Everest Authoring System

Design Guide

By Intersystem Concepts, Inc.

Acknowledgements

We would like to thank the following people for their help during the development of this version:

Malcolm Burk
Gary Klimple
John Simpson
Don White

For More Information & Technical Support

Contact your vendor, or the developer:

Intersystem Concepts, Inc.
P.O. Box 477
Fulton, MD 20759

410-531-9000
301-854-9426 (fax)

http://www.insystem.com

1 2 3 4 5 6 7 8 9 0

Table of Contents

1 Introduction

1.1 Introduction to the Design Guide

The Design Guide

This book bridges the gap between the Tutorial and the Technical Reference/on-line help. The Tutorial helps to get you started, and the Technical Reference provides the specific details. The topics in the Technical Reference are organized in alphabetical order, which is excellent when you know the name that identifies the topic you want.

But, what if you know what you want to do, but don't know what it is called in Everest? That's when the Design Guide is useful. The Design Guide is not alphabetically organized, it is *functionally* organized. Use it to learn what items (features, objects, attributes, commands, etc.) in Everest can do what you need, then consult the Technical Reference/on-line help for the details.

We have attempted to arrange the chapters of this Design Guide so that the information builds on the that of the prior chapter(s). If you so choose, this approach will allow you to learn much about Everest by reading the chapters in the order in which they appear. Each chapter contains many short topics that can be used as a reference as well.

As is true in many software tools, in Everest there is more than one way to achieve a desired result. The ability to choose the best approach is the mark of an experienced author. This book attempts to compare many different authoring techniques, and show you when to use each.

2 Books, Pages and Objects

2.1 What is Everest's Metaphor?

The authoring metaphor (or "approach") used in Everest is one of books and pages. Basically, you as author create pages as you wish them to appear to the end user, and group the pages into books. This metaphor is particularly well-suited for Computer-Based Training applications because books have been used for centuries to disseminate information and to teach. Now, it has also become the metaphor used on the Internet.

All versions of Everest, and its predecessor, Summit (for DOS), have employed pages as the primary container of content. In prior versions of the software, pages were instead called "screens" but the difference is simply one of terminology. In other authoring software, you might have heard pages referred to as frames, cards, scenes, slides, or whatever.

When building a project, most authors think of how the screen (in this usage, the display monitor) will look. In fact, they think of how the screen will look at the beginning, then after the user makes a choice from a menu, and as they move forward, and so on, all the way to the end. With Everest, you take these mental snapshots, and create each one as a page.

Pages are not the most elemental building blocks in Everest (objects are, and we'll cover them in a bit), but they are the easiest to visualize.

2.2 Branching

You will need to tell Everest how you want the user to move from one page to another. This is known as branching. There are several approaches to branching; the three most popular are described below:

Linear

By linking the pages together via branching instructions, you can create a slideshow that the user can step through. This is known as linear branching because it proceeds from start to finish in a straight or nearly straight line. Such branching is depicted well by a flowchart.

Multiple Path

If your project is a Computer-Based Training (CBT) application, it is likely that you will want multiple paths: "if the user answers correctly, branch to the next question, if not, branch to remediation." Such branching is depicted poorly by a flowchart.

Random Access

This is often called non-linear branching, or sometimes hypertext. It means the user can jump from place to place in the material, sometimes freely, usually with some guidance. The standard Microsoft Windows help system employs this type of branching. Such branching is not practical for depiction via a flowchart.

Everest can handle these three branching types (and others as well). The choices are open for you. Later chapters describe how to do each.

2.3 Pages With/Without Branches

Should you include the branching instructions within the pages, or keep them separate? That's a hot topic of discussion these days. Object-Oriented Programming (OOP) says the latter is correct. And, for the long run, it is probably better.

The drawback is that creation of the pages and the branching separately is more difficult than combining the two. Specifically, more difficult to create...but easier to maintain.

Fortunately, Everest lets you take either approach. You can include explicit branching instructions with the pages, or allow the pages to simply branch in the order in which they appear in the flowchart Everest calls the Book Editor.

2.4 What Are Pages Made Of? Objects!

Think about the appearance of the first page in your project. Usually the first page is some variety of title page. Often it contains text and a picture or two. Perhaps your first page includes a multimedia element such as music. It is likely that there is an "OK" or "continue" button somewhere for the user to click.

In Everest, elements such as text, pictures, music and buttons are objects. There are over 30 object classes built into Everest. Via extensibility (sometimes called add-ins, snap-ons, etc.), you can add additional, custom object classes. Each object class has a name, such as Textbox, and a representative icon. The icons of all object classes are displayed in the sizable ToolSet window. To learn the name of an object, simply move your mouse over the icon; the name will be displayed as the caption of the ToolSet window.

Each Everest page that you create is really nothing more than a collection of these icon objects. To create the page, you drag the icons you want from the ToolSet window and drop them on the VisualPage editor.

2.5 Objects Have Attributes

Everest icon objects start life with some preset values known as "defaults" (for example, a Textbox starts out empty). You can modify the values, called attributes, to meet the needs of your project. For example, you will probably type some text into that Textbox object. Doing so modifies the value of the Textbox's Text attribute (you changed it from empty to the text you typed).

Some objects have only a few attributes, some have over 30. Most attributes are edited via the Attributes window. When you change the attribute of an object, it takes effect immediately. That's one of the benefits of an object-oriented approach in software.

As a very simple example, compare non-OOP and OOP approaches to changing the color of some text already printed on the page. In the non-OOP world, you would:

> determine what text was already on the page
>
> erase the text
>
> select the new color
>
> plot the text again

In Everest's OOP world, you:

> change the color attribute of the text

and the object does so for you. Pretty nifty, eh?

Note that Everest does not offer full-blown object-oriented programming. What we have done is implemented the parts of OOP that offer the most benefit for interactive multimedia development.

2.6 Introducing The Book Editor

Everest's Book Editor displays several things. First, it displays an icon for each book (each .ESL file) in the current subdirectory. To open a book (and reveal the pages inside), you double click your mouse on the book. Beneath the book icon, the Book Editor shows the pages of the currently open book. To open a page (and reveal the objects inside), you double click on the page.

Purpose of the Book Editor

Most pages you create with Everest consist of more than one object. A typical simple page has a Layout (that describes the window in which the page appears), Textbox, Button and Wait (for user interaction) object. Complex pages might have 10, 20, or even more such objects, each represented by an icon in the Book Editor.

When the page is presented to the user (sometimes this is called "playing back the page" or "at run time"), the objects do not all flash onto the computer monitor at once. Instead, they appear in a defined order...an order that's determined by you, the author. Of course, on a reasonably fast computer, all the objects may seem to appear at the same time, but in reality they are displayed one at a time *in the order in which you arrange their icons in the Book Editor*.

The Importance of the Order of the Icons

Why are we making such a big deal about what may seem to be a minor distinction? The reason is that the order can be important in certain situations; for example, if you want a text heading to set the stage for an animation sequence, you must be sure to tell Everest to play back the text before the animation.

The way you do so is by placing the icons in the desired top-to-bottom order within the Book Editor. The Book Editor maintains the objects of your page in the order in which they will be played back for the user (basically, it's a flowchart).

The next topic describes the basics of using the Book Editor.

2.7 Editing With The Book Editor

The Book Editor window is displayed (along with the ToolSet and VisualPage editor) when you author. If the Book Editor window is obscured by another window, press Ctrl+B to bring it to the top. Here are several common editing techniques:

Highlighting Objects

Many Book Editor operations require that you first highlight the items (book, page, object) you want to edit. To highlight multiple contiguous objects, hold down mouse button 1 and drag. To highlight scattered objects, hold down the keyboard's Ctrl key while clicking the mouse on the icons.

