SDK Tutorial

for Microsoft Windows

April 24, 2000

SDK Tutorial for Microsoft Windows

(version 4/17/00)

License

Your use of the software discussed in this document is permitted only pursuant to the terms in a software license between you and Icras.

Trademarks

Icras, the Icras logo, DataRover, the DataRover logo, DataRover Remote Access Kit, Magic Cap, the Magic Cap logo, and the rabbit-from-a-hat logo are trademarks of Icras, Inc. which may be registered in certain jurisdictions. The Magic Cap technology is the property of General Magic, Inc., and is used under license to Icras, Inc. Microsoft, Developer Studio, Visual Studio, and Visual C++, are all trademarks of Microsoft Corporation.

All other trademarks and service marks are the property of their respective owners.

Limit of Liability/Disclaimer of Warranty

THIS BOOK IS SOLD “AS IS.” Even though Icras, Inc. has reviewed this book in detail, ICRAS MAKES NO REPRESENTATION OR WARRANTIES, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THE ACCURACY OR COMPLETENESS OF THE CONTENTS OF THIS BOOK. ICRAS 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 may not apply to you.

Patents

The Magic Cap software is protected by the following patents: 5,611,031; 5,689,669; 5,692,187; and 5,819,306. Portions of the Magic Cap technology are patent pending in the United States and other countries.

United States Government Restrictions

This product is “commercial item” as that term is defined at 48 C.F.R. 2.101 (OCT 1995) consisting of “commercial computer software” and “Commercial computer software documentation,” as such terms are used in 48 C.F.R. 12.212 (SEPT 1995) and is provided to the U.S. Government only as a commercial end item. Consistent with 46 C.F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (JUNE 1995), all U.S. Government End Users acquire this product only with those rights set forth therein.

Icras, Inc.

955 Benecia Avenue
Sunnyvale, CA 94086 USA

Tel.: 408 530 2900
E-mail: info@icras.com
Fax: 408 530 2950
URL: http://www.icras.com/

Table of Contents

List of Examples

1

Introduction

This chapter introduces you to Magic Cap packages, gives an overview of the package development process, and describes the Magic Cap development environment. It contains the following sections:

- Introduction to Magic Cap packages
- Overview of the package development process
- Understanding the development environment

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 such as 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, such as an inventory checker. Other packages might not contain any code and 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 files:

Makefile—A file containing a list of instructions for building a software module. The filename for the makefile must be *PackageName*.make. Developer Studio uses the nmake utility to organize the software build process, and Magic Developer also uses this system for building Magic Cap packages. The makefile contains Developer Studio commands that build your package. If you create a new package by cloning an existing package, Developer Studio creates a makefile that you can use as a template. You can modify this makefile, if necessary, as you add files to your project.

C++ Source File—A C++ source file 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 *PackageName*.cdef. Class definition files are compiled by the Class Compiler 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.

See Chapter 6, “Object Tools,” in the *Guide to 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 .odef. As part of the process of building a Magic Cap package, the Object Compiler 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.

See Chapter 6, “Object Tools,” in the *Guide to 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:

   a. In Magic Developer, clone, build, and run a sample package.

   b. Construct your package interface by editing the cloned package in the Magic Cap Simulator.

   c. Dump the package from the Magic Cap Simulator to Magic Developer.

   d. Make modifications, as necessary, to the source code files.

   e. Save the source code files.

   f. Build and run your package.

   This guide provides step-by-step instructions and examples for performing these steps. For further detail, see the *Guide to Development Tools*.

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

   You can use the WinPCLink tool provided, or use the download.bat script. See “Downloading a package to a storage card” on page 29 in the *Guide to Development Tools* for more information about using the download.bat script.

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

   See Chapter 5, “Debugging Tools,” in the *Guide to Development Tools*.

Understanding the development environment

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

Developer Studio

Magic Developer is based on Microsoft's Developer Studio, an application development environment for Windows software. Magic Developer extends the Developer Studio environment with tools for building and testing Magic Cap packages. See Chapter 2, "Building Software Packages," in the *Guide to Development Tools* for more information.

Magic Cap Simulator

While developing Magic Cap packages, you'll use a version of Magic Cap running on the PC called Magic Cap Simulator that lets you create, edit, and specify the behavior of live objects graphically. This allows you to see and use packages much as they will appear on personal communicators.

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, saving time in the development cycle.

Magic Cap Simulator is useful for two purposes. You can use it to run and test packages with greater convenience than downloading the package into a communicator. In addition, you can use it in the software development process to modify objects and then dump them back as ASCII text in object definition files.

See Chapter 3, "Magic Cap Simulator," in the *Guide to Development Tools* for more information on how to use Magic Cap Simulator to create and modify objects for your packages.

Bowser Jo

The classes for developing Magic Cap packages are very large and powerful. To help you navigate through these classes, Icras, Inc. developed Bowser Jo, a tool for viewing the Magic Cap class hierarchy that runs in a Web browser. See Chapter 4, "Bowser Jo," in the *Guide to Development Tools* for more information.

Debugging tools

Part of developing a Magic Cap package is the necessary process of finding and fixing bugs. Magic Developer provides two debugging environments for Magic Cap package development:

- Developer Studio and the Magic Cap Simulator
- GDB and the communicator

See Chapter 5, “Debugging Tools,” in the *Guide to Development Tools* for more information on how to use the debugging tools to develop Magic Cap packages.

Object tools

Object tools translate text descriptions of Magic Cap objects into live, graphical representations of the objects. The Magic Cap Simulator object tools can also 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.

See Chapter 6, “Object Tools,” in the *Guide to Development Tools* for more information on how to use object tools to develop Magic Cap packages.

Magic Script

Magic Cap uses a simple but powerful scripting language called **Magic Script** to provide a high-level way of arranging and connecting objects. Magic Script uses a Java based model of execution. When you create your own packages, you may write and edit Magic Script in any object definition file. For an example, see the sample package TicTacToe.

Magic Script is useful in coordinating user-interactions with viewable objects like buttons. From a performance perspective, writing a script and creating an object subclass are equivalent. See “Magic Script” on page 96 in the *Guide to Development Tools* for more information.

Localization tools

Icras, Inc. provides internationalization support to develop versions of Magic Cap for different languages. In addition, Magic Developer has tools for localizing Magic Cap packages for different languages.

See Chapter 7, “Package Localization,” in the *Guide to Development Tools* for more information on how to use these localization tools to develop localized versions of Magic Cap packages.

2

Magic Developer Quick Start

This chapter provides step-by-step instructions and examples for performing basic package development procedures. It contains the following sections:

- Building and running packages
- Cloning packages
- Constructing packages in the Magic Cap Simulator
- Dumping objects and packages
- Modifying the source code
- Using scripts
- 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, see the *Guide to Development Tools*.

Building and running packages

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

1. Start Developer Studio.

   Double-click the Developer Studio icon on the desktop.

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

   Choose File ▸ Open Workspace, then select the desired workspace.

3. Build the package.

   Choose Build ▸ Build *PackageName* or press *F7*.

4. The first time you test a package it will be necessary to establish the executable and package execution environment. You will only have to do this one time:

   a. If you have not already done so, ensure that the default configuration for your package is "Win32 USA Debug". If this is not the default configuration, select the Build ▸ Set Active Configuration and set the default as specified.

   b. Select Project ▸ Settings and click the Debug tab.

   c. In the "Executable for debug session" text box you need to point to the Magic Cap Windows Simulator. Do so by clicking the button on the right and navigating to:

      <installation directory>\debug\win32\MagicCap-USA.exe

      where <installation directory> is where you installed MagicDeveloper.

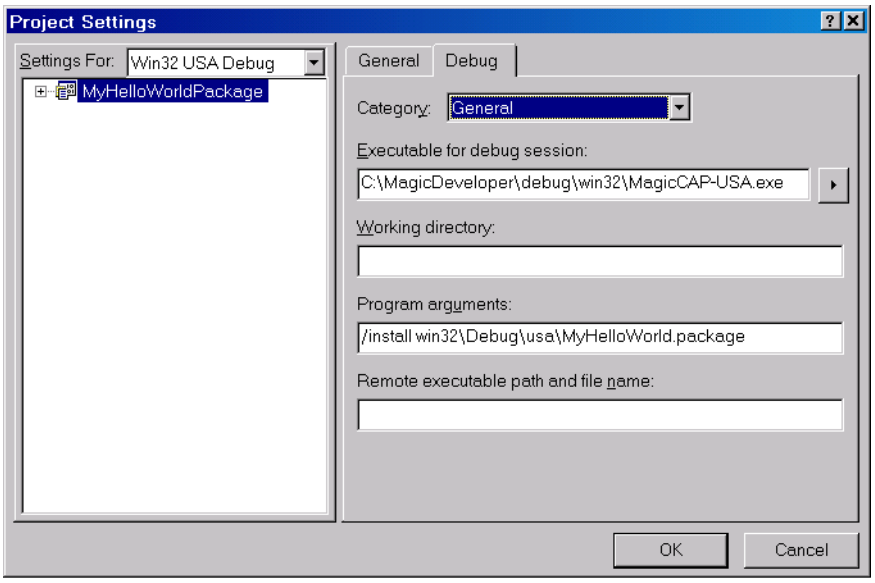
   d. In the "Program arguments" text box you need to tell the simulator where to find your package image file. Do so by typing the following:

      /install win32\debug\usa\<package name>.package

      where <package name> is the name of your package, and then click OK. For example, assuming that you created a package named MyHelloWorld, the command line would be:

      /install win32\debug\usa\MyHelloWorld.package

   e. Select File ▸ Save Workspace to save your changes. Below is a sample of the result when MagicDeveloper is installed in C:\MagicDeveloper.



5. Run the package.

   Choose Build▶Start Debug▶Go or press *F5*.

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

Example 2-1 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 Developer Studio.**

2. **Select the HelloWorld package.**

   Choose File▶Open Workspace, then navigate to the `HelloWorld` folder and open `HelloWorldPackage.dsw`.

3. **Build the package.**

   Choose Build▶Build HelloWorld or press *F7*.

4. Run the package.

Choose Build▶Start Debug▶Go or press *F5*.

Magic Developer builds the package. When the build is complete, an AhoyWorld door appears in the Magic Cap Simulator hallway.

