NeXTSTEP kits extend the concept of object-oriented programming to cover broad functional areas. Each kit supplied with NeXTSTEP consists of a set of classes developed together to amplify your programming efforts in a particular area, by providing a complete framework for you to build on.

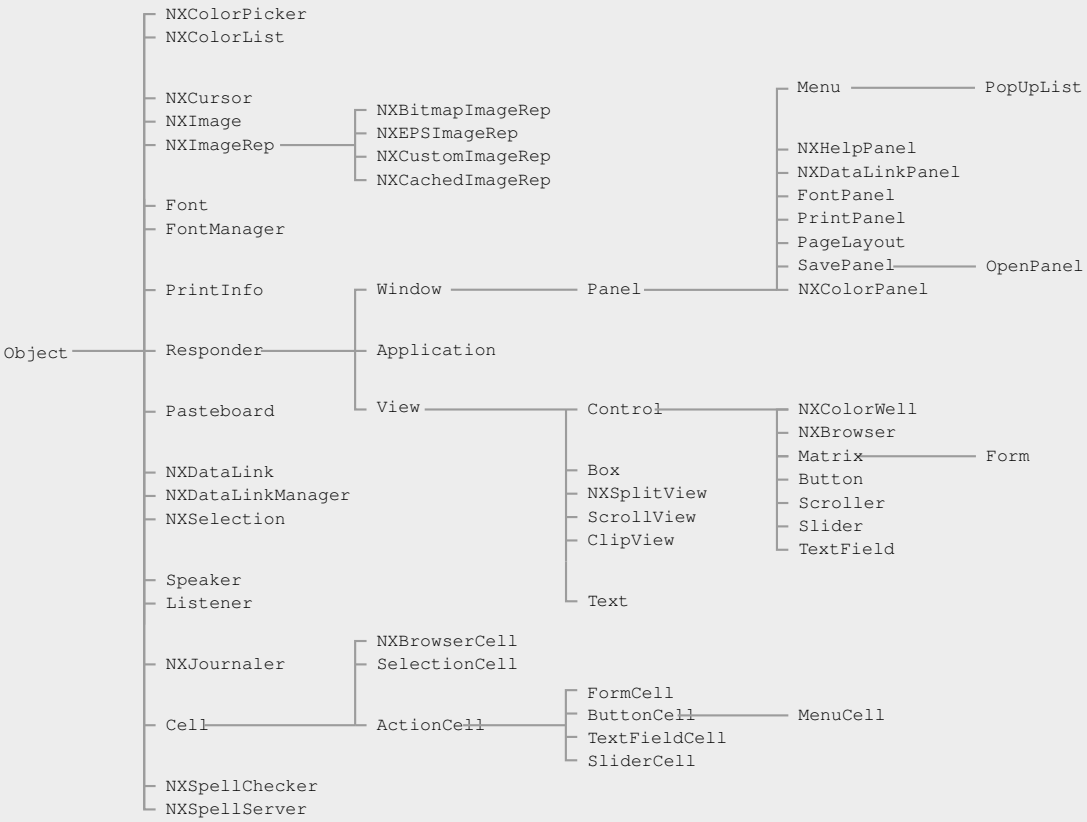
So far, this discussion has focused on a handful of the classes that NeXTSTEP has to offer. This section introduces the broad spectrum of kits and classes provided with NeXTSTEP.

Each of the sections that follows focuses on a particular NeXTSTEP kit. These discussions begin with an overview and class hierarchy of the kit in question. Subsequent pages highlight the key classes in the kit and how the functionality they provide can enhance the applications you develop.

THE APPLICATION KIT

The Application Kit provides the basic framework of a NeXTSTEP application. This kit includes the Application, Window, and View classes that are the core components of every NeXTSTEP application. It also offers variety of classes that extend these classes to ensure greater consistency between NeXTSTEP applications.

APPLICATION KIT CLASSES



Windows

Every application has windows—rectangular areas where information is presented on the screen. The Window class provides an object-oriented interface between the process that supplies the windows (the Window Server) and your application. Each Window object manages a single window; your application affects its windows by sending messages to their corresponding Window objects.

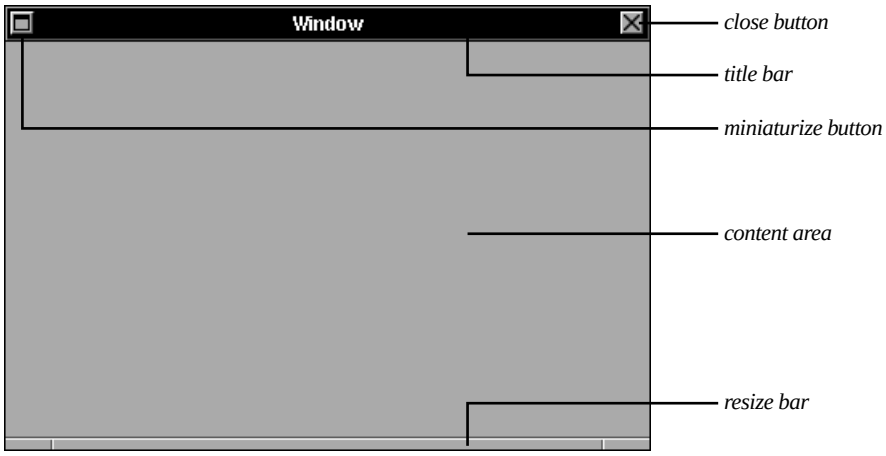
Each window has a content area, where Views handle user-generated events (such as mouse clicks) and draw images. When the user types or uses the mouse within the content area, the Window object gets the event and then sends it to the appropriate View.

Besides its content area, a Window object can also supply the following:

- A title bar, which displays a title and lets the user drag the window to a new position
- Buttons in the title bar, which let the user miniaturize or close the window
- A resize bar, which the user can drag to change the window's size and shape

Project Builder provides one window for every new application. You can drag additional windows into your application from Interface Builder's Palettes window.