There is an alternate way to highlight several objects at once. In the VisualPage editor, click and hold mouse button 1 on the background area (i.e. not on an object) and drag the mouse. A "rubber band" rectangle will appear. Surround the desired objects with this rectangle. When you release the mouse button, Everest will highlight these objects in the Book Editor.

POINTER - General Information

The Pointer is the arrow that appears in the Book Editor to the left of the icons. The Pointer marks where the next (i.e. new) icon will be inserted. You can drag the Pointer with the mouse. Everest also automatically moves the Pointer when you drag icons within the Book Editor.

BOOKS - Opening

To open a book (and view/edit the pages within), double click on its icon. Book names are displayed using upper-case letters and a bold font, and resemble BOOK1.ESL. Only one book can be open at a time. The name of the currently open book is displayed as the caption for the Book Editor window, and the word "open" is displayed within the book editor.

BOOKS - Opening Different Subdirectory

The Book Editor shows all the books in the current subdirectory. To use a different subdirectory, from the Author window's File menu, choose Open Book/Page.

BOOKS - Creating New

To create a new book, from the Book menu, choose New. You'll be prompted to enter the name of the book; enter up to 8 alphanumeric characters. If you would like to also create a new subdirectory to hold the new book, simply prefix the book name with the subdirectory name. For example, if the window shows the current location to be C:\EVEREST, if you enter `project1\book1`, Everest will create C:\EVEREST\PROJECT1\BOOK1.ESL.

BOOKS - Closing

To close an open book, double click on its icon. Alternatively, open a different book.

BOOKS - Copying

To copy an entire book, close it first, then hold down the Shift key and drag the desired book's icon using the right side mouse button. Everest will then ask you to enter a name for the copy. You can also copy books manually outside Everest via DOS or the Windows File Manager: simply copy the desired .ESL file.

BOOKS - Deleting

To delete a book, highlight it and from the Edit menu choose Delete. Careful: deleting a book deletes all its pages! You can also delete books manually outside Everest via DOS or the Windows File Manager: simply delete the desired .ESL file.

BOOKS - Finding a Page

To find a book that contains a page with certain text, from the Book menu choose Find. This feature is helpful when you do not remember the name of a page, or in which book it is located.

BOOKS - Passwording

To assign an editing password to a book, first open the book, then from the Book menu choose Password. Once you assign a book a password, when you later open the book for editing, Everest will ask you to enter that password. Passwords are a convenient way to discourage others from editing the pages of your book.

PAGES - Opening

To open a page (and view/edit the objects within), first open the book in which it is located (if necessary), then double click on the page icon. Only one page can be open at a time. Page names are displayed using lower-case letters and a bold font. The word "open" is displayed next to a page that is currently open.

PAGES - Creating New

First, move the Pointer to the desired location for the new page, then from the Page menu choose New. Alternatively, drag a Page icon from the Toolset and drop it at the desired location in the book.

PAGES - Copying Within Same Book

To copy a page (for example, to use it as a template), hold down the Shift key and drag the desired page's icon using the right side mouse button. When you drop the icon, Everest will prompt you to enter a new name for the page; each page in a book must have a unique name.

PAGES - Copying Between Books

To copy a page from another book, first open the book that contains the page(s) to copy. Highlight the desired page(s) and from the Edit menu choose Copy. Then open the book where you want to place the copies. Move the pointer to the desired location, and from the Edit menu choose Paste. Alternatively, to copy a single page to another book in the same subdirectory, hold down the Shift key, drag the desired page's icon via the right-side mouse button, and drop it onto the destination book's icon. Everest will automatically open that book, and place the page at its end.

PAGES - Copying Objects into Current Page

To add the objects of one page to another, first open the destination page, then drag the source page's icon into it.

PAGES - Re-ordering

To move a page to a different location in the book, simply drag its icon with the right mouse button. Do not hold down the Shift key.

PAGES - Finding by Name

To find a page within the current book, first click on the Book Editor window (to make it the active window), then press the first letter of the page's name.

PAGES - Finding by Content

To find a page within the current book that contains certain text, from the Page menu, choose Find.

OBJECTS - Editing

To edit an object (such as a Textbox or Picture), first open the page that contains the object, then click on the desired object. Its attributes will appear in the Attributes window. For Program and Menu objects, double click on the icon to open the editing window. Object names appear in lower-case letters and a non-bold font.

OBJECTS - Creating

To create a new object, drag its icon from the Toolset and drop it on either the VisualPage editor or the Book Editor.

OBJECTS - Copying

There are several ways to copy an object. Within the Book Editor, hold down the Shift key and drag the source object's icon via the mouse (use the right side mouse button), then drop it at the desired location in the page. Alternatively, in the Book Editor, click on the object to copy (in order to highlight it), then hold down the Shift key while dragging an icon (of the same class) from the Toolset, and drop it on the VisualPage editor.

OBJECTS - Moving

Two different types of movements are available. To change the *position* (execution order) of an object within a page, drag it and drop it at the desired location via mouse button 2 (the right side button). To change the *display location* of a group of objects in the VisualPage editor, highlight those objects in the Book Editor, and from the Object pull-down menu choose Move. You can then specify the distance to move the objects.

OBJECTS - Arranging

Sometimes you might want to visually align several objects (such as a group of Buttons) into a column or row within the VisualPage editor. To do so, highlight the desired objects in the Book Editor, and choose Arrange from the Other menu. You can then specify the distance between the objects.

OBJECTS - Toggling

Objects can be "toggled" off and on. An object that is toggled off is ignored when the page is run. Authors often temporarily toggle off certain objects for debugging purposes. To toggle an object, highlight it, then from the Object menu choose Toggle. The Book Editor displays $ $ next to the icons of objects that are toggled off. To toggle the object back on, repeat the process.

OBJECTS - Hiding

On very complex pages, objects may overlap and become difficult to edit. To temporarily hide an object (so that others beneath it can be edited more easily), highlight the object to hide, then from the Object menu choose Hide. The Book Editor displays H next to the icons of objects that are hidden. To make the object visible again, repeat the process. Note: Hide works only during editing; to alter the visibility of an object at run time, use the Initially attribute.

OBJECTS - Asterisk

An asterisk (*) near an object's icon indicates that you have changed that object. Everest removes the asterisks when you save the changes.

OBJECTS - Commenting

To write a comment about an object, first highlight the object, then double click on the comments box near the bottom of the Book Editor.

2.8 Objects Galore

At this point you might be wondering what the built-in objects can do. Here's a list in the order in which they appear in the ToolSet:

Page. Drag and drop this to create a new page.

Layout. Use the Layout object to set window size, location, title bar caption, etc.

Picture. Displays graphics images (.BMP files, etc.), with special effects and cel-animation if desired.

SPicture. Similar to a Picture, but scales image to fit inside the box.

Textbox. Displays simple text. Text is all one foreground & background color combination (you choose). You can also choose the font and its size. If desired, text can be loaded from an ASCII disk file at design time or at run-time.

Flextext. Similar to a Textbox, but more flexible in terms of colors and fonts. Also allows you to specify hypertext-style jump words.

Input. Accepts fill-in-the-blank style input. Specify anticipated answers with the object.

Mask. Similar to Input, but allows finer control of the characters the user can type, and the formatting that is applied to them.

Check. A check box field. User can toggle the state of the each check box. Offers optional 3-D shadowing.

Option. An option button (also known as a "radio" button). Can be arranged into groups so user can select one of the group. Offers optional 3-D shadowing.

HScroll. A horizontal scroll bar. Allows user to adjust a analog slider.

