Stored Images

User's Guide & Cookbook

The purpose of this document is to give you a comprehensive overview of the features and facilities provided by *Stored Images* and how to access them programmatically. It starts with an introductory chapter. Subsequently step-by-step instructions show how to build applications employing new data set widgets.

Contents

Introduction

“Stored Images” provide you with a convenient way to incorporate images and icons stored in bitmap files on disk into your applications. An instance of *StoredImage* acts as a proxy for a real image which is to be loaded from a file and converted into a legal VisualWorks image as soon as it’s needed (accessed) the first time. This means you can use an instance of *StoredImage* in each and every place where you would use a normal instance of class Image.

You create a new instance of *StoredImage* by giving a filename from where the image is to be loaded:

```
myImage := StoredImage on: 'C:\Images\Magnifier.gif'
```

This creates the new instance but doesn't do anything else. The image isn’t loaded at that time. This happens only when the image receives a message, which is intended to go to the real image and not to the proxy itself; such as this one:

```
myImage displayOn: Window currentWindow graphicsContext.
```

This leads to the image being loaded from file and converted into a legal VisualWorks image, more precisely, into an instance of *CachedImage*. The real image is stored in an instance variable of the StoredImage, providing a form of caching mechanism. After that, in succeeding accesses, there’s no need to reload it from file, since it’s already cached in memory. All subsequent image specific messages are forwarded to the real image. However, you can flush and force a stored images’s target image to be reloaded by sending message *flush* to the *StoredImage* instance, or you can flush all image caches by sending *flushImageCache* to class StoredImage.

Programming Interface

Since an instance of *StoredImage* acts as a proxy for real images, the programming interface of stored images corresponds to that of class *Image* (and subclasses). The only messages specific to stored images are messages to flush image caches (see above) and class messages to create new instances of *StoredImage*. The primary instance creation method is *StoredI-*

mage>>on: aFilename, which configures a new instance to read the image data from the specified file on demand. Further class messages allow to specify search paths used to look up image files.

If you load images from a bitmap file that includes transparency information, and you have an according ImageReader, images loaded via a *StoredImage* proxy will also be images with transparent background. A subclass of *StoredImage*, called *StoredOpaqueImage*, is provided for ImageReaders/file formats that don’t support transparency information. This class allows you to load and construct an instance of *OpaqueImage* from two different files. One containing the image’s figure the other containing its (1 bit plane) mask.

Class *StoredImage* maintains a search path to image files on disk. This allows you to omit long path names, but rather specify the relative name of an image file, provided the referred file resides within a directory registered in the search path.

Using Stored Images in Canvas Painter

Package *"Stored Images"* includes minor changes to class *ApplicationModel*, that enables you to enter filenames of images stored on disk in the selector field of a label's or button's properties page. You do this by simply preceding the filename you enter with a ‘@’-character. This tells the changed method in ApplicationModel to use the *StoredImage* machinery to load the referred image from disk. If you put in two ‘@’-characters, *StoredOpaqueImage* is deployed.

The changes to class *ApplicationModel* introduced with stored images are essentially enhancements to the built-in management facility for visuals. In particular ApplicationModel’s class messages *visualAt:ifAbsent:* is enhanced to look up stored images if the leading character of a visual’s selector is the ‘@’-character. When a stored image is loaded using the visual key approach, it is cached within class *ApplicationModel’s* visuals registry (compare VisualWork Cookbook, Chapter 5, Section “Building a Registry of Labels”).

VisualWorks Image Readers

“Stored Images” itself does not provide any functionality for loading images from bitmap files. It rather depends on an external *ImageReader* package being installed. Image readers are foreseen in the VisualWorks class library by means of the abstract base class *ImageReader*. However, there’s only one specific image reader shipped with VisualWorks (2.5), to read Windows BMP image files.

You may want to watch out for additional image readers, in particular for GIF files. The GIF image format is interesting not only because it’s omnipresent in the web, but also because it has the built-in feature to store transparency information.

In case you are restricted to image readers for formats which don't support transparency (such as BMP), class *StoredOpaqueImage* provides a way to construct images with transparent background from two single image files, one serving as the image's figure, the other serving as it's shape.

Summary of Features

- Stored Images seamlessly provide facilities to load images from bitmap files on disk.
- Due to the proxy approach, deployment of stored images is completely transparent. An application principally doesn't have to be aware that it uses stored images.
- Minor enhancements to class *ApplicationModel* provide a convenient and transparent way to deploy stored images from within Canvas Painter.
- Advanced caching mechanisms reduce the effort and expense of image loading, gratefully accelerating applications using images stored on disk. This feature also works in a completely transparent way.

Cookbook:
Deploying Stored Images

In this chapter we will describe how to deploy *Stored Images* in a VisualWorks application. We assume that image readers for GIF files have been installed.

Setting up image search paths

Strategy

In order to ease the use of stored images, you can configure search paths with class *StoredImage*. Whenever an image is to be loaded, it is looked up in the search paths.