A window



Controls

The Application Kit provides many classes to help you put controls in your application. Controls are Views that translate user actions, such as mouse clicks, into application-specific messages for other objects. For example, clicking the Retrieve Call button in Your Call sends a retrieveCall: message to the CallController object.

The Application Kit provides the following controls: buttons, sliders, browsers, forms, text fields, scrollers, and color wells. All controls are implemented by subclasses of the Control class.

Buttons are the primary controls for setting a state or initiating an action. They can assume a variety of shapes and sizes, such as the switches and radio buttons shown below. A button has the following features:

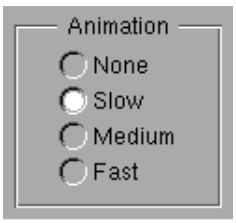
- It has a label, which can include both text and an icon.
- It's highlighted or pushed in (or both) while the user presses it.
- It has either one state or two. A two-state button changes its label—for example, from "Start" to "Stop"—after it's clicked.

A slider is a device that sets a value. The position of the knob within the slider's bar indicates the slider's current value. Sliders are useful for setting either continuous or discrete values, between a minimum and maximum that you specify. A slider is often paired with a text field that displays or sets the value shown in the slider.

Switches

A button being pressed

Radio buttons



A two-state button, before and after clicking

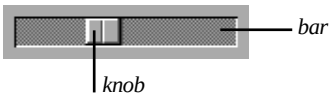
Browsers let users display and select hierarchically organized data such as directories and files. The levels of the hierarchy are displayed in columns.

A group of related Control objects can be collected into a single Matrix object. For example, although the three buttons in YourCall's interface might each be placed individually in the interface, they are instead created as ButtonCells in a Matrix object. The Matrix object groups related controls in rows and columns; it also manages the state of its group. For example, a group of radio buttons is managed by a Matrix object to ensure that only one button at a time is highlighted. Although buttons are among the most commonly grouped controls, other controls—such as text fields and scrollers—can be grouped by a Matrix.

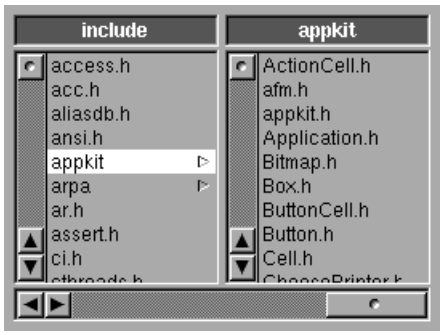
The other controls not discussed here are covered elsewhere in this section. Forms and text fields are discussed in "Text Handling," above. Scrollers are mentioned in "Organizational Views," below. Color wells are discussed in "Support for Choosing Colors," also below.

Interface Builder's Palettes window supplies all the controls. It creates Matrix objects automatically whenever you duplicate a control by dragging its edge while pressing the Alternate key. After creating a control, you can use the Inspector panel to change certain aspects of the control's behavior.

A slider



A browser



Panels, Menus, and Pop-up Lists

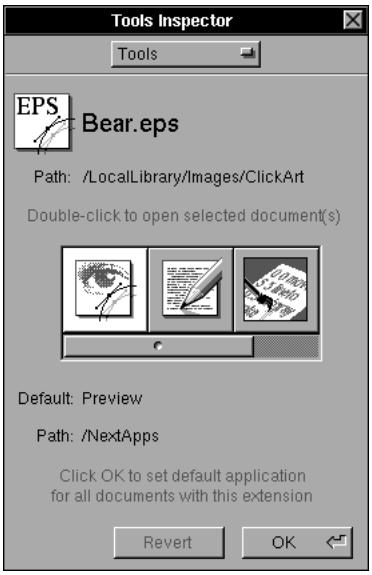
Panels, menus, and pop-up lists are auxiliary windows that let the user give instructions to an application.

Panels look like normal windows, but they don't contain the main information of the application. Because any information they contain is only supplemental, they can save screen space by disappearing when the user switches to another application. Although panels are usually used to get user input, they can also be used to give information—such as help—to the user. The Application Kit provides many Panel subclasses to use for standard panels, such as Font panels, Save panels, and Print panels. A Panel is used in YourCall to alert the user when a user name is needed or when a name isn't in the database.

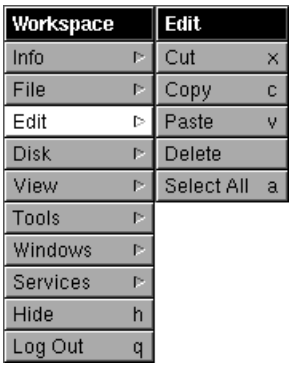
Menus are panels that list commands for the application. The menu commands provide access to the whole range of the application's functionality. NeXTSTEP ensures that only one application's menus are visible; the visible menus for one application automatically disappear when the user starts working in another application. A menu consists of a Menu object, which contains a Matrix object that displays a list of commands.

Pop-up lists let the user choose one option out of many, just as radio buttons do (see "Controls," above). In general, pop-up lists are used instead of radio buttons when space is limited. Pop-up lists look a bit like menus, but they're activated by a button in a window or panel. When the user presses the button, the list is displayed. After the

A panel



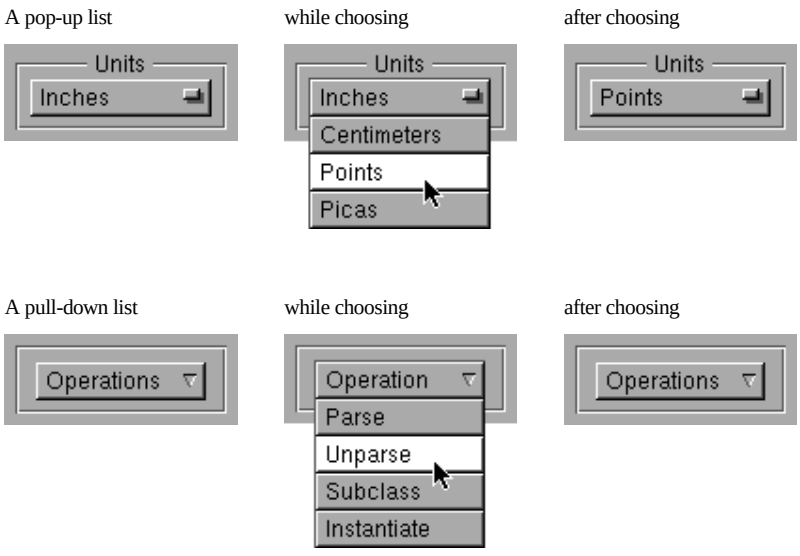
Main menu and an attached submenu



user chooses an item from the list, the button title changes to the chosen item.