VScroll. A vertical scroll bar.

Combo. Displays a drop-down list and allows user to choose one item.

Listbox. Displays a scrollable list of items, and allows the user to select (highlight) one or more.

Button. A push button. Provides "clickable" options.

Menu. Adds/modifies pull-down menus in the window.

Wait. Pauses for a time period and/or waits for a user action, such as a response to a question, click of a button, etc. Here you can also specify the events that cause a branch to another page, as well as the name of the destination page.

Judge. For CBT, marks the location in the page where answer judging occurs. The judge object simply indicates when the user's response is compared with that answer list. The user's responses and answer judgment results can be stored in variables you specify.

Program. A freeform text editor in which you can enter programming code. Everest's programming language, named A-pex3, offers over 30 commands, 40 functions, variables and arrays.

Gauge. Displays an analog representation of a value. For example, a bar that slowly fills, or a needle gauge (like the fuel gauge in a car).

Media. Controls MPC compliant multimedia devices, such as CD-ROM, videodisc, WaveAudio, VCR, Microsoft Video for Windows, etc. Very powerful, if you have the multimedia hardware.

Animate. Plays animation files created with Autodesk Animator (.FLI, .FLC, .AAS files). Sound can be included with animation. Some very impressive animation can be performed (3-dimension, shadows, etc.). Animation can run in background (i.e. while other things are happening on page).

Shape. Displays a simple shape: box, circle, rounded box, etc. Line thickness, drawing size and fill style can be specified.

Line. Draws a line segment; thickness and style can be specified.

Frame. Displays a border, or box with 3-D shadowing, to visually group other objects or text.

Picbin.   Holds a group of icons or pictures for display on other objects, or for animation effects.

Erase.   Erases the page display and/or objects.

Include.   Combines the objects from another page with the current one at run time.   Very much like a subroutine call.

JLabel.   Manually controls the flow of the page.   You can employ the JUMP command in programming to redirect page execution to the location of the JLabel.

Hyperhlp.   Links to the Windows Help system, so you can include hypertext features in your project.

OLE.   Object-linking and embedding.   An advanced way to incorporate portions of other applications within your page.

Timer.   Generates an event at a regular interval.   Can be used to place a time limit on questions, display an updating time clock, or do anything else at a regular interval.

To view technical information about an object, click *once* on its icon in the Toolset, then press Shift+F1 for help.

The Toolset might contain additional icons.   These are icons for objects added via Everest's extensibility feature, which is described in the following topic.

2.9 Extensibility

Everest is an extensible authoring system.   That means its functionality can be expanded via add-on component objects.   Icons for such add-on objects appear inside Everest's ToolSet, and can be dragged into a page just like any of the built-in objects.

These add-on components, in turn, can expand the operation of your project to include: display support for dozens of different graphic file formats; spreadsheets; word processing; scientific instrument support; databases; client-server computing; fractal image compression/decompression; and much more.

Everest add-on components (in the form of .VBX and .OCX files) are sold by vendors such as Microsoft, Borland, Autodesk, FarPoint, MicroHelp and many more.

A later chapter of this Design Guide is devoted to exploring extensibility in depth.

2.10 A Typical Page

Most pages contain several objects.   Here's how to create a very simple page.   First, drag a Page icon from the ToolSet and drop it on the VisualPage editor.   When prompted, enter simple   as the name of the page.   Next, drag one of each of the following objects from the ToolSet to the VisualPage editor: Layout, Textbox and Wait.   After doing so, your Book Editor will resemble:

Object	Name
(Icon) open	**BOOK1.ESL**
(Icon) open	**simple**

(Icon)	simple_layout_A
(Icon)	simple_textbox_A
(Icon)	simple_wait_A

Everest automatically assigns a default name to each object you add to a page. The name is a combination of the page name, object class name and a letter (A for the first, B for the second, etc.). You can change the name as you wish by editing the Name attribute in the Attributes window.

To type some text in the Textbox, first click on it in the VisualPage editor. Later, when ready to view the page's appearance to the end user, press the F8 key to Preview it. To end the preview and return to the editor, double click on the control box in the upper-left corner of the preview window.

2.11 Max Out

Everest has plenty of capacity to handle even very complex projects. Only in very unusual circumstances might you run into a limit of some kind. A few of these possible limits are discussed here.

The maximum number of objects in one page is 250. If you need more, you can do it via the Include object (that treats an entire page as one object). The maximum number of objects of a given class (such as Textbox) in one window is 99. The maximum amount of memory for object attributes storage is 64000 bytes per window.

Before you reach any of these limits, it is likely you will hit the Microsoft Windows resource limit. Windows gives you easy access to more memory than DOS, but its resource limit is quite tight. Every object that has a visible component in the VisualPage editor, such as a Textbox or Picture, consumes resources. Pictures consume the most.

At design time, you can monitor the amount of resources remaining via Everest's About window. Free resources are expressed as a percentage. Anything below 25% is considered low. It is important to keep this Windows limitation in mind when you design your project if you want users to be able to run it.

If your project runs low on resources while it is executing, you can remedy the situation by reducing the number of objects in windows, reducing the number of open windows, and by instructing users to close other applications. You can monitor free resources at run time via the Fre() function.

2.12 Books: .ESL Files

The pages you create for a project are stored on disk in what Everest calls a book. The file name extension of a book is .ESL (which previously stood for "Everest Screen Library" and has been retained for upward compatibility). The first book has a default name of BOOK1.ESL. You can create another book from the Book pull-down menu by choosing New.

Most authors put each project in a book of its own. A page in one book can branch to a page in another; the two books need not be in the same location (i.e. DOS directory).

The maximum number of pages in a book is not limited by Everest. Instead, it is limited by available memory and disk space to approximately 4000 pages. Also, the larger a book is, the longer it takes to find and load objects from it.

Our experience is that the average page occupies less than 2K of disk space. Everest automatically compresses the page before saving it into the book.

Certain graphics files can be embedded into the book. These are also stored in a compressed format (.BMP files compress very well, often to a mere 3% of their original size). These features are described in more detail later in the Design Guide.

2.13 The Starting Page

Users run projects via the ERUN run time software. When a particular user runs your project for the first time, Everest looks for a page named @start, and begins there. Always create an @start page in your book. The @start page is a good place for hardware checks and/or your project's title page.

If you are employing the bookmark feature, when a user resumes where he left off previously, Everest will resume at the page named @restart or @preempt (if one exists in the book). You'll find more information about both elsewhere in this Design Guide.

2.14 Book and Page Utilities

An earlier topic in this chapter discussed basic operations on books and pages, such as creating, renaming, and deleting. The AUTHOR program also contains other utilities; most authors find these of greater use after they have created a fair number of pages and/or books.

Global Search and Replace

To search the current book for a particular text string or number, from the Author window's Utilities menu choose Search and Replace. If you wish, you can narrow the search to only certain attributes, and replace what is found.

Copy Template Page

If you have an existing page you wish to use as a template for a new page, from the Author window's File menu, choose Load template. Then, select the page you wish to use as the template.

Copy/Lock Pages

To copy multiple pages from one book to another, from the Author window's Utilities menu choose Copy Pages. The Copy Pages utility has an option to lock the copied pages to prevent further editing by anyone. If you lock pages, be sure to maintain an unlocked version for yourself!

The Copy Pages utility can also be helpful to recover portions of a damaged book. The undamaged portions can often be recovered simply by copying the pages into a new book.

Importing Text

To import a text file (perhaps in order to paste it into a Textbox), from the Author window's Edit menu choose Load Clipbrd w/ASCII. Then choose the text file to import. Proceed to the desired pasting location, and choose Paste from the Edit menu.

