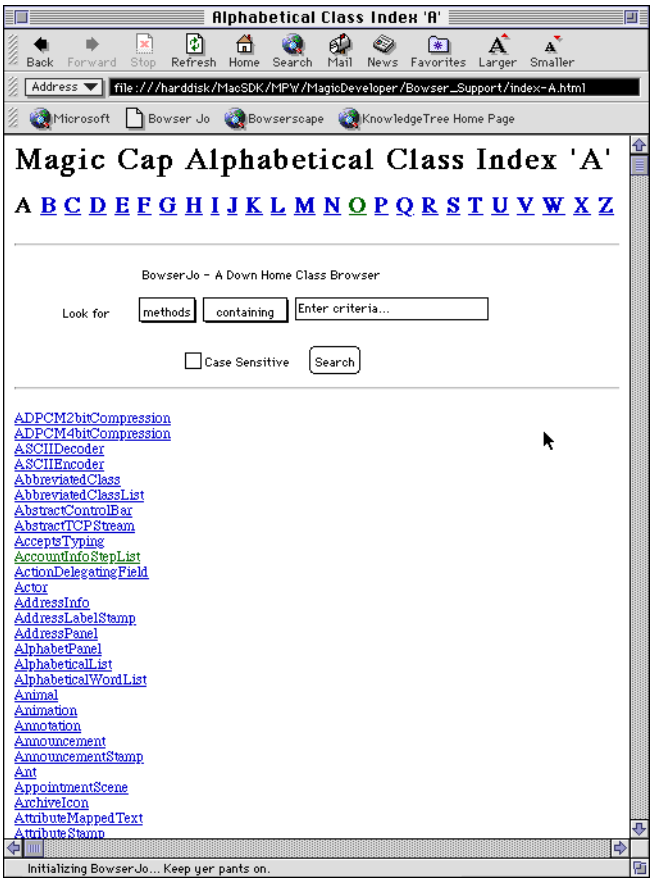

Figure 35 Bowser Jo Window

3 Use Bowser Jo to search alphabetically for a class or to search for Magic Cap constructs that are associated with a particular string.

You can then access class reference information, as described in step 4.

Note: If you want to find information about a class, and you know the name of the class, you can simply *search alphabetically* in the Magic Cap Alphabetical Class Index.

---

a Using the row of letters at the top of the Bowser Jo window, click on the first letter in the class’s name to display a list of Magic Cap constructs beginning with that letter. As shown above in Figure 35, the “A List” (Magic Cap Alphabetical Class Index ’A’) appears by default when you open the Bowser Jo window.

b Use the scroll bar to scroll through the list of classes until you find the desired class.

Note: If you want to find Magic Cap constructs that are associated with a string, you can *search by string matching* in the Bowser Jo window to display the classes, methods, or fields associated with the specified string.

---

c Select an option from the first pull-down menu to specify whether you want to search for *classes*, *methods*, or *fields*.

d Select an option from the second pull-down menu to specify whether you want to search for classes, methods or fields *containing*, *equaling* or *starting with* the specified string.

e Type the desired string in the *Enter criteria...* field to specify the string for which you want to search.

Figure 36 shows the search portion of the Bowser Jo window set up to search for *classes containing* the string *button*.

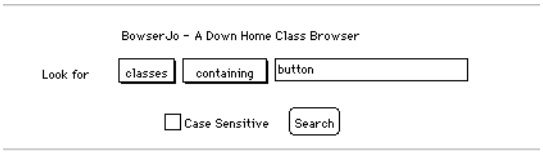

Figure 36 Bowser Jo Window Set up to Search for Classes Containing the String "Button"

f If you want to perform a case-sensitive search, select the *Case Sensitive* check box.

g Click *Search*.

A Search Results window opens, displaying the results of the search. Figure 37 shows the Search Results window for the search set up in Figure 36 (all classes containing the string "button"). As you can see in the figure, this search brings

up 29 classes containing “button.” Use the scroll bar to scroll through the list of classes.