A variation of pop-up lists is the pull-down list, which is usually used like a menu: to display commands. A pull-down list's button title, unlike the title of a pop-up list, doesn't change when an item is chosen. A pop-up or pull-down list consists of a PopUpList object and a Button object that is displayed when the list isn't showing.

Interface Builder supplies a main menu for each new application; it also lets you drag panels, menu items, and pop-up lists into your application. To put certain standard panels into your application, you drag a related menu item into your application's menu. For example, after you drag a Font menu item into your application's menu, your application automatically gets a Font panel (along with other functionality). You can create your own menu items by dragging a general-purpose menu item into your menu, editing its text, and then specifying what message that item should send when clicked.



Text Handling

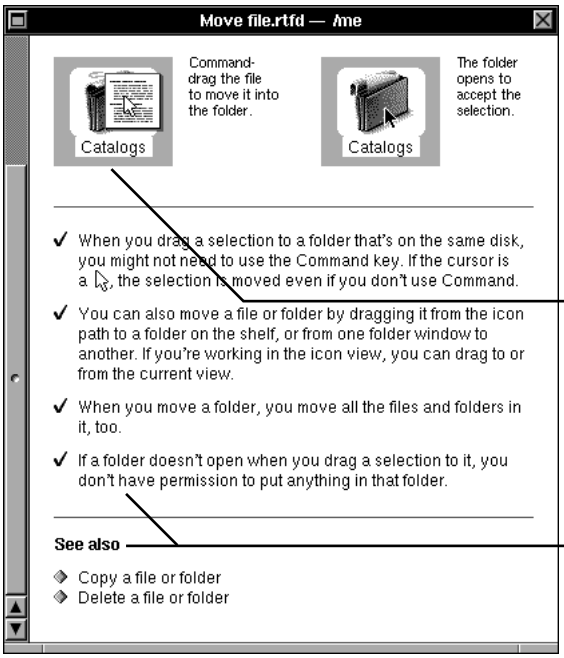
The Text class provides a comprehensive set of text-handling features, so that you'll rarely need to supplement its functionality. Among other things, a Text object can:

- Display text (in either plain ASCII or Rich Text Format.)
- Let the user enter text
- Give the user control of fonts and paragraph formats
- Let the user cut, copy, and paste text
- Wrap text on a word or character basis
- Display graphic images within text
- Read or write a file
- Provide a spelling checker for the user
- Print displayed text and graphics

Although it's possible to customize Text objects, you might not need to, since Interface Builder's Palettes window already provides some of the most useful text configurations:

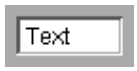
- A ScrollView containing a full-featured Text object, which can be used either for text entry and editing or for displaying uneditable text.
- A Form, consisting of labeled text fields. When the user presses Tab, the insertion point jumps to the next field in the form. (The YourCall application uses a Form for displaying customer information.)
- A TextField, which lets a user enter a single line of data. (YourCall uses TextFields for Question and Answer text.)

Text in a ScrollView



A Form

A TextField



graphic image within a text area

different paragraph styles, different font sizes and styles

Fonts

Three classes support choosing and displaying fonts: Font, FontPanel, and FontManager. Together, these classes let the user specify the fonts used in your application's editable Text objects.

The Font class provides an efficient, object-oriented interface to PostScript fonts. Each Font object records a font's name, size, and style. One Font object is created for each PostScript font used in an application, regardless of how many documents use the font.

The FontPanel lets the user preview fonts and change the font of any selected text. The actual changes are made by sending conversion messages to a FontManager object.

The FontManager is the center of activity for font conversion. It accepts font conversion messages (usually from the Font menu or the Font panel) and changes the font of the currently selected text.

To create FontManager and FontPanel objects, just drag a Format or Font menu item from Interface Builder's Palettes window into your application. Font objects are created automatically whenever your application uses a new font.

A Font panel

Support for Printing

Three classes provide support for printing in NeXTSTEP: PrintInfo, PageLayout, and PrintPanel. Each application that prints has a PrintInfo object that keeps track of information needed for a certain print job. The PrintInfo object gets its information from PageLayout and PrintPanel objects. The illustrations below show the panels that PageLayout and PrintPanel use to request information from the user.

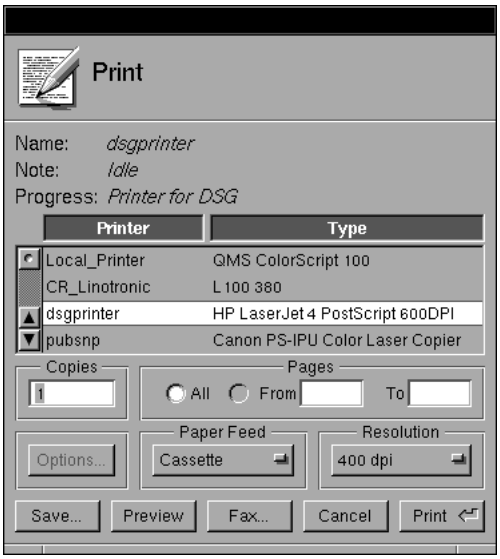
PageLayout and PrintPanel differ in the scope of the information they keep. A PageLayout object usually keeps information about a particular document; this information is used for both displaying and printing the document. A PrintPanel object, on the other hand, doesn't remember anything about particular documents—it keeps track only of the current print job.

Besides printers, PrintPanel also supports fax modems. Clicking the Print panel's Fax button brings up a Fax panel, which gathers the information necessary for faxing.