Compact Book

If you delete pages and/or embedded files from a book, Everest marks the space within the .ESL file as "reusable." The .ESL files does NOT automatically decrease in size. To reclaim such unused space, from the Author window's Utilities menu, choose Compact Book.

The Compact Book feature will also scan your book for internal storage errors, and can correct many types of storage problems. Additionally, it optimizes the arrangement of data within the book to maximize retrieval speed.

Print Book

To print a hardcopy (paper copy) of the pages in the current book, from the Author window's File menu, choose Print Book.

3 Windows

3.1 Working With the Editing Windows

While authoring, several windows appear on your monitor: the VisualPage editor, the Book Editor, the Attributes window, the ToolSet, and others. They can be identified by their captions. To make sufficient room for all these windows, we recommend that authors employ a 1024x768 screen resolution. In 1024x768 resolution you can place the ToolSet on the left, a 640x480 window in the middle, and reasonably sized Book Editor and Attributes windows on the right.

At lower resolutions, you probably won't have enough screen real estate to display all these windows simultaneously. You can overlap them, or open them as needed.

If you have trouble locating a window (perhaps it is hidden beneath another), here's what to do. First, find the Author window (that's the one with the File menu). If it is hidden too, press Ctrl+M (think M for "main menu"). The Author window has a pull-down menu item named "Window." When you pull down this menu, you'll see a list of the other editing windows. Choose one, and Everest will make it visible. Alternatively, press the hot key associated with the window (as shown on the menu).

A check mark next to a window name in the Window pull-down menu list means that window is already open. Selecting it again from the list does NOT close the window; instead, it brings that window to the foreground. To close a window manually, double click in its upper-left corner. Exception: the VisualPage editor cannot be closed.

3.2 Adjusting Editing Window Sizes

You can relocate and resize the editing windows to meet your needs. When you have them the way you want, from the File menu, select Save .INI. Everest remembers the location and size of each window via the EVEREST.INI file.

When you resize the ToolSet, Everest automatically rearranges the object icons to fit. This means you can orient the ToolSet vertically, horizontally (like a button bar) or as a square.

Though you are allowed to resize the Author window, the only reason to do so is to view the status windows that are displayed just beneath the pull-down menus. One status window shows numerical information. These values represent the X-Y location (in pixels) of the highlighted object in the VisualPage editor, as well as its width and height. Or, click inside the VisualPage editor, but not on an object, to learn the X-Y location at which you clicked. There's also an Internet/intranet status window that is handy when you are running projects via the Web.

3.3 Controlling The Run Time (User's) Window

At run time (i.e. when you test run your project, or the end-user plays it back), Everest displays your project's pages within a window. This window acts as a container for the objects of your pages. Be sure you are clear on the distinction between the terms "page" and "window."

page	a group of objects you assemble with Everest
window	the container in which objects are displayed at run time

If you do not specify otherwise in your project, Everest runs your project within a "default" window. However, authors almost always employ a Layout object in their pages to control the size and shape of this window.

Most authors place a Layout object at or near the top of *each* page of their project. The Layout object has attributes that let you control the location of the window on the display, its height and width, as well as color and other appearance items.

You might be wondering why each page needs a Layout object if the first page already has one. Won't the Layout from the first page "stick" as branching proceeds from one page to the next?

The answer is yes; the window will remain the same size when the second page is applied to it. However, when debugging your project, at times you might want to start at a particular page in the middle of the project (Everest lets you start debugging at any page). If that page does not have a Layout object, Everest won't know the size of the window (nor any other Layout attributes), and will employ the defaults. That might make the page awkward to debug. So, for your convenience, we recommend "a Layout on every page."

3.4 A Layout On Every Page

If you do put a Layout object at the start of every page, there are two caveats to keep in mind. The first involves resizable windows.

At your option, via the WindowBorder attribute, you can allow the user to resize and move the window(s) in which your project is displayed. If the user moves a window, when they advance to the next page in the project, that Layout object you put at the top is going to force the window back to its original location!

Fortunately, there is an easy way to address this issue. To prevent the window from moving back, make sure the Relocate attribute of the Layout object is set to No. When Relocate is No, Everest moves and sizes the window only according to the first Layout object applied to it...it ignores any subsequent Layout objects in that window.

The second caveat is related to project maintenance. If you put a Layout on every page, and later decide one of the attributes needs to be changed, you'll need to change it on every page!

As you might suspect by now, there's a very simple solution to this problem: object instancing. Object instancing is a way multiple pages can all share the same object. Information about object instancing appears in the next chapter.

3.5 Choosing Window Sizes

Now that you know that a Layout object determines the size of the container window, you need to decide what size that window will be.

Users run Microsoft windows at a variety of monitor resolutions. Some of the common ones are: 640 x 480, 800 x 600, 1024 x 768 and 1280 x 1024. If you want, at run time, your project can examine the user's monitor resolution (look in Sysvar(19) and Sysvar(20)).

Many authors are tempted to adjust the size of the project container window to match the resolution of the monitor (i.e. maximize the window), so that the window covers the entire screen, even at high resolutions like 1280 x 1024. We recommend that you avoid this. The main objection is that the reason users run Windows at high resolutions is so they can get more application windows on the monitor at once, not so they can have larger application windows.

Recommended Approach

We recommend that you determine the lowest monitor resolution at which users will be running Windows, then design your project to fit inside a window of that size. In most cases, that will be 640 x 480...the default Width and Height for the Layout object.

If you want the user to be able to resize the window as desired, set the Layout object's WindowBorder attribute to 2. If you also want Everest to automatically proportionally resize the objects to match the size of the window at run time, set the Layout object's AutoResize attribute to Yes.

Caveats

Note that the Layout object's Width and Height attributes represent the distance between the outside edges of the window. That means the area inside the window (where you can place objects) is somewhat less (the border takes up some room, as well as the title bar, etc.).

More bad news: the exact number of pixels inside the window varies depending on the resolution in which Windows is run! This is because Windows changes the size of the borders according to the display resolution. Our advice is to not worry too much about this, and design your pages with some extra room at the bottom and right.

If, for some very special application, you must know exactly how many pixels are within the window, you can employ the Gsm() function at run time. The Gsm() function can tell you how many pixels Windows is occupying with its border elements. By subtracting these values from the width and height of the window, you can determine the size of the area inside, and relocate objects at run time.

3.6 Handling Multiple Windows

By default, Everest displays your project in a single window (window number 1). You can open up to 8 windows simultaneously. You do need to exercise caution when opening many windows because each consumes Windows resources. The Fre() function lets you monitor available resources.

You can open a window via the BRANCH, CALL and OPEN commands, or the Create attribute (all described elsewhere).

Even though the windows can communicate with each other, they function as independent units. When two windows are open, it is like running two different projects at the same time. Each window maintains its own backup and menu stacks; each can access a different book; and each maintains a bookmark. Variables in your programming, however, are global (shared by all open windows).

The user can move the focus from one window to another by simply clicking on it. Windows that have a control box can be closed by the user at any time. If the user closes the last open window, Everest handles this as a request to log off. You can force the closure of a window via the Destroy attribute or RETURN command.

4 Objects and Attributes

4.1 Object Attributes

As you probably know by now, pages consist of a collection of objects. In a similar way, objects are a collection of attributes. All objects have attributes. Some have only a few; others have 30 or more. The attributes of a Textbox include the size of the box, the color, the font, and the text itself. You set the values of most attributes at design time (i.e. while authoring).