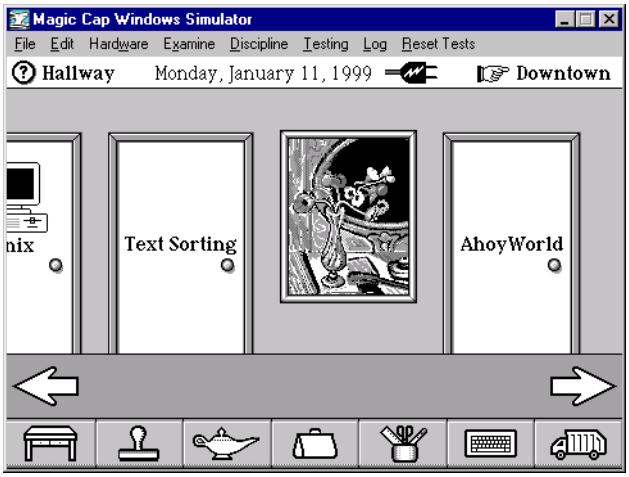


Figure 2-1 AhoyWorld door in the Magic Cap Simulator hallway

When you click the door, it opens and you see the Hello World package.

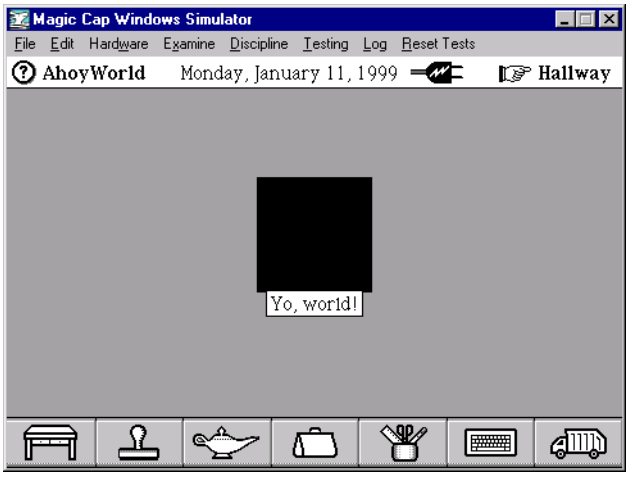


Figure 2-2 Hello World package running

Cloning packages

The easiest way to create a Magic Cap package is to clone one of the sample packages and modify it to suit your needs.

To clone a package, follow these steps:

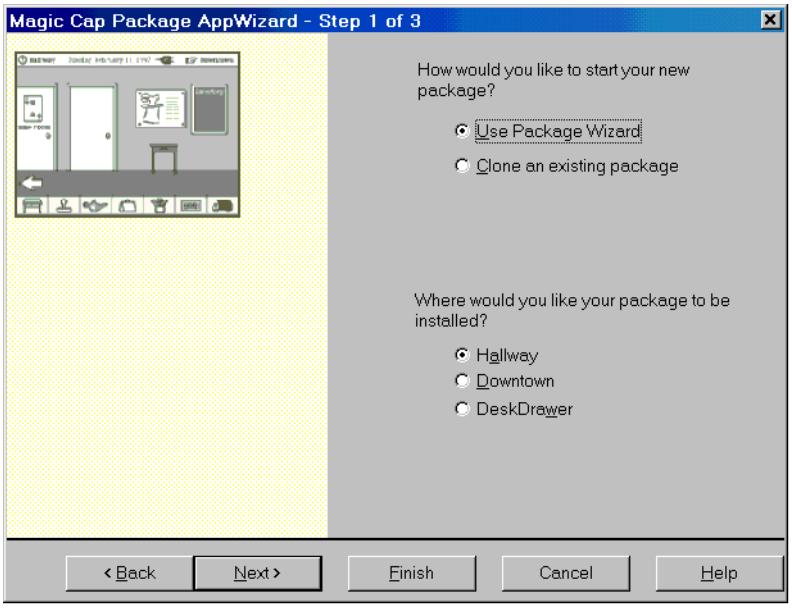
1. Launch Developer Studio and choose File▶New.

   The New dialog box appears.

2. Select the Projects tab.

3. Choose Magic Cap Package AppWizard in the left panel, then specify the name and location of the new project and click OK.

   The Magic Cap Package AppWizard appears.



4. Check Clone an existing package, then select a package and click Finish.

   The New Project Information dialog box appears.

5. Click OK.

   A new workspace opens.

Example 2-2Cloning a sample package

In this example, you will practice cloning a package. You will clone the HelloWorld package you built in Example 2-1 and name it HiWorld.

1. **Launch Developer Studio and choose** File▶New**.**

   The New dialog box appears.

2. Select the Projects tab, if it is not already selected.

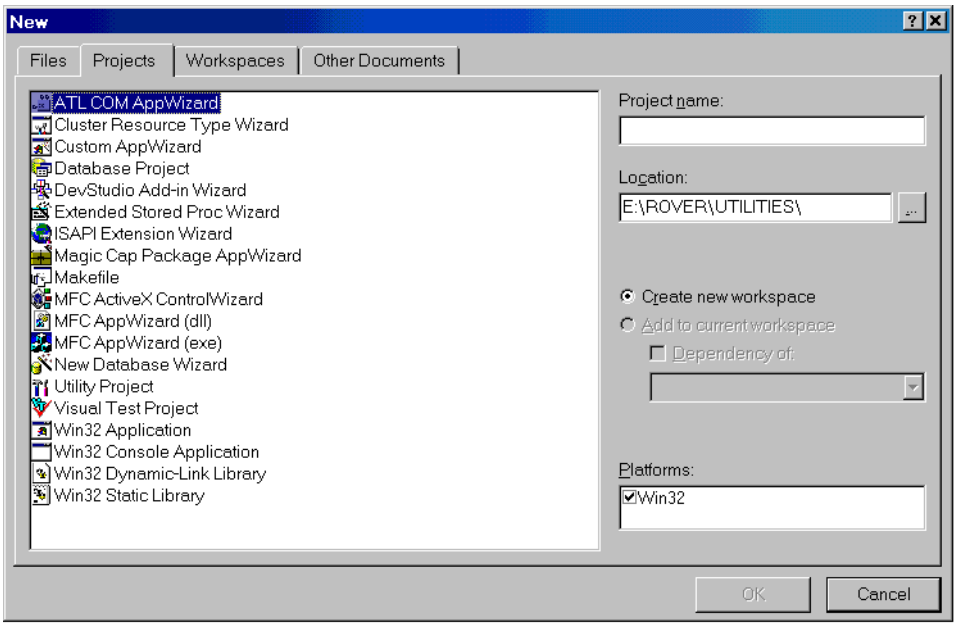


Figure 2-3 Launching the Magic Cap Package AppWizard

3. Enter the following values, then click OK:

Option	Value
`Left Panel`	**Package AppWizard**
`Project name`	**HiWorld**
`Location`	**The samples subdirectory under the directory in which Magic Developer is located.**

The Package AppWizard appears.

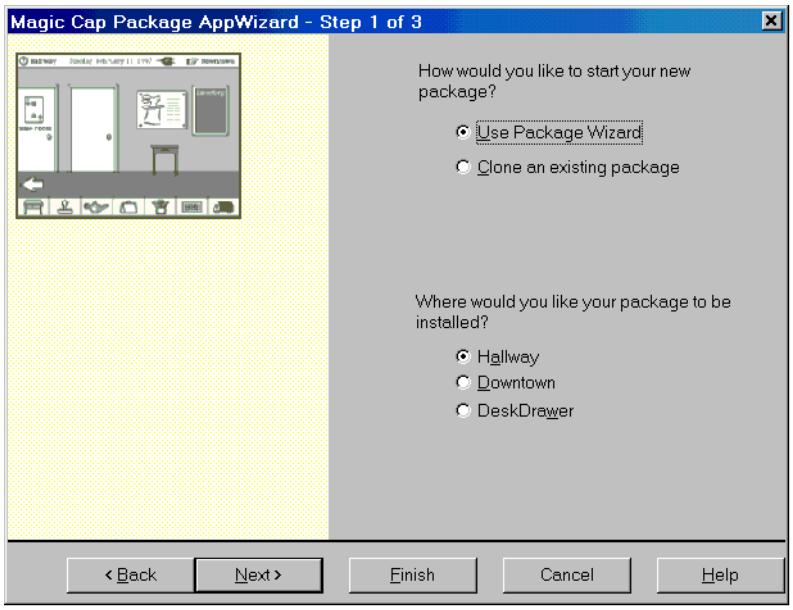


Figure 2-4 Using the Package AppWizard to clone a package

4. **Check *Clone an existing package*, then select HelloWorld in the Packages list and click Finish.**

   The New Project Information dialog box appears.

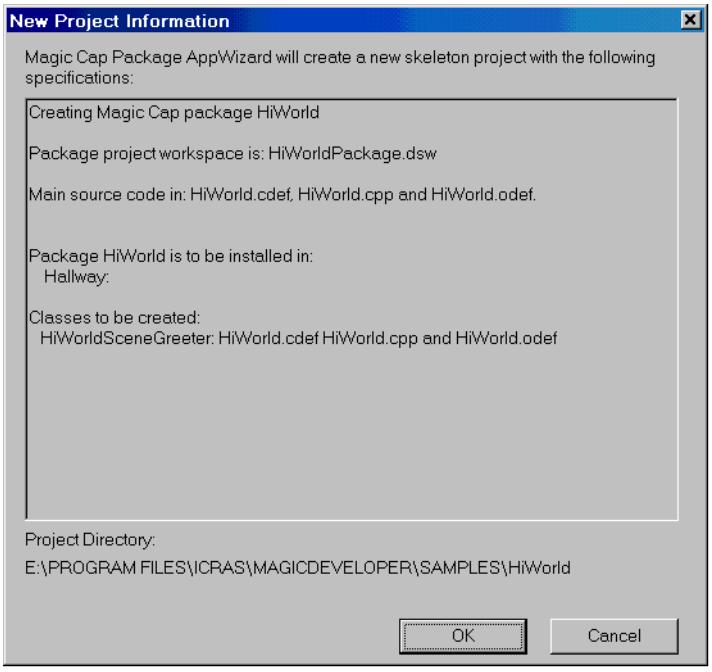


Figure 2-5 The New Project Information dialog box

5. **Click OK.**

   Magic Developer makes a clone of the HelloWorld package, gives the package the name HiWorldPackage, and stores the cloned package in the directory you selected in step 3. The HiWorldPackage workspace opens.

6. **Choose** File ▸ Close Workspace **and close the HiWorld workspace without saving it.**

   You actually use the HiWorldPackage workspace to build HiWorld, so this workspace is not necessary.

7. **Open the HiWorldpackage workspace.**

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 enable construction mode, follow these steps:

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

   The Controls window opens.