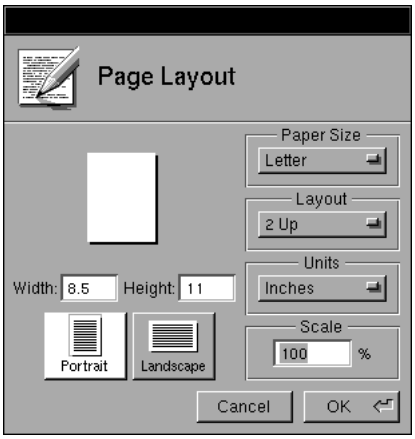
You can easily add support for printing to your application. To get a PageLayout object, just drag a Format menu from Interface Builder's Palettes window into your application's menu. (The Format menu includes a Page Layout command that brings up the Page Layout panel.) PrintPanel and PrintInfo objects are automatically created when your program sends a print message to a View or Window in your application.

Because every visible object already knows how to display itself, you don't have to do *any* additional work to make it print or fax itself. This is one of the advantages of using the PostScript language for all imaging.

A Print panel



A Page Layout panel



File Management

The SavePanel and OpenPanel classes help manage files in an application. Specifically, they query the user for the name of a file to save or open.

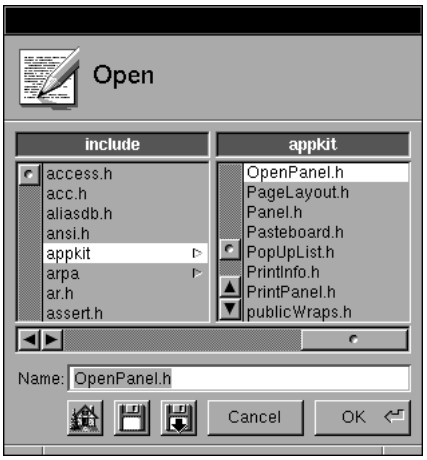
OpenPanel and SavePanel can restrict the types of files they display. You specify these types in a list of filename extensions such as ".tiff" and ".eps". OpenPanel and SavePanel also provide typing shortcuts; for example, pressing Command-space completes a partially specified filename. OpenPanel can also let the user open multiple files at once.

SavePanel takes care of many details of writing files, such as:

- Adding a particular filename extension if the user specifies a name without the proper extension
- Asking for confirmation if the user tries to change the document's name to that of an existing file
- Creating new directories if the user specifies non-existent directories in the file path (after confirming that the directories should be created)
- Alerting the user if a file or directory couldn't be created

Interface Builder supplies default menu items for opening and saving documents. After dragging these menu items into the application, you associate them with code that you write to handle opening and saving your application's documents. In this code, you create SavePanel and OpenPanel objects and write to or read from the file whose name they obtain from the user.

An Open panel



A Save panel

Images

The NXImage and NXCursor classes help you use images in your application. NXImage objects manage general-purpose images for you. NXCursor objects are more specialized: They let you determine which image the Window Server displays for the cursor.

Each NXImage object contains one or more representations of a single image, but doesn't render any of them until you tell it to. Each image can be drawn to a View or to another image. An image can be created from:

- Encapsulated PostScript code (EPS)
- Bitmap data in Tag Image File Format (TIFF)
- RenderMan Interface Bytestream code (RIB) for photoreal 3-dimensional rendering
- Raw bitmap data
- PostScript code in an object that's designated to do the drawing

NXImage manages an image by:

- Reading image data from a file or other source. Bitmaps can be created from uncompressed bitmap data or from data that uses Lempel-Ziv Welch, PackBits, or JPEG compression.
- Keeping one or more representations of the same image. For example, an NXImage object might have an EPS representation and two TIFF representations (such as 2-bit gray scale and 12-bit color) of the same image.
- Choosing the representation that's appropriate for any given display device, such as the screen, a fax modem, or a printer. For example, when displaying to a monochrome screen, an NXImage object will choose a 2-bit gray-scale representation over a 12-bit color representation. The chosen representation is cached in an off-screen window.
- Copying the image from the off-screen cache to where it's needed, using a flexible, efficient imaging technique known as *compositing*.

A composite image

NXImage objects are useful for rendering images that will be drawn more than once or that you want to keep multiple representations of.

NXImage also handles transparency. For example, in the car on the facing page the side windows are semi-transparent. When the car is composited over the text, NXImage allows the text to show through the windows.

Unless you design a cursor, you probably won't have to deal with NXCursor objects at all—NeXTSTEP's default handling of the standard cursors is adequate for many applications. For custom cursor images or specialized cursor handling, however, you need to use the NXCursor class.

Each cursor image is contained in a separate NXCursor object. The Application Kit provides two ready-made NXCursor objects: the I-beam cursor, which is displayed over editable or selectable text, and the standard arrow cursor. A third standard cursor, the spinning wait cursor, is displayed automatically by the system, so it has no global NXCursor object. You can make your own cursor design (cross hairs, for example) and put it in an NXCursor object.

One way to make an NXCursor object the current cursor is to send a set message to that object. Another way of setting the cursor is to send a message to a View, telling the View to use a particular NXCursor object whenever the cursor is inside a specified rectangle.

Standard arrow cursor

I-beam cursor

Support for Choosing Colors

NXColorPanel and NXColorWell provide the standard user interface for selecting color in an application.

NXColorPanel lets the user preview and specify colors in any of the following modes:

- Color wheel
- Slider (including RGB, CMYK, and HSB color model sliders)
- Custom palette (loads a TIFF image for the user to choose colors from)
- Custom color lists, including a list of PANTONE. Colors for calibrated color selection

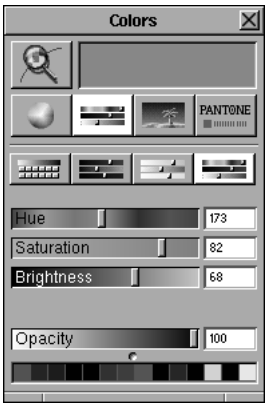
In any mode, the user can also capture a color from anywhere on the screen.