While you are editing an object, Everest lists its attributes in the Attributes window. (You can open the Attributes window, if needed, by double clicking on an object in the Book Editor.) Those object attributes that can be changed at design time are listed in a scrollable area of the Attributes window.

Editing

To change an attribute, first click on it to load it into the editing field near the top of the Attributes window. You can then change the value as you wish. Click on the arrow to the right of the editing field to display a list of possible values of the attribute. Not all attributes have a list, but many do.

Those attributes that do have a list of possible values can also be edited by double clicking on their name in the attribute list. Doing so either advances the value to the next possibility, or opens a window from which you can select a value.

4.2 Object IDNumbers

One of the attributes that most objects have is an identification number (IDNumber). The IDNumber uniquely identifies the object within its class in a window. (A "class" is simply the object's type, such as Textbox, Picture, Button, etc.).

Most of the time, you can ignore the IDNumbers and simply let Everest assign them as you create your project. However, it is important that you understand the significance of the IDNumber because it plays a very important role in the operation of your project.

To understand the significance of the IDNumber, consider an example. Say your first page contains a Textbox object. That Textbox's IDNumber attribute is likely 1 (the default; you can assign any IDNumber in the range from 1 to 99).

On the second page in your project, you also have a Textbox object. If you also assign this Textbox IDNumber 1, then at run time, when you branch from the first to the second page, the

Textbox of the second page will REPLACE the Textbox from the first page. Since both IDNumbers are the same, they actually refer to the same Textbox in the window.

But, what if the IDNumber of the Textbox on the second page is not also 1...what if it is 2? Everest deems these to be two different Textbox objects, so upon running the second page, it ADDS another Textbox to the window.

So, the IDNumber lets you control how Everest places objects into the container window at run time. Sometimes when you branch from one page to the next, you'll want to replace/update an object, and other times you'll want to add another object. Assigning appropriate IDNumbers is the way to make this happen.

4.3 Even More About Object IDNumbers

The IDNumber is one of the most important attributes of an object. When you drag an object from the ToolSet and drop it to add it to the page, Everest assigns the next available IDNumber on that page. So, the first Textbox on every page will be assigned IDNumber 1. You can modify the number as you wish within the range 1 to 99; you do *not* need to number the objects consecutively.

It is important to note that the IDNumber is not unique to the object. For example, a page can have a Textbox with IDNumber 1, as well as a Button with IDNumber 1. Even though their IDNumbers are the same, they are considered different objects because they are in different classes.

Ready for an esoteric fact? Rarely is this encountered, but you should know it anyhow: one design time page (not run time window!) can have two Textboxes that both have the same IDNumber. At design time, they will appear as two separate Textboxes, but at run time, can you guess what will happen? (No, the program will not blow up.) As the page is executed, the first Textbox with IDNumber 1 will appear, and then the second Textbox with IDNumber 1 will REPLACE it. This is really the same as the earlier example that had two Textboxes with the same IDNumber on separate pages.

At some point it is likely that you will no longer want one or more objects to remain in the window. The next topic talks about one way to remove them.

4.4 Which IDNumber Should I Assign?

As already mentioned, you can assign an object an IDNumber from 1 to 99. Which number should you assign? About 90% of the time you can simply leave the IDNumber unchanged (i.e. accept the one Everest assigned by default). However, there are situations in which you might want to choose a different IDNumber. Three of the most common situations are described below:

1) When you want to replace an object within a window with a new one (of the same class) at run time, assign the new one the same IDNumber as the previous.

2) When you are creating objects that are to remain in the window for a long time (for example, navigation buttons), assign the objects IDNumbers that are "unusual." We recommend 90 to 99. That gets them out of the way, and helps to isolate them from the objects that will be changing frequently within the window.

3) t The Erase object (which removes other objects from the window) lets you specify a range of IDNumbers to remove from the window. All objects, regardless of class, that have

IDNumbers in the range you specify are erased. You can assign an IDNumber either inside or outside the erasure range you choose, as dictated by the effect you want.

A technical note: objects with high IDNumbers (such as those from 90 to 99) do not consume any more memory than those with low IDNumbers.

4.5 Object Instancing

One of the most powerful features of Everest is known as object instancing. Via object instancing, your page can reference an object previously created (on a different page).

For example, if you previously created a particular Textbox on page45 and want to also display it on page88, you can do so without recreating the Textbox (and retyping all that text!). Here's how:

1) on page88, create a new Textbox
2) double click on the Textbox in the Book Editor to open the Attributes window
3) from the Attributes window's Options menu, choose Instance of...
4) a list of the names of Textbox objects (that have their SaveAsObject attribute set to Yes) is displayed
5) double click on the one you want

The steps above make page88 "point to" the same Textbox object that page45 does.

Was the Instance of window empty when you tried this? Here's why: only those objects previously saved into the book with their SaveAsObject attribute enabled will appear in the Instance of... window. By default, SaveAsObject is disabled, so the objects you've created previously are not available for instantiation. That's easily remedied: open an existing page, find the desired object, double click on its SaveAsObject attribute to enable it, and resave the page. Now, it should be available for instantiation.

4.6 So What's The Big Deal About Instancing?

In the example in the previous topic, object instancing saved retyping the text of a Textbox object. That's nice, but is there more? Yes! When the text of the Textbox is changed in one location, all the pages that refer to that Textbox (via object instancing) are updated automatically by Everest.

That's the solution to the "Layout on every page" problem of the previous chapter. If you refer to the same Layout object via object instancing on every page, later when you make a change to it, all pages will be updated for you. That's a big time saver!

Object instancing also helps to minimize the disk space used by your project. Even if two (or more) pages refer to the same object via instancing, that object is stored only once on disk.

4.7 Instance Attributes

Now you say, "that's nice, page88 needs the same Textbox that page45 has, but I want to display it at a different location in the window." Everest has the solution for you: instance attributes.

Certain attributes of an object are defined locally with that object (and are stored with that page's Book Editor). These are known as instance attributes, or local attributes. They can be safely changed without impacting the object elsewhere.

It is easy to recognize instance attributes because they are displayed using a bold font in the Attributes window. The location and size attributes of objects (Left, Top, Width and Height) are always instance attributes. So, on page88 you can change them (relocating the Textbox as needed) and not worry about altering page45.

4.8 Save As Object?

By default, objects are NOT stored in a way that makes them accessible to other pages via object instancing. You must tell Everest you want an object to be available elsewhere. To do so, set the SaveAsObject attribute of the object to Yes.

If you discover that you need an object that had not been previously saved as an object, simply load the original page that has the object, change its SaveAsObject attribute to Yes, and resave the page. You will then be able to use object instancing from other pages to access that object.

Why doesn't Everest automatically save ALL objects so they can be instantiated? The reason is that every object that has SaveAsObject set to Yes requires extra disk space for storage. Since most objects are referred to by only one page, it is not efficient to save all of them as independent objects.

Other Reasons to Enable SaveAsObject

There are two other situations in which you should set SaveAsObject to Yes: 1) Program objects that are referenced via the GOSUB command, 2) objects with large amounts of text.

When SaveAsObject is Yes, Everest combines the instance (local) attributes of an object with the page and saves the resulting combination. It also saves the object independently of the page. When SaveAsObject is No, Everest saves all attributes of an object combined with the page, and does not save the object independently.

Each page is limited to approximately 32,000 characters. When SaveAsObject is No, objects with long text strings (particularly Textbox and Program objects) can cause the limit to be exceeded. That's why you should set SaveAsObject to Yes if such objects contain lots of text.

Object Manager

Once an object has been saved into the book with SaveAsObject enabled, it acquires an independent existence. If you delete a page that has such an object, Everest removes the page, but leaves the object in the book. Everest keeps the object because it might be referenced by other pages.