2. Click *general*.

   The General Controls window opens.

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

   The Stamper icon at the bottom of the screen changes into a Magic Hat.

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 your package, you can modify its appearance interactively in Magic Cap Simulator, and then store any changes in 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.

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. Build a package, then run it in Magic Cap Simulator.

   See “Building and running packages” on page 14 for details.

2. Turn on construction mode.

   See “Enabling construction mode” on page 20 for details.

3. Return to the package you want to modify.

   a. Hold down the *Ctrl* key and click 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 to return to the package’s scene.

4. Click the Magic Hat.

   The Magic Hat window opens.

5. Place a viewable object in your package.

   a. 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.

   b. Optionally open another drawer by clicking it, or move to another stack of drawers by clicking the arrows below the stack.

   c. Click 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 at the same time, hold down the *Ctrl* key when you click the first object. The object is pasted into the scene, but the object window remains open.

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

   See “Dumping objects and packages” on page 32 for details.

Example 2-3 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 2-2, and add a smiley face stamp.

1. **Start Developer Studio, then build and run the HiWorld package.**

   See “Building and running packages” on page 14 for details.

   The HiWorld package appears in the Magic Cap Simulator as shown earlier in Figure 2-2. Notice that the contents of the HiWorld package—including the label for the black box—are still identical to those of the HelloWorld package.

2. **Turn on construction mode.**

   See “Enabling construction mode” on page 20 for details.

3. **Return to the package you want to modify.**

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

      A list of scenes you’ve visited recently appears.

   b. Click the HiWorld scene.

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

   a. Click the Magic Hat.

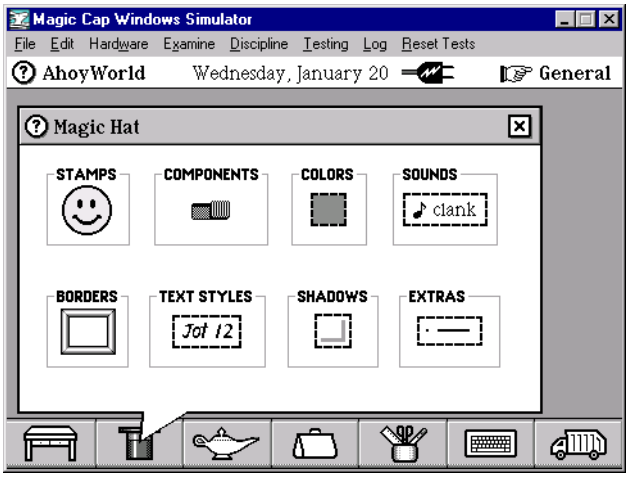


Figure 2-6 The Magic Hat window

   b. Select *stamps*.

   c. Click the smiley face stamp and slide it to the top of the black box that represents the HiWorld package.

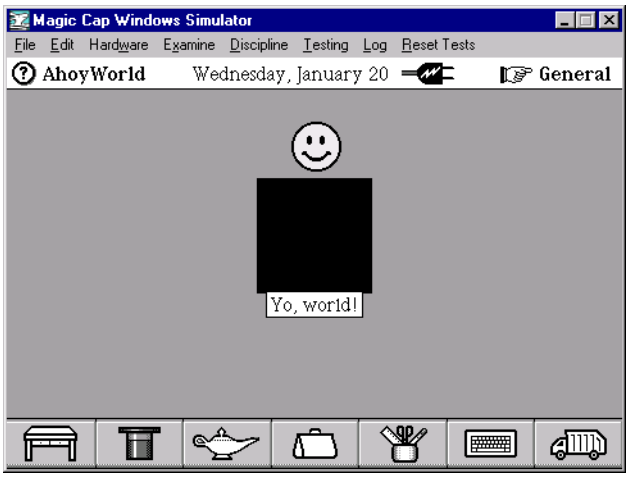


Figure 2-7 Smiley face added to the HiWorld package

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

   See "Dumping single objects" on page 33 for details.

Modifying viewable objects

Once you add objects to a package, you can use the authoring tools or coupons to modify them. In addition, you can create custom images by copying and pasting graphics from other applications.

Using the authoring tools

Magic Cap's **authoring tools** let you to move, copy, stretch, and tinker viewable objects. Use the tinker tool to display an object's label and specify its position. You can also use the tinker tool to set properties for the object, including whether it can be moved or copied.

To modify a viewable object with the authoring tools, follow these steps:

1. Build and run the package, then turn on construction mode in the simulator.

   See "Building and running packages" on page 14 and "Enabling construction mode" on page 20.

2. Click the Tool holder at the bottom of the screen.

   The Pencils Tools window opens.

3. Click the right or left arrow until you see the authoring tools in the Tools window.

4. Select the tool you want to use.

   The Tools window closes, and the selected authoring tool replaces the Tool Holder icon at the bottom of the screen.

5. Use the selected tool to modify a viewable object:
   - Click the move or stretch tool, then drag the object or its edges to move or stretch it.
   - Click the copy tool, then click the object to copy it.
   - Click the tinker tool, then click the object to set its properties as described in "Using the tinker tool" below.

Using the tinker tool

To modify an object with the tinker tool, follow these steps:

1. Click the tinker tool, then click the object you want to modify.

   The Tinker window appears.

2. Use the Tinker window to set properties for the object and to select and position a label. For example:
   - 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.
3. Close the Tinker window by clicking the close box.
4. If you added a label, use the label maker to add or change text for the label. See "Using text coupons" on page 29 for details.

Example 2-4 Modifying objects with authoring tools

In this example, you will modify the HiWorld package by changing the size of the package's black box.

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

   See "Building and running packages" on page 14 for details.

2. **Click the Tool holder at the bottom of the screen, then use the right or left arrow to display the authoring tools.**

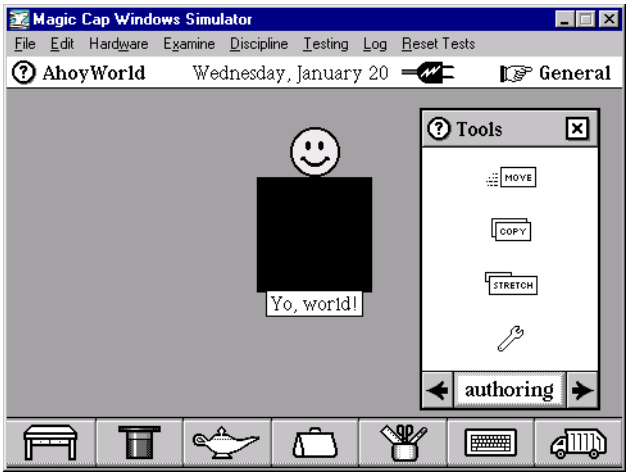


Figure 2-8 The authoring tools

3. **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.

4. **Drag the edge of the black box to resize it.**

5. **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 3.)

---

6. **Dump the modified object to the log file.**

   See “Dumping single objects” on page 33 for details.

Using coupons from the Magic Hat

Coupons from the Magic Hat let you apply the following intangible attributes 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 viewable object with coupons from the Magic Hat, follow these steps:

1. Launch Developer Studio, then build and run a package.

   See "Building and running packages" on page 14 for details.

2. Turn on construction mode.

   See "Constructing packages in the Magic Cap Simulator" on page 20 for details.

3. Click the Magic Hat.

   The Magic Hat window opens, as shown earlier in Figure 2-6.

4. Click the type of coupon you want to use.

   A window displaying coupons in the selected category opens.

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

   See the following sections for more information.

Using color coupons

When you select *colors*, the Colors window opens. 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.

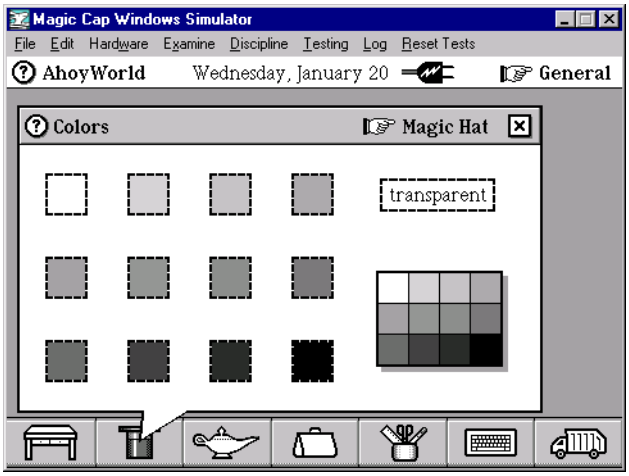


Figure 2-9Colors window

- If you select a single coupon, the Colors window closes and the color coupon you selected remains on the screen. 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. 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.

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 2-10 shows the Sounds window with the *standard* drawer open. To open another drawer, click it. To move to another stack of drawers, click the arrows below the stack.

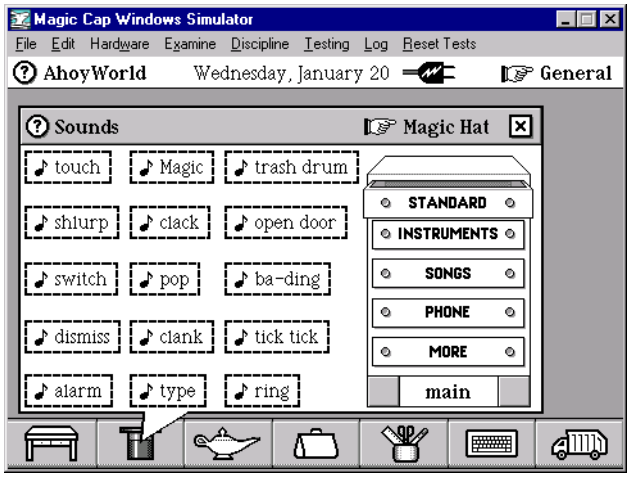


Figure 2-10 Sounds window

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

Using shadows and extras coupons

When you select *shadows* or *extras*, a window displaying the available coupons in the selected category opens. Figure 2-11 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.