The user can keep frequently used colors in a row of swatches along the bottom of the Colors panel. These swatches remain constant between applications. For example, if a user adds a color swatch in one application and then starts up a second application, the Colors panel in the second application displays the same color swatches as in the first application, including the new color swatch.

The user can set the color of a selection by dragging a swatch from the Colors panel into a color well associated with the item. (The Text object also lets users set text colors by dragging directly into a selection.) A color well, which is provided by an NXColorWell object, is a control for setting and displaying a single color value. For example, a drawing program might provide two color wells: one for setting the color of the selected graphic's outline, and one for the color of its fill.

To get a Colors panel, just drag a Colors menu item from Interface Builder's Palettes window into your application. A Colors panel is also available in any application using color wells—just double-click the border to display the panel. Interface Builder's Palettes window also has a color well that you can drag into your application.

A Colors panel



A color well

Organizational Views

The Application Kit has three classes for organizing Views within a window: Box, ScrollView, and NXSplitView.

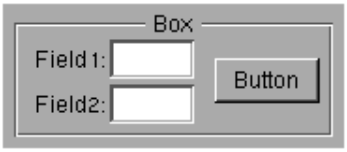
A Box object provides a box that surrounds one or more Views, grouping them both visually and programmatically. The box can have a title.

A ScrollView object is useful when a View might be too big to fit in the space available. By putting the View inside a ScrollView object, you determine how much of the View is visible. The user can then manipulate scrollers to determine exactly which part of the View is visible. The ScrollView object doesn't actually do the scrolling—it works by coordinating the actions of a ClipView object and two Scroller objects. The ClipView object clips off the parts of the View that aren't being displayed. The Scroller objects let the user see and change the position of the View in the ScrollView.

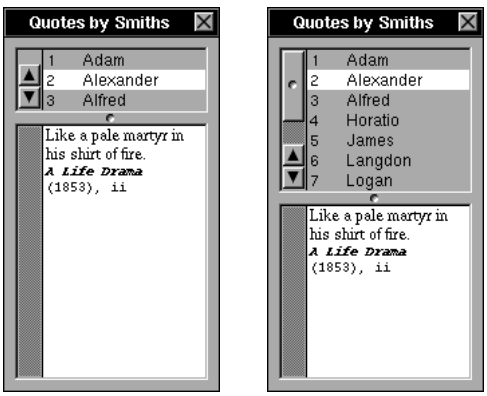
An NXSplitView object lets you display two Views in a fixed amount of vertical space, while giving the user the option of determining how much of each View should be shown. The two subviews in an NXSplitView are separated by a horizontal bar called the divider. When the user moves the bar, one subview gets smaller and the other gets larger, but the sum of their heights remains the same.

With Interface Builder, it's easy to put Views into Box and ScrollView objects and to manipulate the Views. An NXSplitView object requires a little more care: You specify its position and size by direct manipulation in Interface Builder, and then you programmatically connect its subviews.

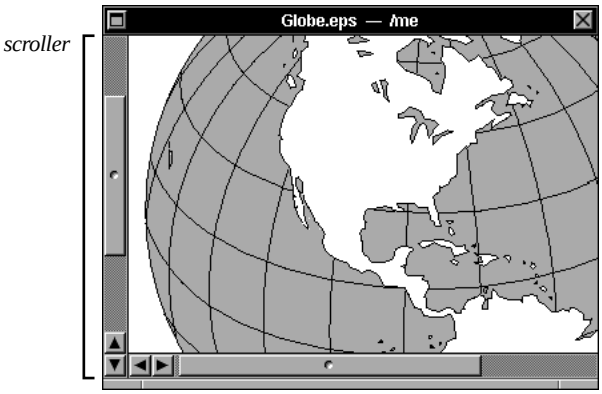
Views in a box



A split view before and after dragging the divider



A document in a ScrollView



The Help System

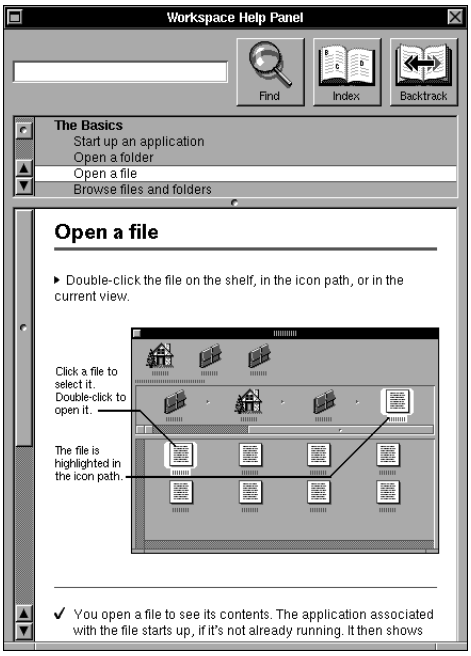
NeXTSTEP also provides a standard way to implement context-sensitive, application-specific help for your programs.

The NXHelpPanel class provides a consistent user interface for help. Using a book metaphor, the help panel presents information that's accessible in a variety of ways. A table of contents and index help users look up and access help by browsing through topics. Hypertext-style links enable quick jumps between related topics. The backtrace mechanism lets users flip back through previous information. Interface Builder provides ways to connect the help system to the controls and fields in an application's user interface to more closely match user needs.

NeXTSTEP supports help text development through the Edit text editor. Using Edit, you can create text containing both images and hypertext-style links. Support for multilingual help text is provided through NeXTSTEP's bundles facility.

NeXTSTEP tools provide additional aid in adding help to your application. Project Builder will automatically create a directory for your help text through its Add Help Directory menu. Interface Builder lets you add a standard Help item to your application's menu. This standard menu item automatically incorporates NeXTSTEP's generic help on a variety of operations, such as using a mouse, printing, faxing, saving a file, and so on.