You might wish to determine if such independent objects exist in a book. To do so, from the Author window's Utilities pull-down menu, choose Object Manager. In the Object Manager window, you can select the object class, cross-reference an object (to determine which pages reference it, if any) and delete objects.

4.9 Object Instancing Vs. Include Object

Another way to access previously created objects is via the Include object. The Include object takes another page (and all its objects) and combines it into the current page. In effect, Include treats a whole page of objects as one object.

Include is useful when there are so many common objects you want to combine into the current page that you are reluctant to access each via object instancing (simply because it would increase the length of the page).

In general, however, we recommend that you employ object instancing instead of the Include object because it makes the objects easier to edit. With Include, the objects do not appear in the VisualEditor. With object instancing, they do. With Include, to edit an object you must first load the page referenced. With object instancing, you simply double click in the Book Editor to open the Attributes window.

Note that Wait objects are not allowed in included pages.

4.10 Question Mark Attributes

Those attributes with names ending with a ? are Boolean (Yes/No) attributes. The easiest way to change (toggle) their value at design time is to double click on them in the Attributes window.

When employing such attributes in A-pex3 programming, omit the question mark. For example:

```
Input(1).WordWrap = "Yes"
```

For faster execution and smaller storage, in A-pex3 programs set Boolean attributes to numeric values, -1 for Yes, 0 for No. For example:

```
Input(1).WordWrap = -1
```

When examining the value of Boolean attributes in A-pex3 code, do NOT compare the value to Yes or No. For example, do NOT use:

```
IF Input(1).WordWrap = "Yes" THEN   $$ incorrect!
```

instead compare the value to 0 or -1, as shown on the next line:

```
IF Input(1).WordWrap = -1 THEN      $$ correct syntax
```

The chapters that follow have much more information about A-pex3 programming.

4.11 Multiple Windows And Attributes

By default, when you employ an object attribute in an A-pex3 program, Everest refers to the object in the currently active window. The number of the currently active window can be found in the Sysvar(8) variable.

In certain situations, you may want to read from or write to the attributes of an object located in a different window. To do so, prefix the object name as illustrated:

```
Window(2)!Input(1).WordWrap = -1
```

The example above sets the WordWrap attribute of the Input object in window number 2. Note the exclamation point in the example; that's how Everest detects a window reference.

If desired, you can express the window number via a variable. For convenience, employ 0 for the currently active window. For example, the following two statements refer to the same window and object:

```
Window(sysvar(8))!Textbox(1).Text = "Done"
```

```
Window(0)!Textbox(1).Text = "Done"
```

4.12 Customizing Default Attributes

All objects have preset values for their attributes. What if you prefer different attributes? Does that mean each time you add an object to your page, you have to manually set those attributes? No, there's an easier way! You can customize the default attribute values for most objects.

To do so, simply create a page with the special name @default. Add objects to it, and set their attributes as you prefer. Be sure to save the @default page in your book. From then on, during this session, any new objects you create on other pages will start out life with the attributes you chose for that object class.

If you leave Everest, and restart the program later, you will need to manually reload the @default page from your book in order to reactivate your custom default attributes. Simply load the page as if you were going to edit it, then go about loading and editing other pages, creating new pages, etc.

5 Events and Branching

5.1 Event-Driven Model

Everest employs what is known as an event-driven model. This lets you design your project to "wake up" and respond to a user action, such as a key press, when the event occurs. Most Windows applications are designed in this fashion because it allows multi-tasking.

In a non-event driven model, software spends most of its time asking "Did you press a key yet? Did you press a key yet?" It sits there polling or waiting for a key press, keeping the computer's powerful microprocessor busy. As Scotty would say, "And at warp 10, we're goin' nowhere mighty fast!"

In the event-driven model, the software says to the computer "I've finished my job. I'm taking a nap. Call me when a key press comes in." The computer (or actually Windows) now has time to let other applications do their job. If the user presses a key (or clicks the mouse, etc.) Windows directs the event to the appropriate application, and says "Hey, you! Wake up! Here's an event for you to handle."

While it is possible to create your Everest application in a non-event driven fashion, we recommend that you do not. The non-event approach ties up the computer and prevents the user from interacting with other Windows applications that might also be running.

5.2 What Are Event Codes?

In Everest, all events can be represented by a number. For example, when the user presses the letter A on the keyboard, event code 65 is generated. Every key on the keyboard is represented by a number (that can be found in Appendix A).

To make your project event driven, tell Everest to "wake up the project when you see event code X" where X is the code number. You specify the desired "wake up and do something" event codes in the Activator attributes of a Wait object.

In addition to keypresses, projects typically need to respond to other user actions, for example, a mouse click on a Button object. Keypresses have predefined event codes, but other actions do not. Therefore, for such actions, you choose and assign a numeric event code yourself...simply pick any number from -32000 to 32000 (except 0). If you prefer, you can also use a quoted string instead of a number.

Button Clicking Example

For example, you probably want to know when the user clicks a Button object. To do so, choose an event code, for example -100. Then, in the Attributes window for the Button, find the ClickEvent attribute, and set it to -100.

To detect the event code (-100 in this example) and do something useful (such as BRANCH to another page), you need to tell the Wait object to look for -100. This is described in the next topic.

5.3 The Main Event: The Wait Object

To detect events codes, and do something useful, employ the Wait object. Put it in your page after the other objects you want to make visible in the window. In the Wait object's Activator attributes, enter the events (numbers or quoted strings) you want to wait for. For example, if the Enter key is your "next page" key, you would look up its event code in Appendix A, learn it is 13, and set NextActivator to that value.

If consulting Appendix A is too tedious, ask Everest to generate the correct keypress event code for you: double click on an Activator name in the Attributes window, then press the desired key.

If you have several different event codes triggering the same action, simply list them separated with commas. For example, you can set NextActivator to 13,114,34 to trap three different event codes.

After entering event codes in the Activator attributes, you should specify the action to perform via the corresponding Action attributes.

A very common action is a branch to another page, as described in the next topic.

5.4 Creating Simple Branching

Perhaps the most common action that your project will take in response to user input (such as a key press or button click) is a branch to another page in the book. This type of branching is easy to construct in Everest.

In a Wait object, if you have specified one or more NextActivators, you should also specify a NextAction. Everest performs the NextAction when the user triggers a NextActivator. For example, if you want to branch to the page named page2, you would enter

```
BRANCH page2
```

as the NextAction attribute of the Wait object. The BRANCH command is a "go to" type command. Execution of your project continues at the page you specify. You can find more details in subsequent topics in this chapter.

If you cannot remember the name of the page to which to branch, double click on the NextAction attribute. This opens the page name dialog box. When you double click on the page you want, Everest automatically constructs the appropriate branching command for you.

The action need not be a BRANCH command; it can be any legal A-pex3 programming command(s). The next chapter has programming details.

Next Page Preloading

Whenever possible, design your project so that branching to the next page occurs via the Wait object's NextAction attribute. Said another way, if you are going to be using the BRANCH command to get to the next page, put that BRANCH command into the NextAction attribute.

Why? At run time, Everest examines the NextAction attribute. If it sees a BRANCH command, it preloads the contents of the indicated page from disk into memory, and compiles Program objects, *while the user is busy viewing the current page*. This happens transparently without the user knowing it. Then, when the user proceeds ahead, Everest can display the next page much more rapidly because everything (pictures, text, compiled Programs, etc.) is already in memory.

Preloading can dramatically increase the apparent execution speed of your project, especially when it is running via the Internet, so take advantage of it whenever possible. All you need to do is specify the name of the next page in a BRANCH command in the NextAction attribute.