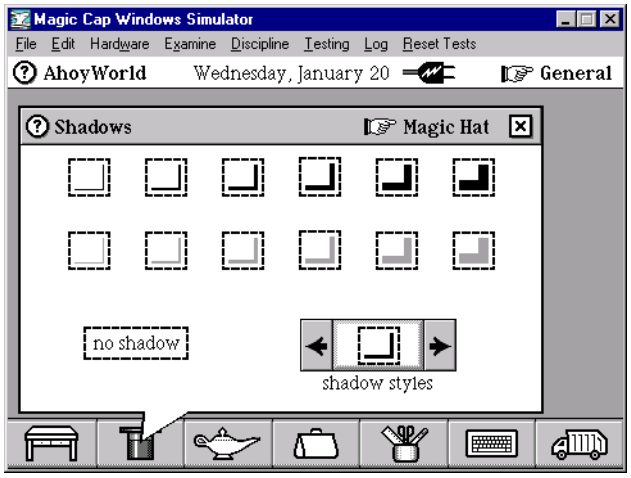


Figure 2-11 Shadows window

- If you select a single coupon, the window closes and the coupon remains on the screen. 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 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.

Using 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 a name card, for example.

To create a text coupon, follow these steps:

1. Hold down the *Ctrl* 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.

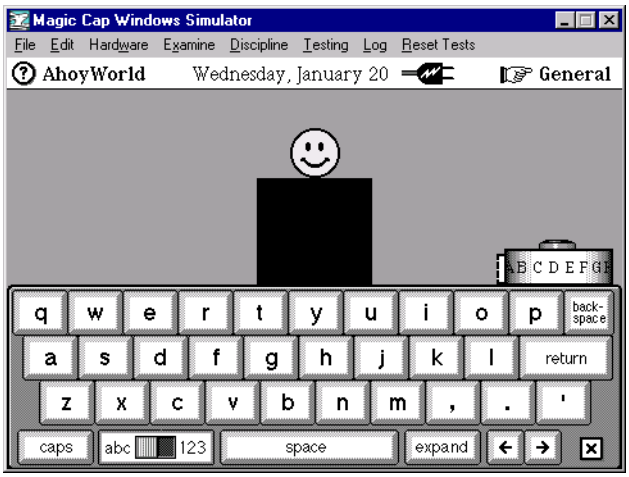


Figure 2-12 Creating a text coupon with the labelmaker

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 2-5 Modifying objects in the HiWorldPackage

In this example, you will add a label to the smiley face stamp you added to the HiWorldPackage in Example 2-3.

1. **Build and run the modified HiWorldPackage you created in Example 2-2.**

   See “Building and running packages” on page 14 for details.

2. **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 2-8.**

   The authoring tools appear only when construction mode is turned on. See “Enabling construction mode” on page 20 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 stamp above the black box.**

   The Tinker window for the smiley face opens.

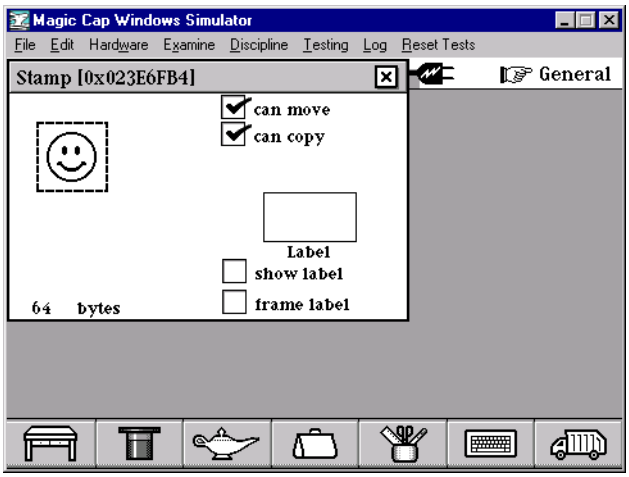


Figure 2-13Tinker window for the smiley face

6. **Check the** *show label* **and the** *frame label* **checkboxes.**

7. **Select the location for the label by sliding the word** *Label* **to the desired location on the box that represents the smiley face.**

   In this example the label will appear above the smiley face.

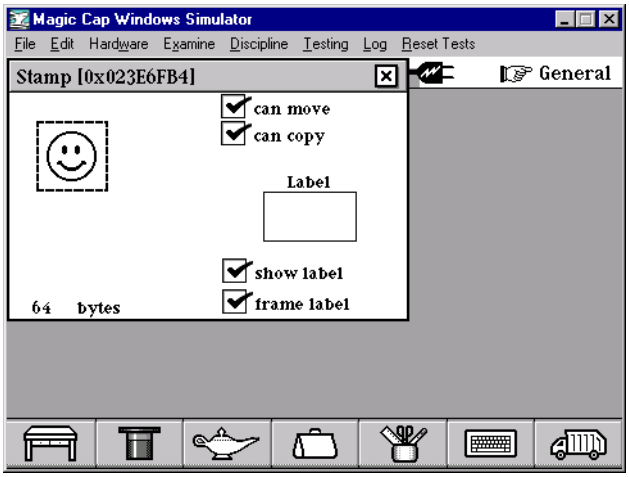


Figure 2-14 Specifying the position of the label

8. **Close the Tinker window.**

9. **Add text for the label.**

   a. Hold down the *Ctrl* 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.

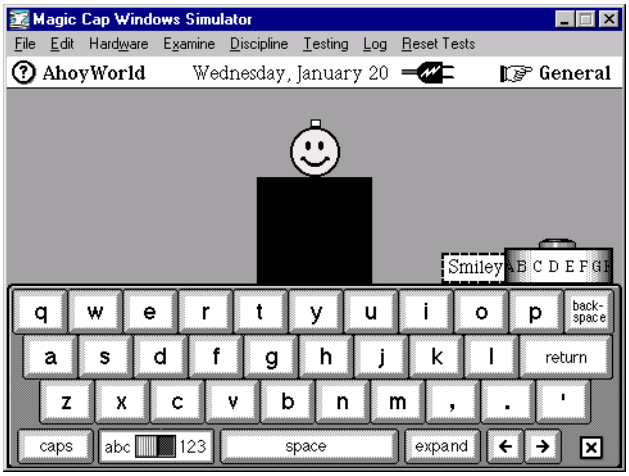


Figure 2-15 Creating a text label

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.

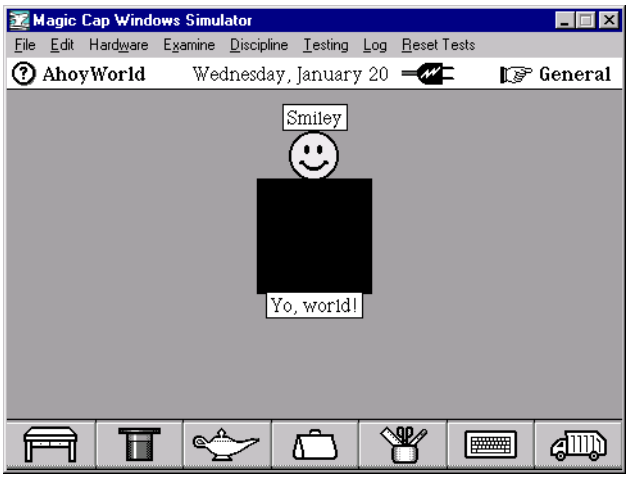


Figure 2-16 Text coupon added to the smiley face's label

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.

---

10. Dump the modified object to the log file.

See "Dumping single objects" on page 33 for details.

Dumping objects and packages

After you have constructed a package in the Magic Cap Simulator, you can 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. See "Dumping single objects" on page 33.
- Package dumping is useful when you have made wholesale changes to your package's interface. See "Dumping an entire package" on page 35.

Dumping single objects

To dump a single object from the Magic Cap Simulator and merge it into your source code, follow these steps:

1. In the Magic Cap Simulator, inspect the object that you want to dump.

   a. Choose Examine▶Show Inspector.

      The Inspector opens. This window displays a list of the current hierarchy of viewable objects, called the view list.

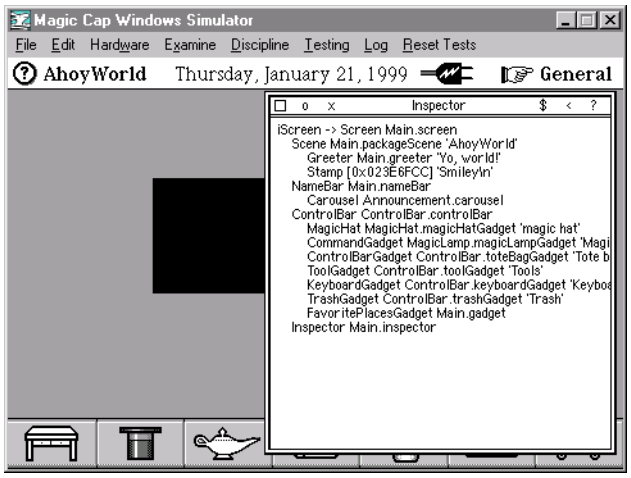


Figure 2-17 The Inspector window

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

      The Inspector minimizes.

c. Click the object that you want to dump.

The object that you clicked becomes the Inspector's target object, and the Inspector window now displays its fields. Figure 2-18 shows the Inspector window displaying the fields for the smiley face stamp that you added to the HiWorld package in Example 2-3.

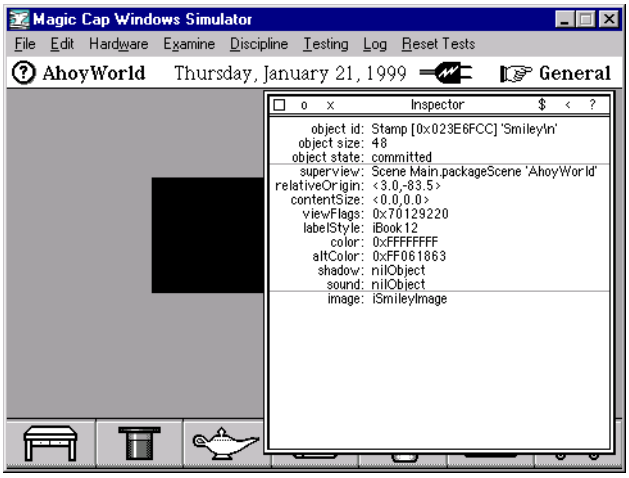


Figure 2-18 Inspector window listing the fields for the smiley face

2. Choose Examine ▸ Dump Inspector Target.

The text representation of the selected object is dumped to the log file.

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

a. In Developer Studio, click the FixUpLogFile icon on the Magic Developer toolbar.

Magic Developer opens a new window which contains a copy of the Log file.