A Help panel



Spell Checking

While NeXTSTEP offers a standard spell checking service for most text editing applications, it also provides ways to implement customized spell-checking services. This mechanism can be used to implement language-specific and technology-specific spell-checking services. It's implemented with a combination of classes and protocols.

The NXSpellChecker class provides a standard panel that's used for spell-checking in all applications. The NXSpellServer class provides a way for you to register a spell-checking service and make it available in the spelling panel.

A spell-checking service is usually a small application whose sole purpose is to provide spell checking—although it's possible for another application, such as a word processor, to provide its spell checking machinery to other applications through such a service. The Application Kit provides several protocols that a spell-checking service can implement in order to perform spell checking.

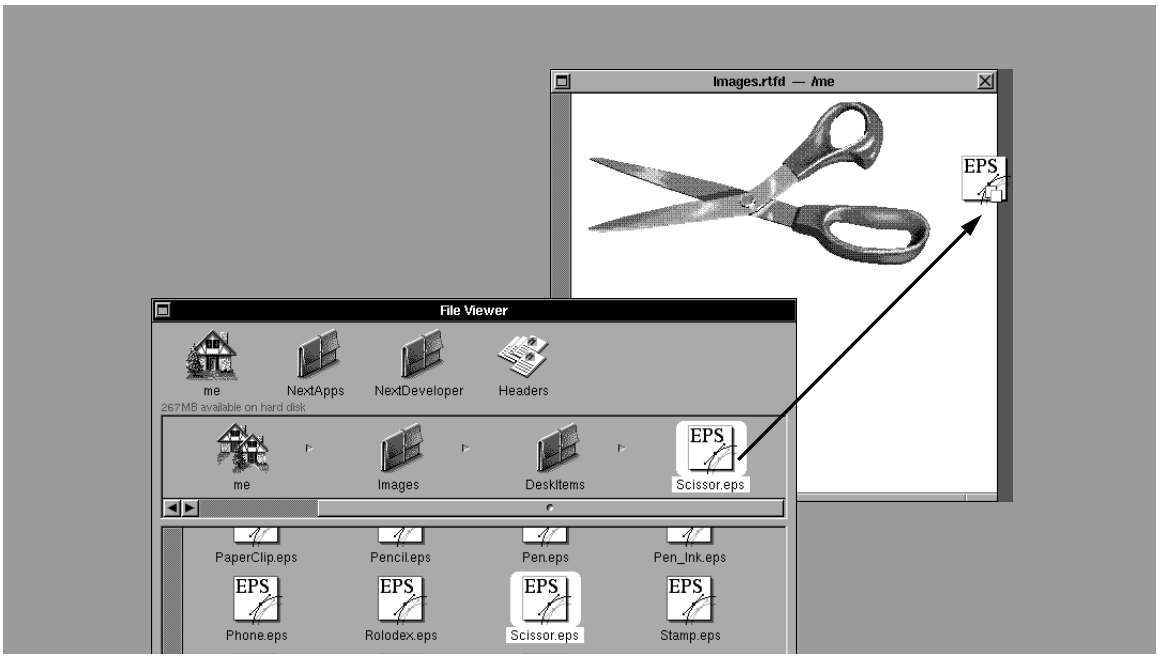
A Spelling Checker panel

Drag and Drop

The Application Kit's dragging protocols let your application accept data and documents created in other applications through a simple drag-and-drop user interface. These protocols also enable your applications to send their data to other applications running in NeXTSTEP.

The dragging protocols include methods to be implemented by a dragging source view, and a dragging destination view. The NXDraggingSouce protocol lets you specify an image to represent the data being dragged and the behavior of your application after dragging is complete. The NXDraggingDestination protocol provides ways to accept data being dragged in and incorporate that data into a document. The Application Kit also implements methods that provide information about a dragging session, such as source application, destination application, and so on.

Dragging an image file from the Workspace into Edit

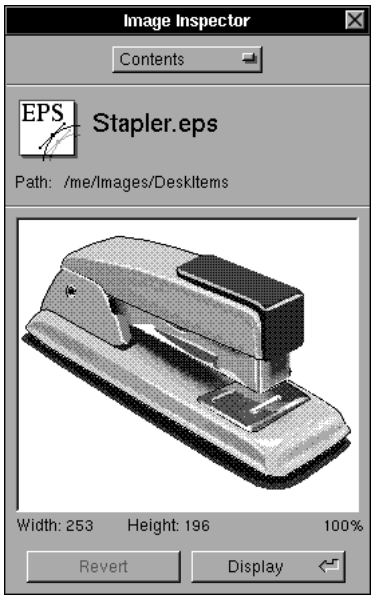


Workspace Inspectors

Another set of NeXTSTEP protocols provides you with the ability to implement inspectors for an application's documents. Inspectors are used by the Workspace Manager to let users "peek" at the contents of a file without actually opening the document and starting the application.

An Attributes Inspector panel

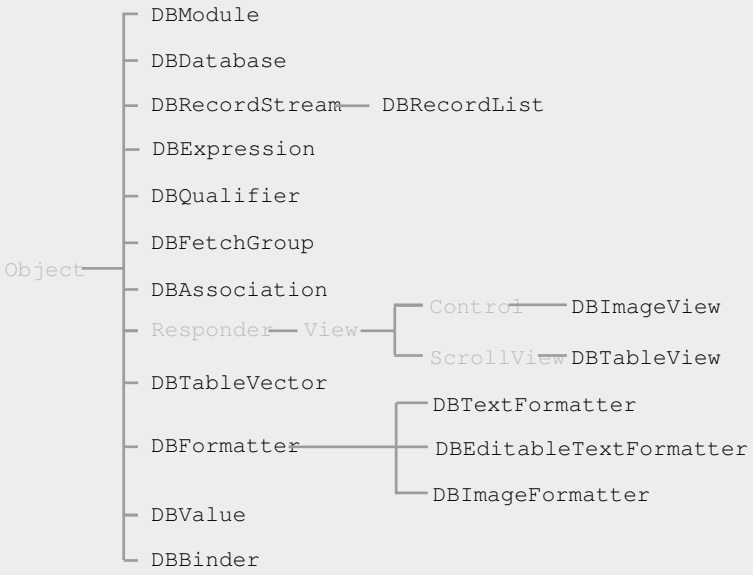
An Image Inspector panel



THE DATABASE KIT

The Database Kit answers a major challenge in custom application development: getting information from large databases to the desktop in the most useful form possible. By providing data access within NeXTSTEP's object-oriented application framework, this kit improves database programmer productivity and ensures that database applications mesh seamlessly with other applications.