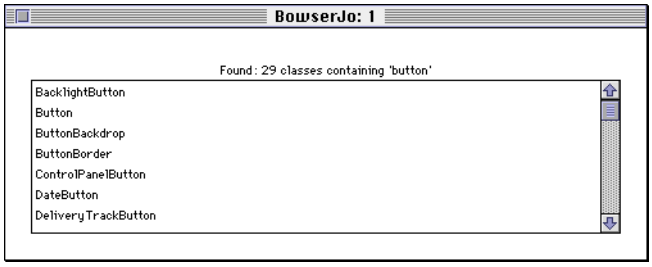

Figure 37 Search Results Window

Note: When you search for fields or methods, the Search Results window displays the results in the following format:

```
<class name>_<field or method name>
```

---

4 Display reference information for the class(es) that you found in step 3.

- For classes that you found in the Magic Cap Alphabetical Class Index, click the class name to display the reference information window.
- For classes that you found in the string matching Search Results window, double-click the class name to display the reference information window.

Figure 38 shows the first screen of reference information for the class “Button.” The reference information consists of the following:

- A list of classes from which the class inherits. The class inherits directly from the first class listed (in this example, “Stamp”) and inherits certain attributes from the other (mixin) classes (in this example “HasBorder” and “HasTimedAction”).
- The class’s subclasses (the list of classes on the right of the screen).
- A hierarchy of the class’s direct superclasses (in this example Object -< Viewable -< Stamp -< Button). Click on a superclass to display the class definition for that class.
- A list of operations belonging to the class.
- A list of fields belonging to the class and each of its superclasses; fields belonging to the superclasses are listed first, with the fields belonging only to the class itself appearing at the end of the list.
- A list of attributes belonging to the class.
- The instance definition for the class.
- The class definition for the class.

Use the scroll bar to scroll through the information. Click on a class, operation, field, or attribute name to display further information about that item.

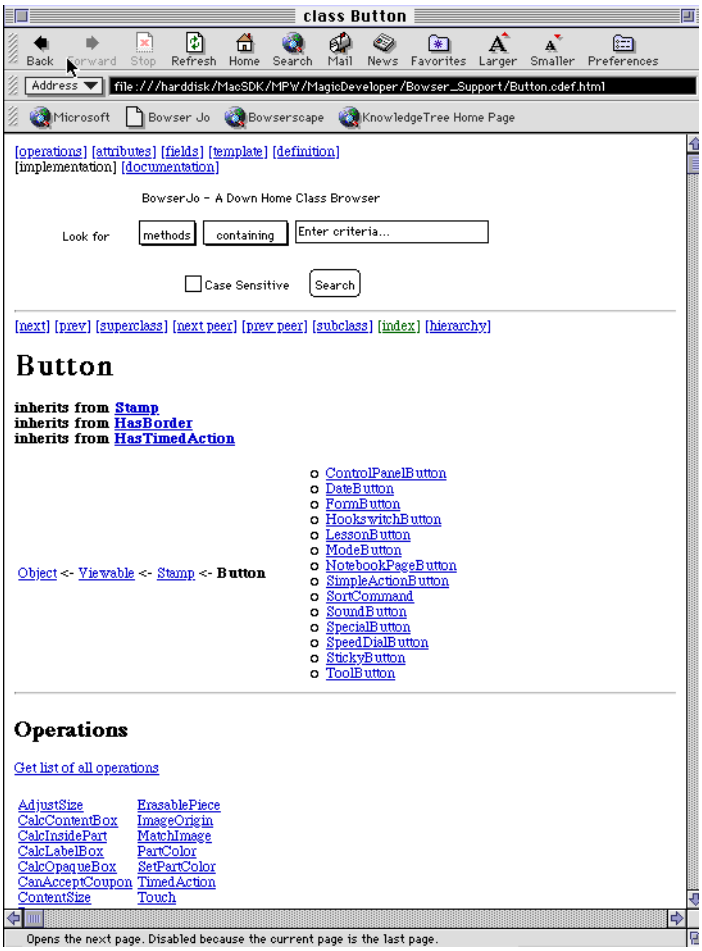

Figure 38 Information Window for Class “Button”

Example 8—Using Bowser Jo

This example is an exercise in finding classes and subclasses in Magic Cap. You will practice navigating the Magic Cap class hierarchy and accessing reference information using Bowser Jo.