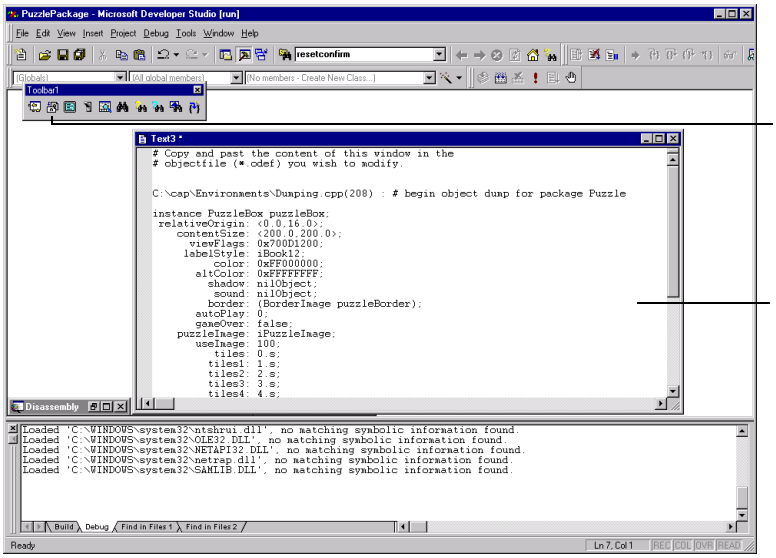


Figure 2-19 Use the Magic Developer toolbar to open the Log file

b. Copy the dumped object to the clipboard.

c. Open the `Objects.odef` file, then replace the object definition with the copy on the clipboard and save the file.

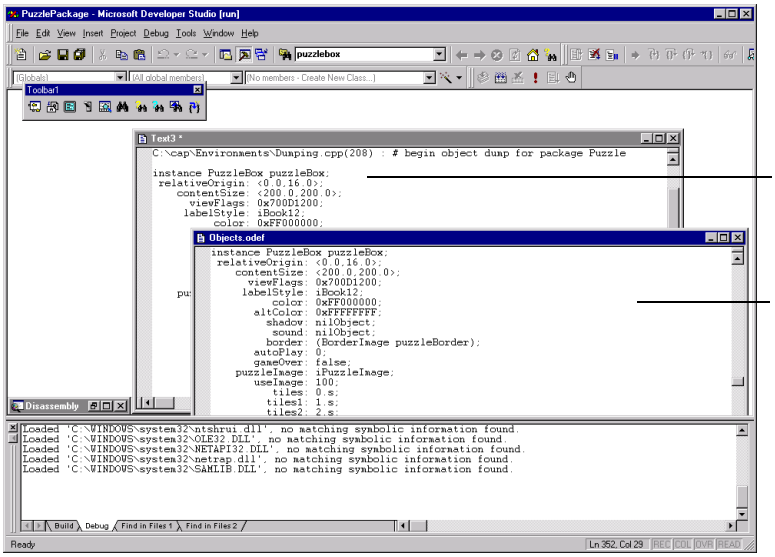


Figure 2-20 Copy object definitions from the Log file to Objects.odef

4. Save the `Objects.odef` file.

5. Build and run the modified package.

Dumping an entire package

To dump an entire package from the Magic Cap Simulator and merge it into your source code, follow these steps:

1. Dump the package from the Magic Cap Simulator.

   a. Choose Examine▶Dump Package in the Magic Cap Simulator.

      A dialog box lets you specify the location and name of a file.

   b. Specify the location and name of the dump file, then click Save.

   c. Quit the Magic Cap Simulator and return to Developer Studio.

2. Import the package into Magic Developer by doing either of the following:

   - Delete `Objects.odef` and rename the dump file to `Objects.odef`.
   - Open the dump file and `Objects.odef`, then copy each object instance from the dump file into `Objects.odef`. This technique lets you preserve any comments in your source code.

3. Save the files and rebuild the package.

Modifying the source code

Magic Cap software is constructed with conventional software development tools and some unique tools developed by Icras. 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 the Object Compiler. For the purposes of grouping or managing large numbers of objects, you may need to add instance definitions to your package.

Figure 2-21 shows an example of an instance definition.

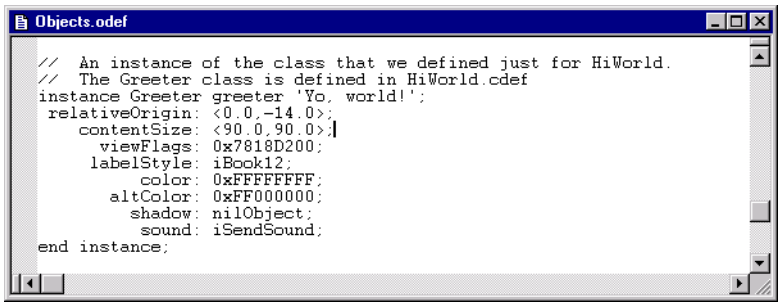


Figure 2-21 Instance definition in an Objects.odef file

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.

   See "Building and running packages" on page 14 for details.

2. Choose File▶Open, and 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 the Class Compiler. Figure 2-22 shows an example of a class definition.

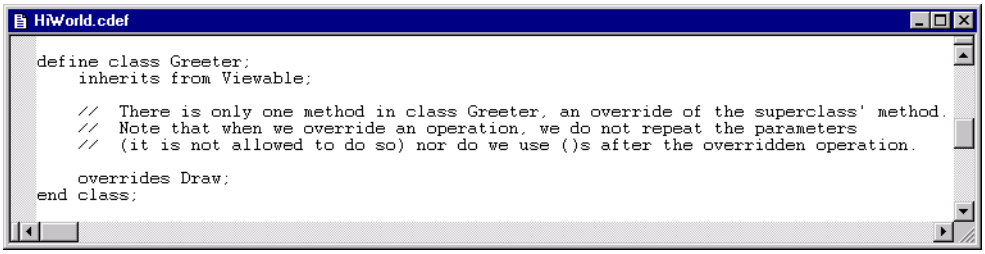


Figure 2-22Class definition in a .cdef file

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.

See Chapter 6, "Object Tools," in the *Guide to 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.

   See "Building and running packages" on page 14 for details.

2. Choose File▸Open, and 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 2-6 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 HiWorldPackage you created in Example 2-2.**
2. **Choose** File▸Open**, and open the** `Hiworld.cdef` **file.**
3. **Add a new class** `Face` **that inherits from class** `Stamp`**.**

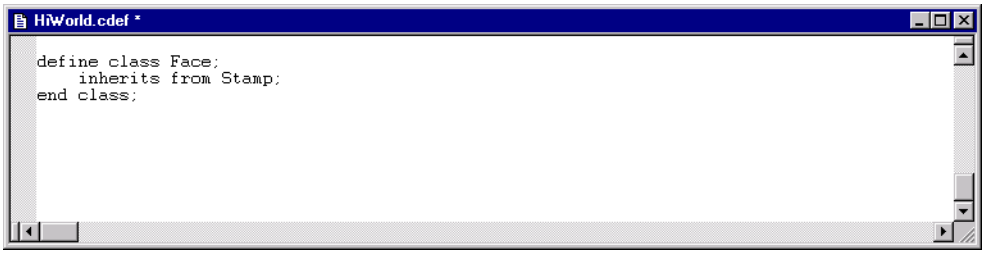


Figure 2-23 Adding a new class to the Hiworld.cdef file

4. **Save the `Hiworld.cdef` file and close it.**
5. **Modify the objects in the `Objects.odef` file to use the new classes.**
   a. Choose File▶Open, and 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`.

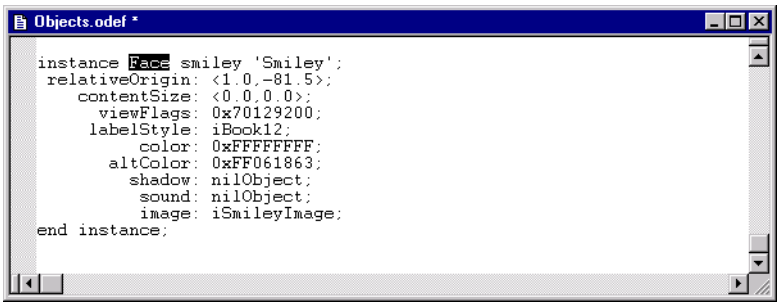


Figure 2-24 Changing the smiley face object instance definition

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. Magic Developer requires 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). The Class Compiler uses this convention to match operations to C++ functions. Of course, your source code can have functions that are not methods, and ordinary C++ rules apply to 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 2-7 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 Developer Studio.**
2. **Clone the EmptyPackage package, giving the new package the name TemperatureExample.**
3. **Build and run the TemperatureExample package.**
4. **The first time you test a package it will be necessary to establish the executable and package execution environment. You will only have to do this one time:**

   a. If you have not already done so, ensure that the default configuration for your package is "Win32 USA Debug". If this is not the default configuration, select the Build ▸ Set Active Configuration and set the default as specified.

   b. Select Project ▸ Settings and click the Debug tab.

   c. In the "Executable for debug session" text box you need to point to the Magic Cap Windows Simulator. Do so by clicking the button on the right and navigating to:

   <installation directory>\debug\win32\MagicCap-USA.exe

   where <installation directory> is where you installed MagicDeveloper.

   d. In the "Program arguments" text box you need to tell the simulator where to find your package image file. Do so by typing the following:

   /install win32\debug\usa\<package name>.package

   where <package name> is the name of your package, and then click OK. For example, assuming that you created a package named MyHelloWorld, the command line would be:

   /install win32\debug\usa\MyHelloWorld.package

   e. Select File ▸ Save Workspace to save your changes.

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

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

   The meter is located in the *choices* drawer in the Components window. See "Adding new viewable objects" on page 20 for details.

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

   See "Dumping an entire package" on page 35 for details.

8. **Tie both meters together.**

   a. Define a converter class (`TemperatureConverter`) that converts between Fahrenheit and Celsius, then add this class to the `TemperatureExample.cdef` file, as shown in Figure 2-25.

   - 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.

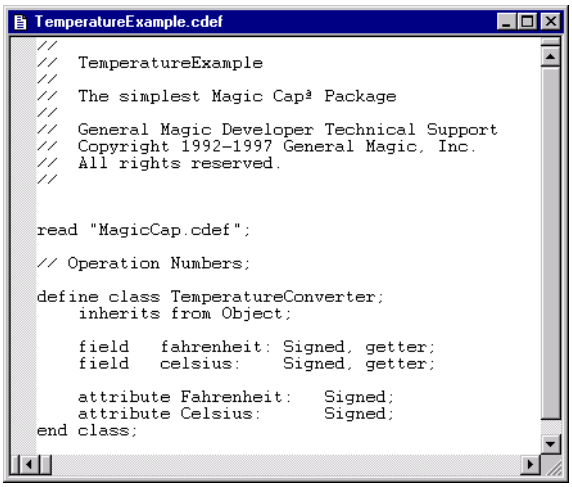


Figure 2-25 Adding class `TemperatureConverter` to the .cdef file

b. Add the two meters as subviews of the `Scene` object in the `Objects.odef` file, as shown in Figure 2-26.