DATABASE KIT CLASSES

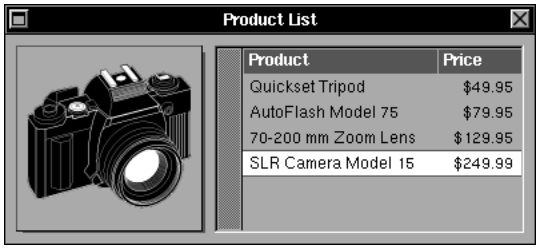


The Database Kit provides a layered approach to data access, enabling applications to choose the level of data manipulation they wish to implement.

The Interface Layer

The top level of the Database Kit—the level that conceals the most detail from application developers—consists of classes that can be used to construct an application solely within Interface Builder. An application can be built on this layer with no source other than its .nib file.

DBTableView is a user interface device for displaying scrollable data browsers. A single table view can be used to display data from one or several entities. Multiple table views can be used to implement master-detail browsing. When browsing data, other objects in the interface are automatically synchronized to the data selected in a table view.



An image view and a table view

DBImageView lets your applications display graphic data from a database (or any other source): employee photos, illustrations, or other visuals. Like DBTableView, DBImageView can be used in any NeXTSTEP application, whether or not it's based on the Database Kit.

The interface layer also provides several classes for formatting data, including DBImageFormatter, DBTextFormatter, and DBEditableFormatter.

DBModule is mostly a convenience class; its role is to connect with objects in the user interface and dispense messages to the appropriate access layer objects as the interface is manipulated. To do this work, DBModule makes use of other interface layer objects, including DBFetchGroup, DBRecordList, and DBAssociation. These classes provide the actual mechanism for accessing the data and synchronizing its display in the user interface.

The Access Layer

The core component of the access layer is DBDatabase. An object of this class can be thought of as representing the database server by managing the connection to the server and by mediating data transactions between the server and other objects. The DBDatabase object interprets a model created with the DBModeler application, using information in the model to create and configure instances of other access layer classes. These classes, DBRecordList, DBExpression, and DBValue, are used to represent the contents of database entities.

Database Adaptors

Adaptors insulate Database Kit applications from database management system specifics. With the proper adaptors, a Database Kit application could work with data stored in ORACLE, SYBASE, or any other data management system.

Since the Database Kit provides full access to the database, programmers need not write code to interact with the adaptor layer. NeXTSTEP provides adaptors for the ORACLE and SYBASE database management systems; third party vendors provide adaptors for other database servers.

THE INDEXING KIT

The Indexing Kit is a set of classes for managing data, especially the large amounts of data characteristic of information-intensive applications: document control systems, library search and retrieval systems, and so on. The Indexing Kit is able to efficiently store and retrieve a wide variety of data types, including text, sound, images, and Objective C objects.

INDEX KIT CLASS

The Indexing Kit is divided into several layers, to let programmers choose the level of detail that best suits the needs of a particular project.

Storage Management

The basis of the Indexing Kit is the IXStore and IXStoreFile classes.IXStore is a fast, transaction-oriented, compacting storage allocator. IXStoreFile is a file-oriented subclass of IXStore. IXStore's transaction model supports access by multiple users and provides a tracking mechanism to ensure data integrity. The upper layers of the Indexing Kit build on this storage by adding data typing and specialized access and retrieval mechanisms.

Associative Access

Classes in this layer—IXBTree and IXBTreeCursor—provide facilities for flexible storage and associative retrieval of values by key. IXBTree provides the basic storage mechanism, organizing untyped blocks of storage by key value. The keys for accessing this storage can be strings, floating-point numbers, or other values, including complex structures. IXBTreeCursor provides the mechanism for traversing the data in an IXBTree. Using multiple IXBTreeCursors, a data manager can perform searches on multiple keys and allow shared access to data by multiple users.

Data Management

The Indexing Kit's data management layer provides access to structured data. IXRecordManager objects can maintain groups of objects, each of which represents an individual record. IXRecordManager builds and maintains indexes of these objects, based on the values they return in response to a specified message. As objects are added to the IXRecordManager, they are automatically indexed appropriately. The IXRecordManager provides fast and space-efficient object storage and retrieval, using either Indexing Kit protocols or standard Objective C archiving.

File System Searching

The IXFileFinder and IXFileRecord classes implement a simple mechanism for accessing the UNIX file system. IXFileFinder treats files in the file system as records. Its indexing facilities, based on IXRecordManager, enable fast access to specific system files. An IXFileFinder can be configured to ignore specific files or file types, or particular subdirectories within the target directories. In addition to its indexing facilities, IXFileFinder supports whole and partial word searches by literal strings and regular expressions.

Text Parsing

The text parsing layer of the Indexing Kit is useful for providing fast, indexed access to large bodies of text. The text parsing layer can also be used to manage multiple text documents in cooperation with lower layer components of the Indexing Kit. Two classes, IXAttributeParser and IXAttributeReader, provide the mechanism for indexing the contents of a text file based on word frequency or other attributes.