Here are some questions that you might want to answer by using Bowser Jo:

1 How can you set an animation to turn right when it hits any wall?

a Display the reference information for class “Animation” (find the class in the Magic Cap Alphabetical Class Index and click on the class name). Refer to the section "Searching with Bowser Jo" on page 39 for details.

b Locate “canTurnRight” in the fields list.

c Set the value for “canTurnRight” to value true.

2 What are the subclasses of class “Window.”

a Display the reference information for class “Window” (find the class in the Magic Cap Alphabetical Class Index and click on the class name). Refer to the section "Searching with Bowser Jo" on page 39 for details.

b The information window lists all the subclasses for class window on the right of the screen.

3 What are the fields that class “SimpleActionButton” has but class “Button” does not?

a Display the reference information for class “SimpleActionButton” (find the class in the Magic Cap Alphabetical Class Index and click on the class name). Refer to the section "Searching with Bowser Jo" on page 39 for details.

b Notice that “Button” is a superclass of “SimpleActionButton.”

c Scroll down to the fields list. As shown Figure 39, the last two fields—”target” and “operation”—belong only to class “SimpleActionButton,” not to any of its superclasses.

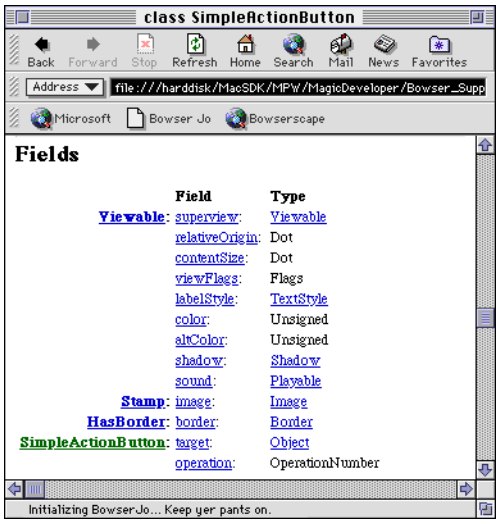

Figure 39 Field List for class SimpleActionButton

4 What are the mixin classes from which class “Viewable” inherits?

a Display the reference information for class “Viewable” (find the class in the Magic Cap Alphabetical Class Index and click on the class name). Refer to the section "Searching with Bowser Jo" on page 39 for details.

b Click the superclasses of class “Viewable” to display information about these classes. You will find that all superclasses of class “Viewable” except “Object” are mixin classes.

5 What operations does class “ObjectList” override?

a Display the reference information for class “ObjectList” (find the class in the Magic Cap Alphabetical Class Index and click on the class name). Refer to the section "Searching with Bowser Jo" on page 39 for details.

b Scroll down to the class definition information to find the list of overridden operations (At, InstallInto, FindElementAfter, etc.)

6 What is the difference in implementation between MakeValid for class "Object" and MakeValid for class "ObjectList"?

a Display the reference information for class "ObjectList" (find the class in the Magic Cap Alphabetical Class Index and click on the class name). Refer to the section "Searching with Bowser Jo" on page 39 for details.

b Notice that class "ObjectList" inherits from class "Object." In other words, "ObjectList" inherits all the operations in "Object."

c Scroll down to the class definition information.

d Notice that the operation "MakeValid" is overridden. Since "MakeValid" is inherited from "Object" (in which the operation is originally defined), this is the difference in implementation between the two classes.

Localizing Packages

Magic Cap licensees can develop localized versions of Magic Cap communicators for different national markets, and Magic Cap package developers can also develop localized versions of their packages for these different markets.

The key to developing a localizable Magic Cap package is to isolate localizable features so that you can localize the package for different languages without modifying the source code. Magic Developer includes localization tools that help package developers separate the tasks of feature development and localization.

Much of the effort in localizing a Magic Cap package is in translating text strings. Example 9 illustrates the basic steps for localizing the text in a package. Refer to Chapter 9: "Package Localization" in *the Guide to Magic Cap Development Tools* for information about localization tools and localization files and complete instructions for localizing packages.

Example 9—Localizing the Text in a Package

In this example, you will practice localizing the text in a package. You will change the *HiWorld* package's text from US English to Japanese.