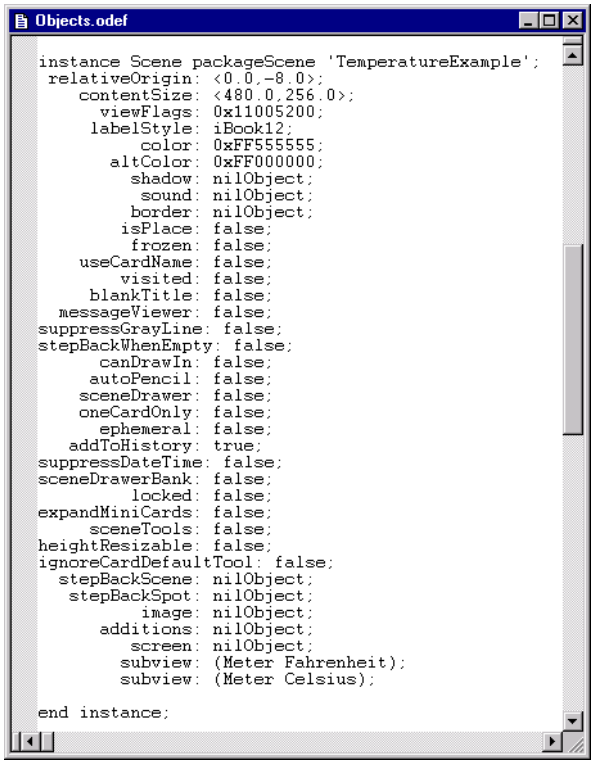


Figure 2-26 Adding the meters as subviews of scene

c. Create instance definitions for the two meters and the converter and add them to the `Objects.odef` file, as shown in Figure 2-27.

- Use the instance of 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.

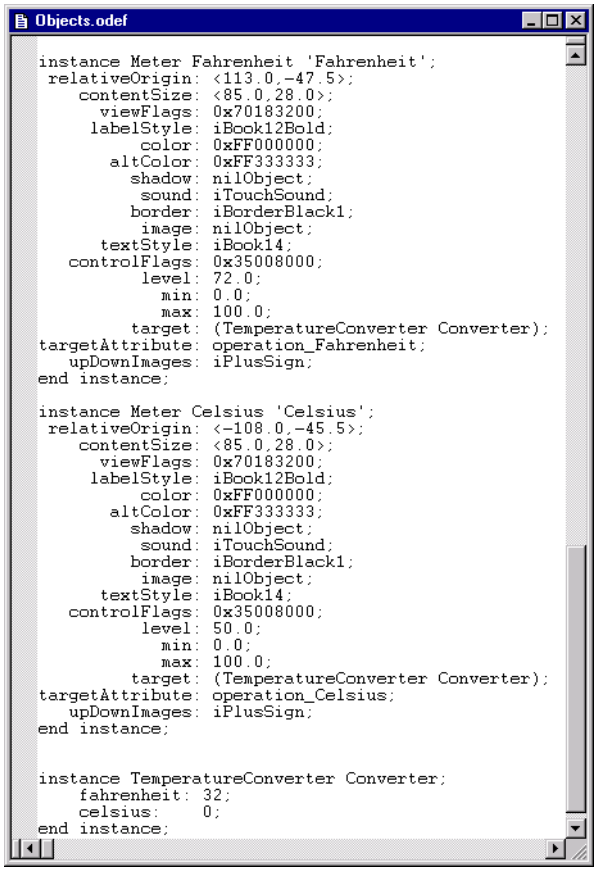


Figure 2-27 Adding object instance definitions for the meters and converter

d. Implement the methods of the new class `TemperatureConverter` in a C++ source file, as shown in Figure 2-28. 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.

TemperatureExample.cpp

```
/*-----------------------------------------------------------------------
#
#   TemperatureExample
#
#   The simplest Magic Cap® Package
#
#   General Magic Developer Technical Support
#   Copyright 1992-1997 General Magic, Inc.
#   All rights reserved.
#
-----------------------------------------------------------------------*/

/* Magic Cap System Software includes */
#include "MagicCap.h"
#include "Debug.h"

#include "TemperatureExample.xh"
#include "TemperatureExample.xph"

/* your code goes here, don't forget the CURRENTCLASS */
#undef CURRENTCLASS
#define CURRENTCLASS TemperatureConverter

Method void
TemperatureConverter_SetFahrenheit (Reference self, Signed newValue)
{
    Signed celsius;

    SetField(self, fahrenheit, newValue);
    celsius = ((newValue - 32) * 5) / 9;
    SetField(self, celsius, celsius);
}

Method void
TemperatureConverter_SetCelsius(Reference self, Signed newValue)
{
    Signed fahrenheit;

    SetField(self, celsius, newValue);
    fahrenheit = (newValue * 9) / 5 + 32;
    SetField(self, fahrenheit, fahrenheit);
}


#undef CURRENTCLASS  /* this goes at the end of the file */
```

Figure 2-28 C++ source file for the TemperatureExample package

e. Choose File▶Save to save the changes to the files.

9. Build and run the modified package.

Figure 2-29 shows the finished TemperatureExample package. When you change the temperature on either meter (by clicking - or +), the other meter will change accordingly.

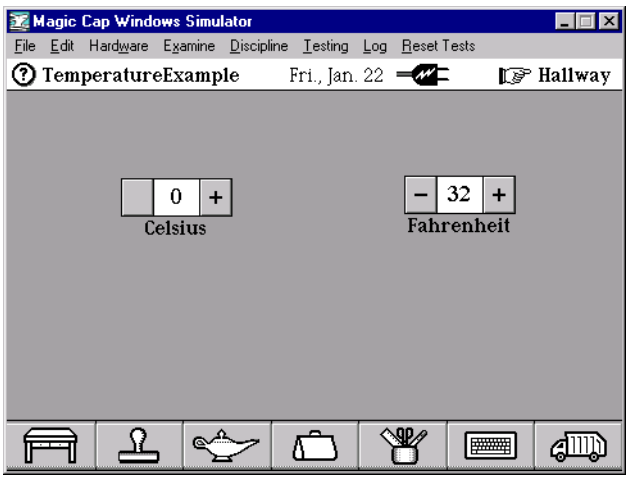


Figure 2-29 TemperatureExample Temperature Conversion package

---

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. Choose File▶Open, and open the `Objects.odef` file.
2. Attach the script to the object's instance definition.

   Use either the term `script` or the term `ScriptedMethod`. For example, 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.

   See Chapter 6, “Object Tools,” in the *Guide to 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 2-8 Adding a script to a user interface component

In this example, you will practice adding a script to a user interface component. You will add a button and a slider—a user interface component that allows users to control continuously adjustable levels—to a package.

1. **Start Developer Studio.**
2. **Clone the EmptyPackage package, giving the new package the name SimpleControl.**
3. **Build and run the SimpleControl package.**
4. **The first time you test a package it will be necessary to establish the executable and package execution environment. You will only have to do this one time:**

   a. If you have not already done so, ensure that the default configuration for your package is "Win32 USA Debug". If this is not the default configuration, select the Build ▸ Set Active Configuration and set the default as specified.

   b. Select Project ▸ Settings and click the Debug tab.

   c. In the "Executable for debug session" text box you need to point to the Magic Cap Windows Simulator. Do so by clicking the button on the right and navigating to:

      <installation directory>\debug\win32\MagicCap-USA.exe

      where <installation directory> is where you installed MagicDeveloper.

d. In the "Program arguments" text box you need to tell the simulator where to find your package image file. Do so by typing the following:

/install win32\debug\usa\<package name>.package

where <package name> is the name of your package, and then click OK. For example, assuming that you created a package named MyHelloWorld, the command line would be:

/install win32\debug\usa\MyHelloWorld.package

e. Select File ▸ Save Workspace to save your changes.

5. In the Magic Cap Simulator, enter construction mode.

6. Add a button and a slider to the SimpleControl package.

a. Click the Magic Hat.

b. Select *components*.

The Components window opens.

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

d. Click the *choices* drawer to open it.

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. See “Using text coupons” on page 29 for details.

h. Tinker the slider to turn its label off. See “Using the tinker tool” on page 24.

Figure 2-30 shows the button and slider added to the SimpleControl package.

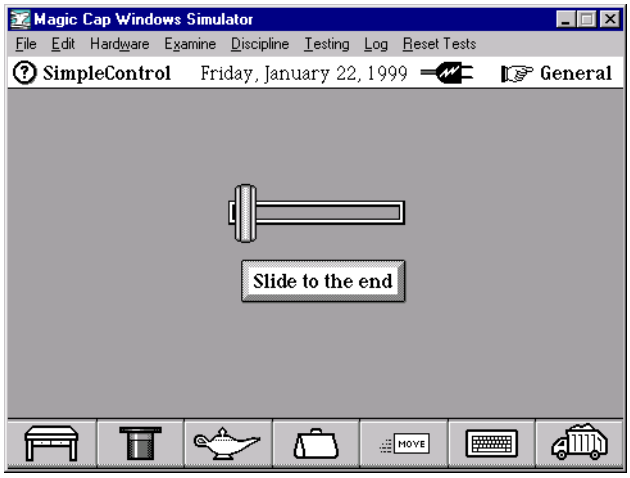


Figure 2-30 Button and slider added to the SimpleControl package

7. **Dump the package back into Magic Developer and merge it with your source code.**

   See “Dumping an entire package” on page 35 for details.

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

   a. Choose File▶Open, and 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 2-31. This script adds an action affecting the slider to the button—the slider will slide to the right end when the button is pushed.

   c. Name the `Button` instance `pushbutton` and the `Slider` instance `slider`, then change references to these instances in the `subview` fields of the `packageScene` instance.