Basic Steps

1. Send message *searchPath:* to the *StoredImage*. The methods expects a single String or Filename or a collection of Strings or Filenames as the parameter. This is converted internally to an ordered collection.

```
StoredImage searchPath: 'V:\etc\images'.
StoredImage searchPath: #('C:\Nt' 'V:\etc\images').
```

2. If you also want to provide a search path for looking up shapes, send *shapesPath:* to the *StoredOpaqueImage*.

```
StoredOpaqueImage shapesPath: 'V:\etc\shapes'.
```

Variant A: Expanding the search path

You can add a path to the search paths by something like

```
StoredImage searchPath add: 'D:\ images'.
```

Deploying Stored Images programmatically

Strategy

An instance of *StoredImage* acts as a proxy for an image stored in a bitmap file on disk. The image is automatically loaded on demand. This is, when a message is being sent to the proxy which is intended to go to the image itself.

Basic Steps

1. Create an instance of *StoredImage* as a proxy for a file on disk with instance creation method *on:* or variants. The image isn't loaded at that time. Only the filename is looked up in the search paths and registered with the instance.

```
myImage := StoredImage on: 'myImage.bmp'.
```

2. When a method intended to go to the real image is sent to the proxy, the image is loaded from disk and the message is forwarded to the image.

```
myImage displayOn: aGraphicsContext.
```

3. You can release a *StoredImage's* previously loaded real image, by sending *flush* to an instance. Alternatively, you can release and unload all images loaded by existing instances of StoredImage by sending *flushImageCache* to the class.

```
myImage flush.
StoredImage flushImageCache.
```

Analysis

Due to the proxy pattern, *StoredImages* provide essentially the same interface as instances of class *Image* and subclasses. This means that you can use stored images in each and every place where you would use instances of *Image*. Examples are images held in instance variables, in class variables, or in *ApplicationModel's* visuals registry.

You can install instances of *StoredImages* permanently in your image without concern that you may either end up with lot of image loading or with a lot of memory wasted by loaded images. On one hand, once loaded, instances of StoredImage retain their images, thus acting as an image cache. On the other hand, you can selectively unload loaded images by means of the flush messages.

Deploying Stored Images in Canvas Painter

Strategy

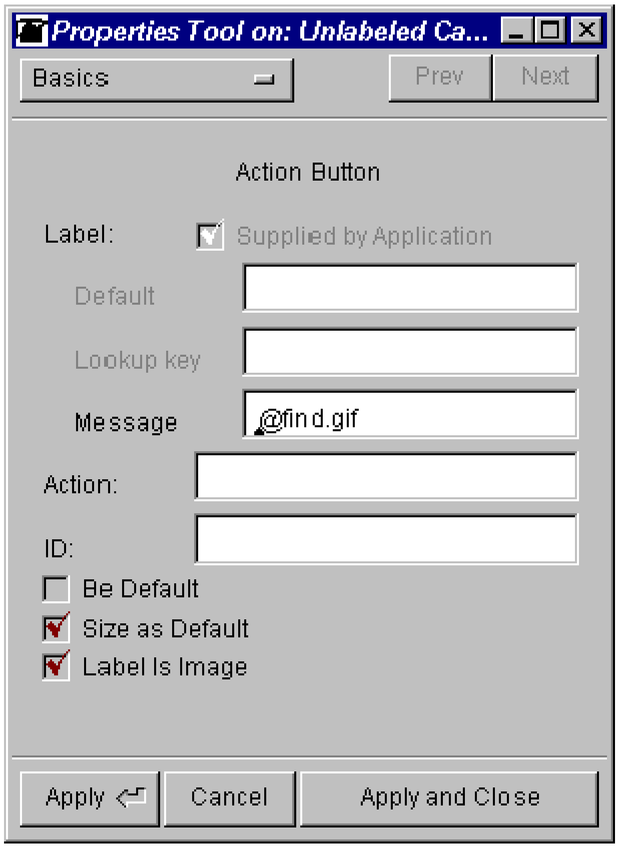
The enhancements to class *ApplicationModel* provided with Stored Images, allows you to refer to images from within the properties page of a label or a button in Canvas Painter.

Basic Steps

1. Add a button to a canvas, select it and open the „Basics“ properties page.

2. Turn on check box „**Label is Image**“

3. Enter the filename of the image you want to appear as the buttons label into the **Message** aspect, preceding the filename with a **‘@’** character.



Label: [✓] Supplied by Application

Default

Lookup key

Message @find.gif

Action:

ID:

[ ] Be Default

[✓] Size as Default

[✓] Label Is Image

4. Ensure that you enter a valid filename. This is either an absolute path, or the name of a bitmap file, residing in one of the directories registered as *StoredImage’s* search path.

Variant

Provide for a *StoredOpaqueImage* to be created by preceding the filename with **‘@@’**. In this case you must have configured both, a search path and a shapes path. Two different bitmap files, a figure and a shape file, are expected to exist in these two search paths.

Analysis

When an instance of *StoredImage* is created as result to a @-reference from within an application’s window spec, it is registered in class ApplicationModel’s visuals registry (*ApplicationModel class>>visualAt:put:*). This ensures that the image won’t be reloaded each time the application is restarted. However, you may want to remove these entries from the visuals registry when your application is closed. Do so by something like ...

```
ApplicationModel visuals removeKey:#’@find.gif’
```