Note: You must have Claris FileMaker Pro, version 3.0 to localize text to Japanese. In addition, Japanese localization requires the Japanese version of FileMaker Pro.

---

1 Launch Magic Developer.

2 Choose a locale.

From the **Target** menu, select **Japan**.

3 Choose an existing package to localize.

For the purposes of this example, select the *HiWorld* package.

4 Extract the text strings from the object instance definition file.

From the **Utils** menu, select **Create Phrase File for File Maker**.

Magic Developer creates a phrase file with the name `Japan for File Maker` in the *HiWorld* package's folder.

5 Launch the FileMaker Pro application and open the localized phrase file template.

Open the Phrase file templates folder (inside the SDK folder) and double-click `Japan Phrasefile template`.

This launches Japanese FileMaker Pro and opens the localized phrase file template.

6 Import the phrase file into the localized FileMaker Pro phrase file template.

a From the FileMaker Pro **File** menu, select **Import / Export**, then **Import Records**.

The FileMaker Pro application displays an Import dialog box. Locate the file you created in step 4, select it, then click *Open*.

The Specify Field Order for Import dialog box appears, as shown in Figure 40.

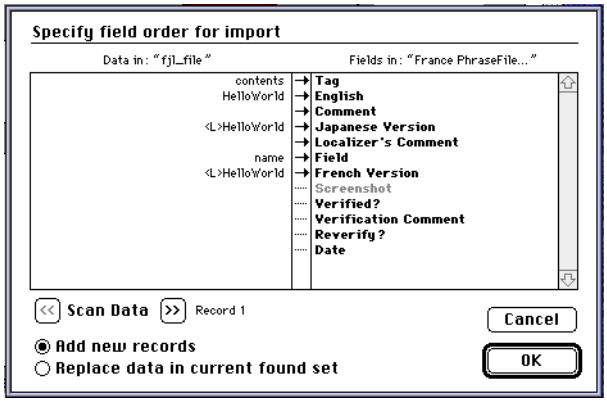

Figure 40 Specify Field Order for Import Dialog Box (FileMaker Pro)

b Select the fields you want to import.

In most cases, you will accept the fields that are already selected. At a minimum, be sure to select the following: *Tag*, *International English*, *locale version (Japanese Version* for the purposes of this lab), and *field*.

c Click *OK*.

7 Translate the phrases using the FileMaker Pro template.

Each object to be localized has a separate record. The phrase file template is an extensible FileMaker Pro template that provides you with several fields:

Tag name—The name of the object used in the instance definition file.

Field name—The field of the object to be localized.

English string—The original English text string.

Localized string—The field that holds the localized string. Before localization, it contains a copy of the English string preceded by <L>.

In addition, the template includes several informational fields that you can use to control your work flow. These include a Screenshot window that can hold a graphical representation of a viewable object that needs to be modified for localization.

You can navigate from one object record to the next by using FileMaker Pro's book mechanism. This a rolodex-like icon in the upper left corner of the phrase file template.

8 Export the text strings from FileMaker Pro to a localized phrase file.

a From FileMaker Pro's **File** menu, select **Import / Export**, then **Export Records**.

b Enter the name under which you want to save the localized phrase file.

Note: Be sure to use an English string for this filename because the MPW scripts used by Magic Developer are based on English language names.

---

c Select *tab-separated text* format.

d Click *OK*.

The FileMaker Pro application displays an Export dialog box. You use this dialog box to select the fields to export. This list is the same as that in the Import dialog box you used in step 6.

e Select the fields you want to export. Choose the same fields that you chose to import in step 6.

f Select the *Don't format output* button.

g Click *OK*.

9 Convert the localized phrase file into a Unicode-based phrase file.

a From the MPW **Utils** menu, select **Create Locale.Package.Phrases file**.

The standard file locator dialog box appears.

b Select the filename you entered in step 8b.

c Click *Open*.

10 Build and run the localized package.

Note: Before you build the package, be sure to select **auto run** from the **Build** menu so that Magic Cap will start and run the package in the Magic Cap Simulator after a successful build.

When the build is complete, the localized *HiWorld* package appears in the Magic Cap Simulator hallway, as shown in Figure 41.

Figure 41 HiWorld Package Localized for Japan