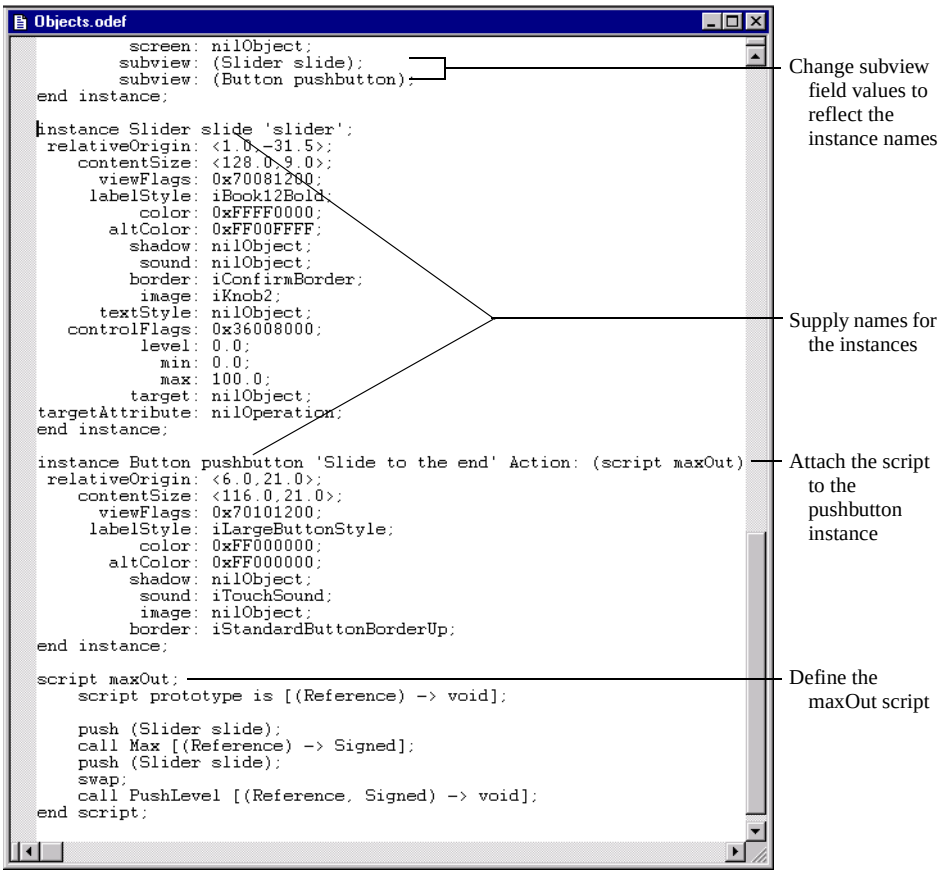


Figure 2-31 Modified Objects.odef file

9. 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. In order to use Bowser Jo, you must have a Java-enabled web browser installed on your system.

To search with Bowser Jo, follow these steps:

1. Launch your web browser.
2. Open the file `index.html` in the `docs\htmlhelp` subdirectory of the directory in which Magic Developer is installed.

   Your web browser loads an HTML page for Bowser Jo that displays “Magic Cap Alphabetical Class Index ‘A’”.

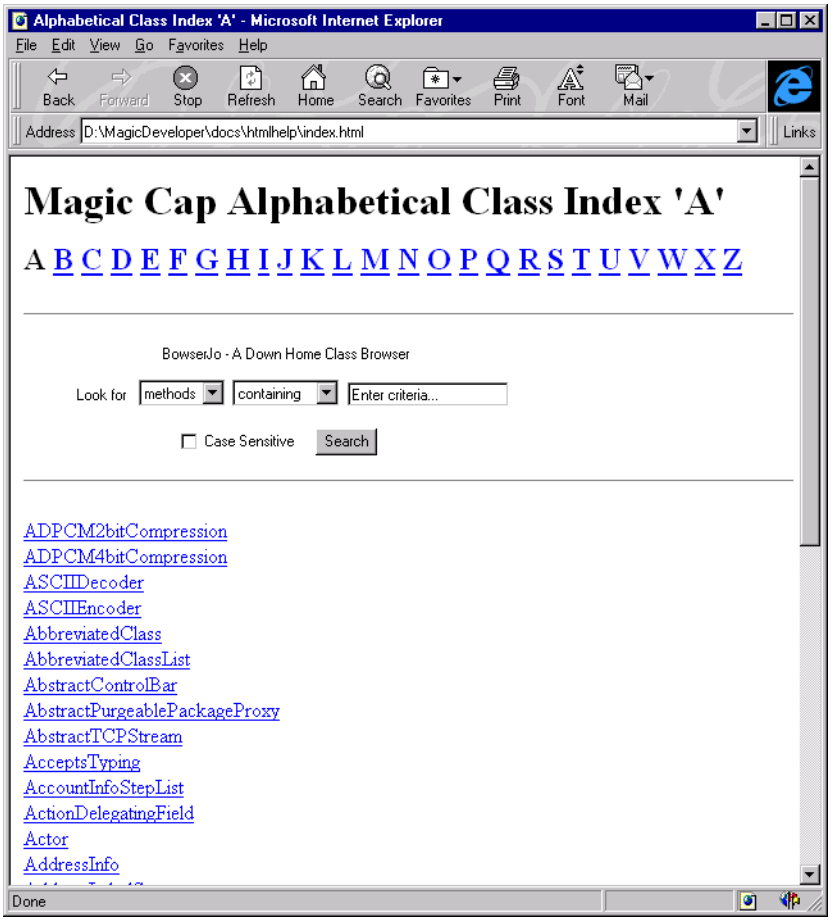


Figure 2-32 Bowser Jo home page

3. Use Bowser Jo to search either alphabetically or by string matching:

- To search alphabetically, click the letter of the alphabet corresponding to the class, then click the class and the element within the class for which you want information.
- To search by string matching:

  a Select *methods*, *fields*, or *classes* in the first drop-down list.

  b Select *containing*, *equaling*, or *starting with* in the second drop-down list.

  c Type the string for which you want to search in the *Enter criteria* field.

  d Optionally check *Case Sensitive* to perform a case-sensitive search.

  Figure 2-33 shows the Bowser Jo window set up to search for classes containing the string `button`.

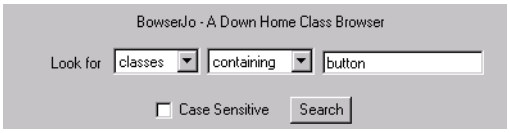


Figure 2-33 Bowser Jo set up to search for the string button

  e Click *Search*.

  Bowser Jo returns a list of methods meeting the criteria you specified.

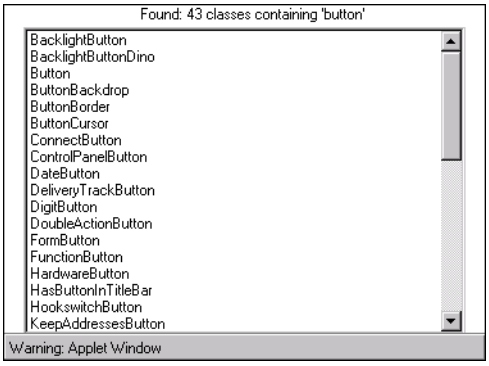


Figure 2-34 Bowser Jo search results window

f Display reference information by double-clicking the name in the search results window.

Bowser Jo displays the information you specified.

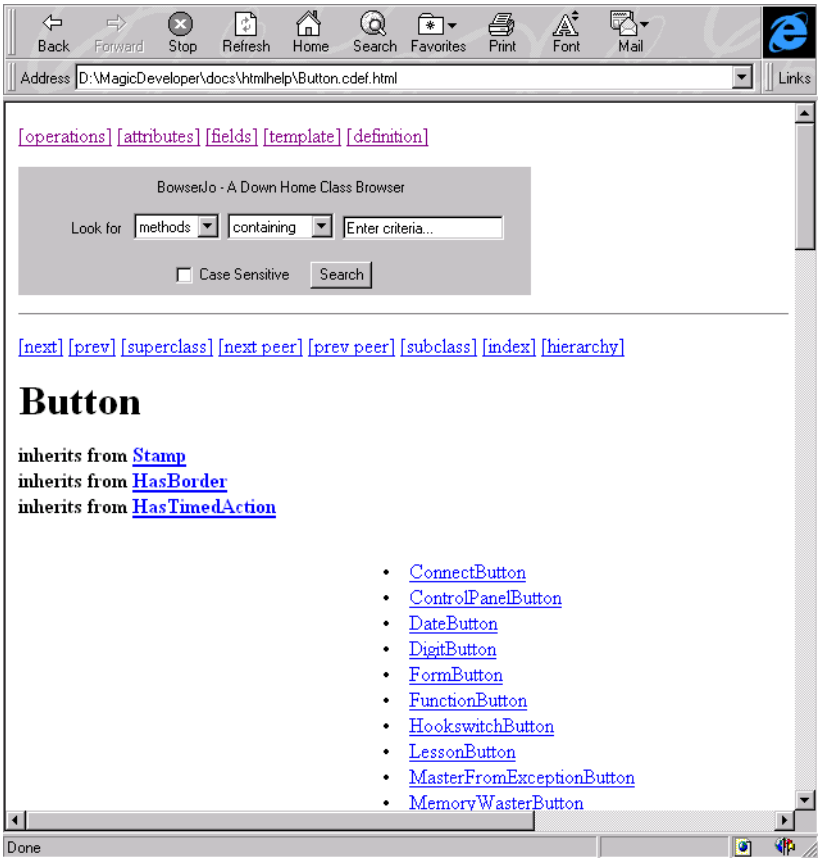


Figure 2-35 Bowser Jo displaying information about the Button class

Figure 2-35 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.

Example 2-9 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). See "Searching with Bowser Jo" on page 49 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). See "Searching with Bowser Jo" on page 49 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). See "Searching with Bowser Jo" on page 49 for details.
   b. Notice that `Button` is a superclass of `SimpleActionButton`.

c. Scroll down to the fields list. The last two fields—`target` and `operation`—belong only to class `SimpleActionButton`, not to any of its superclasses.