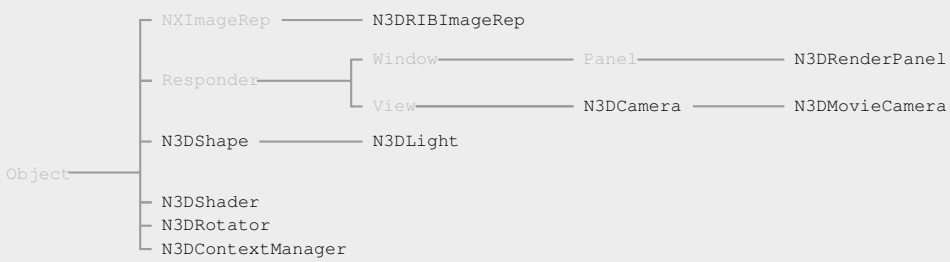
Query Processing

At the top level, the IXAttributeQuery class, with the Indexing Kit's query language, provides access to data through simple query expressions. Using IXRecordManager and IXFileFinder facilities, an IXAttributeQuery object can accept a query language string and return objects matching the query.

THE 3D GRAPHICS KIT

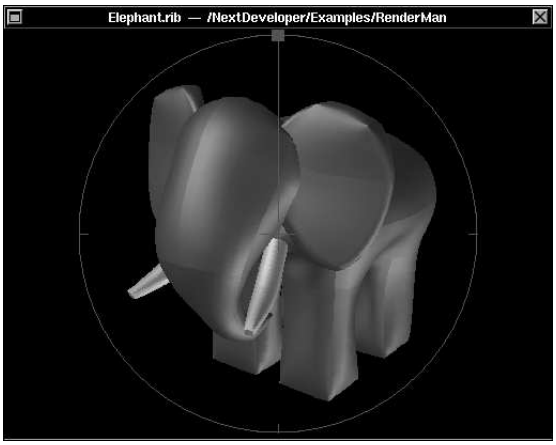
The 3D Graphics Kit provides photoreal and interactive rendering of 3-dimensional scenes, seamlessly integrating the display and printing of 3D images with the standard PostScript capabilities of NeXTSTEP. Using 3D Kit objects, it's easy to implement complex graphing and data modeling, 3D component design and testing, and photorealistic animation.

3D GRAPHICS KIT CLASSES



Drawing and Printing

The N3DCamera class provides the basic mechanism for drawing on the screen and printing to the printer. As a subclass of the Application Kit's View class, the N3DCamera class performs both 2D (PostScript) and 3D (RenderMan) drawing from its drawSelf:: method.



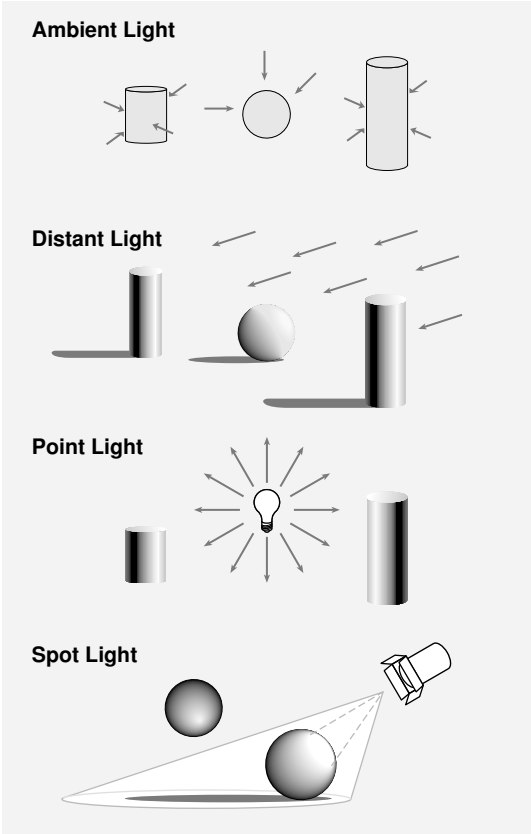
The N3DMovieCamera class extends the N3DCamera class by adding methods for creating 3D animation sequences. Like its superclass, N3DMovieCamera can be used both for interactive and photoreal rendering of scenes.

Modeling

The N3DShape class provides a basic architecture for spatial organization of 3D models. By subclassing N3DShape and incorporating code to render the RenderMan image primitives, you can create specific components of a 3D model.

Lighting

The N3DLight class provides control over various RenderMan-standard lighting features. You can set spot lights, flood lights, point lights, and ambient lights, adjust their intensity, and set their color and other attributes.



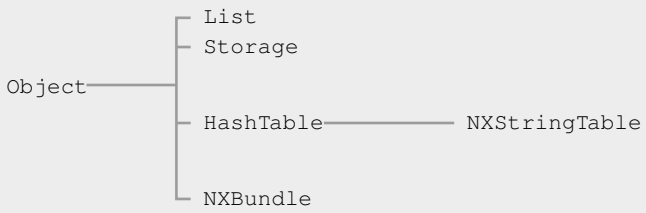
3D Object Manipulation

The N3D Rotator class provides an easy-to-implement "virtual sphere" for rotating objects in 3D space.

COMMON CLASSES

A handful of classes come with the NeXTSTEP run-time system for the Objective C language. They include, most prominently, the Object class, which defines the basic functionality inherited by all objects. The other common classes are useful for creating objects to manage data of various kinds.

COMMON CLASSES



Object is the root of almost all classes. Classes inherit certain basic abilities from Object, such as allocating and freeing space for each object, copying objects, and testing for the ability to respond to a message.

The Storage class provides dynamically allocated storage of arbitrary data. Essentially an object-oriented dynamic array, this simple class provides methods for adding, listing, counting, and removing data from its storage.

The List class provides a dynamically-sized array of Objective C objects. Along with its storage and retrieval methods, List provides a mechanism for sending the same message to all the objects it contains.

HashTable associates two items of data—a key and a value. The values it stores can be of any C or Objective C type, including objects. HashTable provides a convenient and efficient way to store and access unordered data by key value. One example of using a HashTable for object storage and access is found in the YourCall application described in "Step-By-Step Through a NeXTSTEP Application."

NXStringTable is a HashTable subclass that associates two character strings: a key and a value. By using an NXStringTable object to store your application's character strings, you can reduce the effort required to adapt the application to different language markets.

NXBundle is an object that corresponds to a directory where program resources are stored. The directory "bundles" a set of resources, and NXBundle makes those resources available to an application. NXBundle is used to implement features such as multilingual application development and application-specific help.