Preloading Tip

If your page is complex, it may branch to the next page via a BRANCH command located in a Program object. Does this mean you can't take advantage of Everest's preloading feature? No, you still can! Simply put a duplicate BRANCH command into the NextAction attribute of the final Wait object in the Page, and leave the NextActivators empty. Everest will dutifully preload the page you specify, and if the next BRANCH command executed goes to that page, regardless of the location of that BRANCH (i.e. in a Program or in another xxxActivator attribute), your project will receive the performance boost.

What if the branching of the complex page is determined by user action, and therefore you don't know which page will be next? Decide which is the most likely next page, and put that one in a BRANCH command in NextAction. If the user branches elsewhere, Everest simply removes from memory what had been preloaded.

5.5 Buttons and Events

To create a page that branches when the user either presses a particular key OR clicks on a button, enter the keypress event code as the ClickEvent attribute for the Button.

For example, if Enter (keypress 13) is the next page key, and you have a "Continue" button on the page that should also branch to the next page when clicked, enter the number 13 as the ClickEvent attribute of the button. Keypress codes can be found in Appendix A. (If you'd like Everest to automatically generate an event code for you, double click on ClickEvent in the Attributes window, then press the desired key.)

Of course, you will still need a Wait object to detect event code 13 and issue a BRANCH command. See the previous topic.

5.6 Detecting Other Events

Other events can be detected in a similar fashion. In addition to Next, the Wait object has attributes that let you detect Back (previous page), Menu, Help, Comment and Quit actions. Again, you enter the numeric event code(s) as the Activator, and the desired operation as the Action.

The Wait object also offers additional attributes listed as Other1Activator to Other8Activator and Other1Action to Other8Action. You can use these for any purpose you want. Think of them as local hot keys that spring into action when their particular event is detected. (In fact, in earlier software, these attributes were named Hot1xxx through Hot8xxx. That is, until we noticed that this produced an attribute with the unfortunate name of Hot4Action.)

Here's an example that uses the Other attributes:

start a new page

add a Textbox object to the page

add two buttons to the page

set the first button's Caption to `Up`

set the first button's ClickEvent to `-100`

set the second button's Caption to `Down`

set the second button's ClickEvent to `-101`

add a Wait object to the page

set the Wait object's Other1Activator to `-100`

set the Wait object's Other1Action to:

```
      x = x + 1: Textbox(1).Text = x
```

set the Wait object's Other2Activator to `-101`

set the Wait object's Other2Action to:

```
      x = x - 1: Textbox(1).Text = x
```

Preview (run) the page

click the Up and Down buttons to change the value displayed

5.7 Detecting a Large Number Of Events

The Wait object has room to perform approximately 15 different actions. What if you need more? There is an easy answer: use the Wait object's AllOtherAction attribute. When you enter something for the AllOtherAction attribute, Everest "wakes up" for every event, places the event code in the variable you specify in the EventVar attribute, and executes the action you specify in AllOtherAction.

Most authors use a GOSUB command in AllOtherAction to execute a Program object located in the same page. In the Program object you can write code (such as a large IF...THEN block) that compares the value in the EventVar variable and takes the desired action. Such a program might resemble:

```
IF event = 112 THEN     $$ F1
  CALL help
ELSEIF event = 113 THEN $$ F2
  BRANCH page2
ELSEIF event = 114 THEN $$ F3
  JUMP top
ENDIF
```

You might be wondering why we recommend a GOSUB to a Program object located *in the same page*. If the Program object is stored with another page, GOSUB must find and load it from disk, a significantly slower process than executing a Program object already contained in the page.

5.8 Branching and/or Erasing

When your project branches to another page, the objects of that page are applied to the container window. Recall from the previous chapter that the IDNumber attribute helps to determine whether the new objects are added to the window, or replace previous objects.

It is likely that at some point you will want to remove all or some of the objects that already exist in the window. Many authors put an Erase object at or near the top of the page to remove previous objects from the window.

With an Erase object, you can specify a range of IDNumbers to remove. All objects, regardless of class, with IDNumbers in the range are removed. The A-pex3 ERASE command works similarly.

Other authors prefer to put an Erase object or command near the bottom of the page, after a Wait object, but before a branch to another page. They feel that each page should "clean up" after itself before moving to the next page. Either object erasing technique works; use the one you prefer.

5.9 Using Back-up and Menu Branching

Many projects created with Everest have a top-down, menu structure (the user sees a main menu, then branches through a series of pages for menu item 1, returns to the menu, does the same for menu item 2, etc.). In such projects, authors commonly want the user to be able to back up or return to the most recent menu at will. You can implement this with branching, and a tiny bit of care.

The Wait object has BackActivator and BackAction attributes, as well as MenuActivator and MenuAction attributes. Use these as you do the NextActivators and NextAction attributes as described earlier in this chapter, except let them handle user requests to branch back to a previous page or menu.

Everest offers two branching keywords that make this task easier; @prev and @menu. To branch back to the previous page upon triggering of the BackActivator, set BackAction to:

```
BRANCH @prev
```

To branch back to the most recent menu page upon triggering of the MenuActivator, set MenuAction to:

```
BRANCH @menu
```

The Backup and Menu Stacks

You might be wondering how Everest knows which page is the previous one or the most recent menu. As a user runs the pages in your project, Everest remembers the page names via what are called the backup and menu stacks. In fact, each window has an independent backup and menu stack (see tech ref documentation for Sysvar(71) to Sysvar(78) and Sysvar(81) to Sysvar(88)).

Everest does not put the name of every page on the backup and menu stacks...only those you tell it to via the BackUpStack and MenuStack attributes of the Layout object.

Some care must be taken when deciding which pages to place on the stacks. When a user backs up to a previous page, objects that were present in the window when the user initially encountered the page while progressing forward may no longer be there (due to an Erase of a subsequent page). (Programmers can employ the Obj() function to determine which objects are currently within a window.)

A reasonable rule of thumb: pages that start a new "topic" in the project, and therefore typically erase most or all previous objects in the window, should be placed on the backup stack. Pages whose main purpose is to add objects to those already in the window should NOT be placed on the backup stack.

Avoiding Branching When Stack is Empty

When branching to @prev or @menu, some additional care must be taken to avoid performing the branch when there is no page to branch to (i.e. if the backup or menu stack is empty). In the following examples, the BRANCH command is performed only if there is a page on the corresponding stack:

```
IF len(sysvar(70 + sysvar(8))) > 0 THEN BRANCH @prev
```

```
IF len(sysvar(80 + sysvar(8))) > 0 THEN BRANCH @menu
```

5.10 Stop and Comment Branching

As wonderful as your project is, eventually the user will want to stop running it. Perhaps you would like to save the user's place (a bookmark) so that he can resume the next time.

To Stop Running

When you project reaches a conclusion and you want to stop running it, use one of the following BRANCH commands. A good place for these commands is the QuitAction attribute of the Wait object.

```
BRANCH @finish
```

```
BRANCH @end
```

```
BRANCH @exit
```

@finish

A branch to @finish automatically saves the user's bookmark (if user records are active) at the current page, and branches to a page named @finish (which you need to create). Authors often use the @finish page to display a message such as "Thank you for using this application. The next time you come back you will be able to resume where you left off." The @finish page is also a good place to close any devices opened via the Mci() function, or perform other housekeeping duties.

At the completion of the @finish page, you should perform a `BRANCH @exit` command. This terminates execution of your project, and closes all windows associated with it. Important: in general, you should NOT use the `BRANCH @end` command to exit the @finish page, because doing so will make the user's bookmark point to @finish, rather than at the page at which they quit.