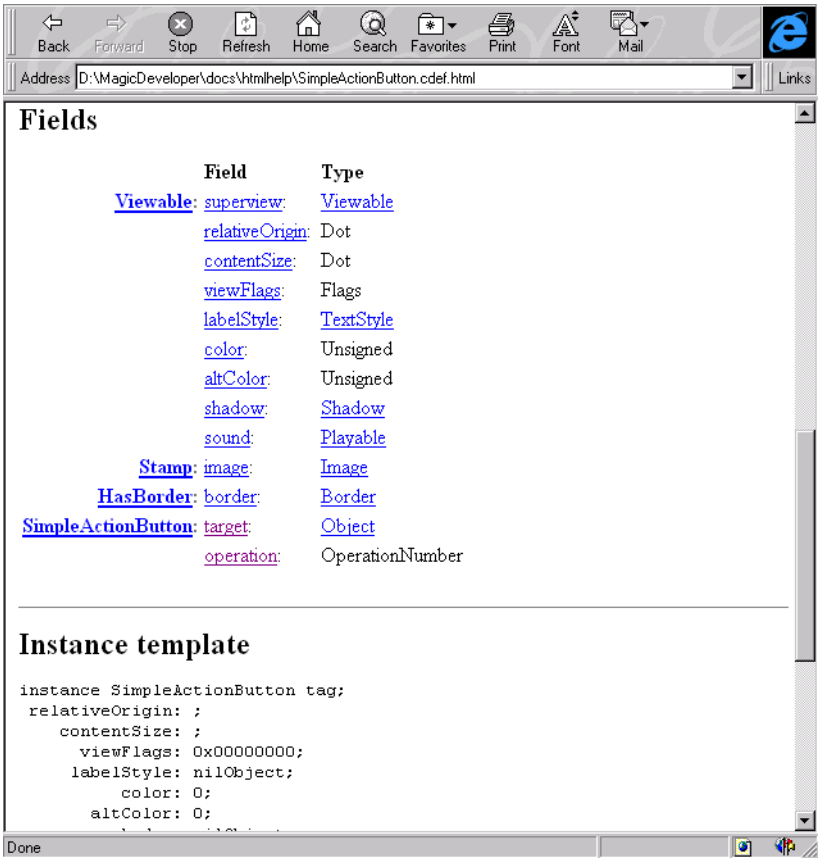


Figure 2-36 Field list for class SimpleActionButton

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). See "Searching with Bowser Jo" on page 49 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). See "Searching with Bowser Jo" on page 49 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). See "Searching with Bowser Jo" on page 49 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

Icras, Inc. 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 2-10 illustrates the basic steps for localizing the text in a package. See Chapter 7, "Package Localization," in the *Guide to Development Tools* for information about localization tools and localization files and complete instructions for localizing packages.

Example 2-10 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.

1. **Launch Developer Studio and open the workspace file for the HiWorld package.**

2. **Choose** Build▸Set Active Configuration**, then specify** *Win32 Japan Debug***.**

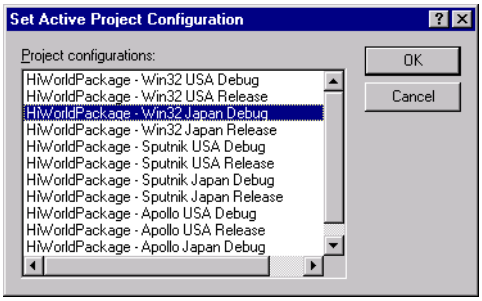


Figure 2-37 The Set Active Configuration dialog box

3. **Choose** File▸Open**, then open the phrase file for Japan,** `Japan.Package.Phrases`**.**

4. **Place a** `//` **comment before the following line:**

```
dont require phrases for textual fields
```

5. **Build the package by choosing** Build▸Build**, or by pressing** ***F7*****.**

   Developer Studio displays an error for each missing phrase in the Build window.

6. Update the phrase file by copying the replacement strings from the Build window and pasting them into the `Japan.Package.Phrases` file.

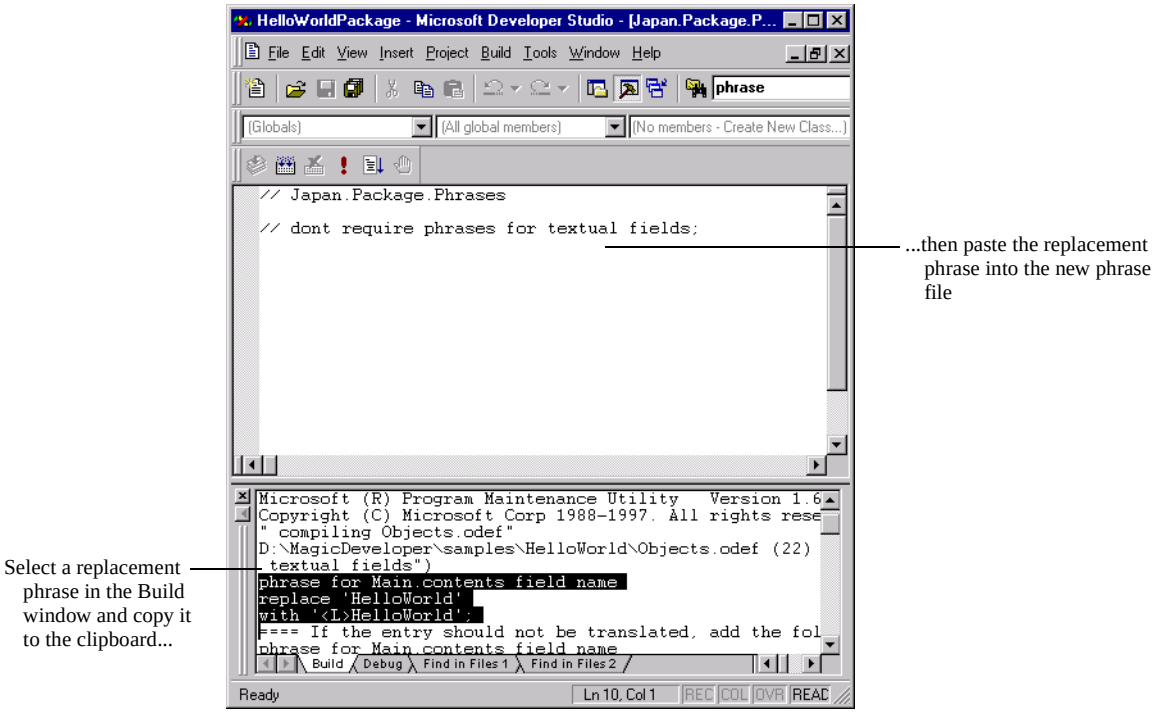


Figure 2-38 Updating the phrase file

When you are finished, the new phrase file should look similar to the following:

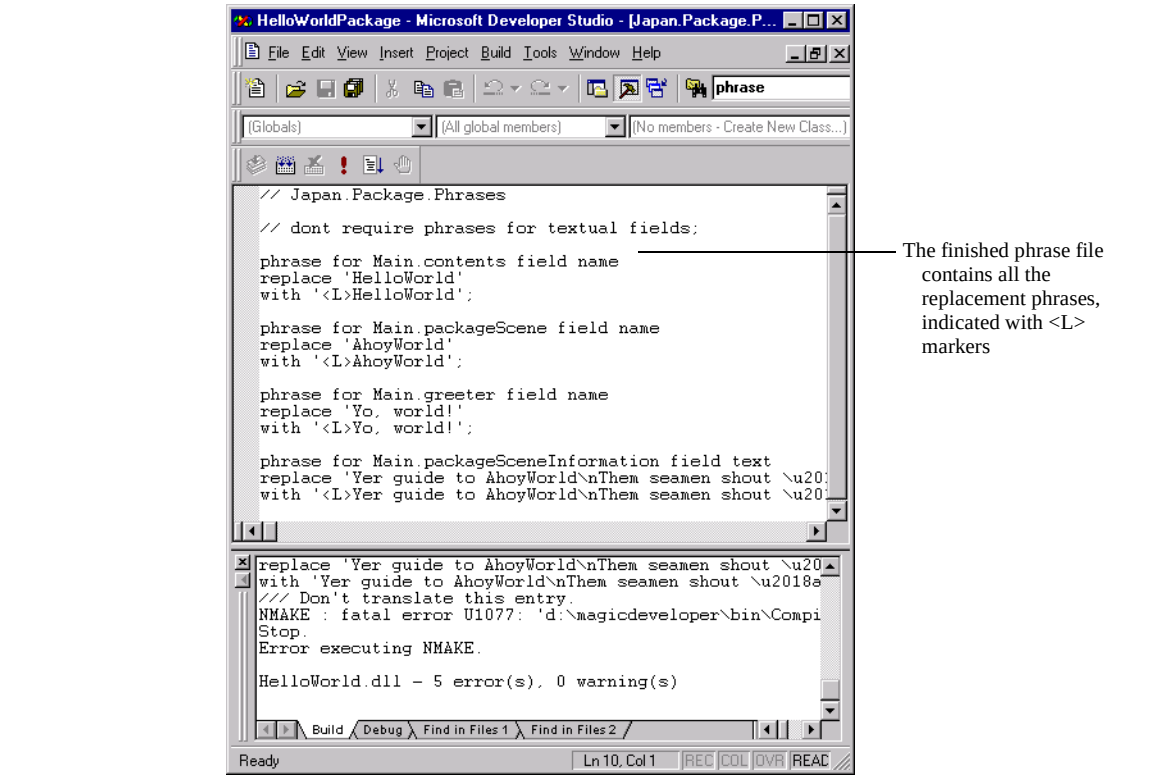


Figure 2-39 The updated phrase file

7. **Build and run the package.**
8. **If you have correctly replaced every phrase, Developer Studio displays no more errors. In the simulator, you'll see the** `<L>` **strings displayed in the application. These** `<L>` **strings indicate where the translated text will appear.**

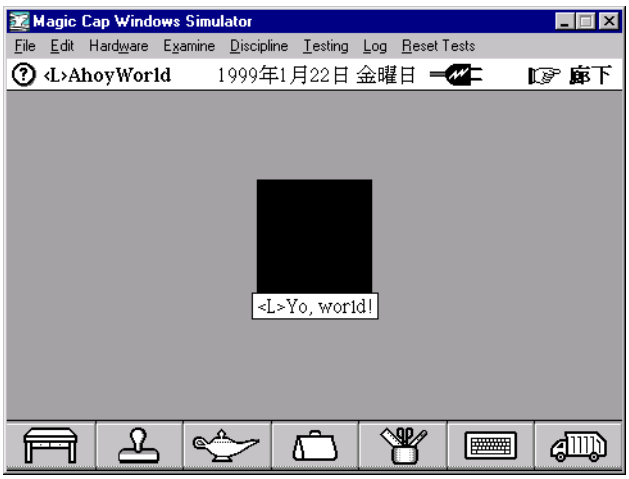


Figure 2-40 The preliminary package localized for Japan

9. **Translate the strings marked with an** `<L>` **in** `Japan.Package.Phrases`**.**
10. **Convert the translated strings into Unicode.**
11. **Build the localized package with the translated phrase file.**

Index

P

R

S

T

V