If the user manually closes the window in which your project is running, Everest automatically saves his/her bookmark, but does not jump to the @finish page. If you want the @finish page to be displayed in this situation, put the following in the Layout object's CloseEvent attribute (all on one line):

```
IF ext(124) = 1 & sysvar(172) = 0 & sysvar(18) # "@finish" THEN
sysvar(172) = -1: BRANCH @finish
```

@end

A branch to @end saves the user's bookmark (if user records are active) at the current page, then closes the application and stops its execution. You do not need to (and should not) create a page named @end.

@exit

A branch to @exit does NOT save the user's bookmark (even if user records are active). It immediately closes the application and stops its execution. You do not need to (and should not) create a page named @exit. Use `BRANCH @exit` to exit from an @finish page.

The Bookmark

The "bookmark" mentioned above is a built-in feature of Everest. The run-time software automatically saves the user's place in your project (including window contents, variables, scores, etc.). Next time the user logs on, this enables him/her to resume where left off (i.e. just as if he had never logged off). Each user has his/her own independent bookmark that contains the information unique to that person.

To Collect Comments

To let the user enter a comment, use

```
BRANCH @comment
```

which displays a comment window, if both of the following conditions are met:

You can edit EVEREST.INI with the INSTRUCT program.

1) The Comments item in the EVEREST.INI file contains the name of the file in which to save the comments. This file name should have the extension .ECM.

2) Message -031 in the EVEREST.MSG file contains the instructions you want to display in the comment window.

After the user enters (or cancels) a comment, execution of your project resumes where it left off. Comments can be viewed via the INSTRUCT program.

For Summit for DOS Users

In Summit for DOS, F5 is the default comment key. To reproduce this functionality in Everest, in a Wait object enter

```
        116
```

for the CommentActivator attribute, and

```
BRANCH @comment
```

for the CommentAction attribute.

Retrieving User Records and Comments

To view and print user records and comments, use Everest's INSTRUCT administrator program.

5.11 Advanced Branching

Everest also has the ability to branch to a page in another book, open additional windows, call subroutines, and more. These are considered "advanced branching techniques" and are described in the A-pex3 programming chapter.

5.12 Using Global Hot Keys

What if you want an event to be detected anywhere it occurs in your project? You could put the event code in every Wait object, but that would be tedious. The better solution: global hot keys.

Global hot keys sit quietly in the background, and spring into action when the particular event with which you associate them occurs.

For example, say you want to display a glossary whenever the user presses Alt+g. Appendix A tells us the event code for Alt+g is 4071. So, you need to tell Everest to trap event 4071 as a global hot key. To do so, simply create a page with the special name @k<event code number>. In this example, you would create a page named @k4071.

When Everest detects an event for which there is an @k<event code number> page in the book, it automatically and immediately runs that page. In fact, it performs a CALL to that page just as if you had used a CALL command in an A-pex3 program.

The Subtle Point

In case you missed the subtle point, it is restated here:

the presence of a page with the name @k<event code number> in the book is what activates <event code> as a global hot key.

Using the Page

What actually occurs on the global hot page is up to you. You can put objects on the @k<event code number> page just as if it were any other page. By default, at run time the objects are displayed in the current window. If you want them in a different window, start the hot key page with a Program object that opens a window via a BRANCH, CALL or OPEN command.

Always RETURN

When you are done handling the hot key, and want to return where the event first occurred, use the RETURN command. It is important that you eventually RETURN from a global hot key because doing so releases the memory needed to record where the event occurred (the RETURN information is saved on the CALL stack in Sysvar(58)).

A Print Screen Hot Key

Many authors provide a "print screen" feature in their projects so the user can print any page on their printer. A global hot key makes this easy. First, choose a keypress that will become the print screen hot key; most people make the obvious choice: the PrintScreen key. The event code for PrintScreen is 44. So, create a new screen with the name @k44.

On the @k44 page, place a Program object. In the Program editor window for the Program object, enter

```
dummyvar=ext(19)
RETURN
```

The ext(19) function prints the contents of the current window (actually, it asks Windows to do so...and the Windows Print Manager handles the request). Try it by running your project, and pressing PrintScreen.

5.13 When Global Hot Keys Are Not Global

Global hot keys are associated with a particular window and are recognized by Everest only when that window has the focus. An example to illustrate: if window 1 uses book HOT.ESL that contains @k<event code> pages, and window 2 uses book COLD.ESL that does not contain any @k<event code> pages, then when window 2 has the focus, no hot keys are active.

If you need the hot keys active in all windows, simply be sure to put @k<event code> pages in each book accessed by the windows.

Another note: local event codes (such as those specified with a Wait object) have priority over global hot keys.

5.14 Manually Disabling Global Hot Keys

Unlike in Summit for DOS, in Everest a user can invoke a global hot key while that hot key is still being processed (i.e. before a RETURN command is encountered). If you want to prevent this, or simply want to deactivate a global hot key temporarily, use the following programming technique:

```
cooloff = chr(windno) + cvi(event)
sysvar(100) = sysvar(100) + cooloff
```

where

windno	is the number of the window that contains the book with the @k<event> page

event	is the number of the hot key event to disable

So, to prevent the Alt+g glossary hot key of the previous example from being reactive in window 1, you would perform the following calculation on the @k4071 page:

```
cooloff = chr(1) + cvi(4071)
sysvar(100) = sysvar(100) + cooloff
```

When you are ready to re-enable the Alt+g global hot key, use the following A-pex3 program:

```
heatup = chr(1) + cvi(4071)   $$ key to re-enable
found = 0
DO
  found = found + 1
  found = (sysvar(100) $- found) $* heatup
  IF found = 0 THEN OUTLOOP         $$ not found!
  IF found ^/ 3 = 1 THEN OUTLOOP    $$ found
LOOP
IF found > 0 THEN              $$ if found, remove heatup
  extra = sysvar(100) $- (found + 3)
  sysvar(100) = sysvar(100) $\ (found - 1) $+ extra
ENDIF
```

The program above searches for the hot key code, and extracts it from Sysvar(100). You can find more about programming in the A-pex3 Programming chapter.

5.15 Event Handling While Branching

Depending on the complexity of your pages and the speed of the computer, branching from one page to another at run time can require anywhere from a fraction of a second to several seconds. During this brief time period, the user might press keys on the keyboard. Or, objects might generate events.

Potentially, these "in-between pages" interruptions can produce confusion. For example, if keypresses are processed as they occur, and if the user presses a key while branching from page1 to page2, the Wait object from page1 might process it, or the Wait object from page2 might...it would depend on which Wait object was active at that exact moment.

Fortunately, you can influence how Everest handles such interruptions. By default, it queues user keypresses until the next Wait object, and allows events to be processed as they occur. This approach works best for most projects. However, you can change the approach if you prefer.

To change how Everest handles "in-between" keypresses, use a calculation in your project (a Program object in the @start page, or the @define Program object are good places) to set the value of Sysvar(156) to one of those shown below:

-1	queue keypresses (the default)
0	process keypresses as they occur (not recommended)
1	discard keypresses

To change how Everest handles "in-between" events, use a calculation to set the value of Sysvar(158) to one of those shown below:

-1	queue events (numeric only, discard others)
0	process events as they occur (the default)
1	discard events and perform Ext(101) (Windows catch-up)
2	discard events

The following A-pex3 example tells Everest to discard keypresses and discard events that occur while branching between pages:

```
sysvar(156) = 1: sysvar(158) = 1
```

Your project is allowed to change the values in Sysvar(156) and Sysvar(158) as frequently as needed (such as for one special section of the project, etc.).

6 A-pex3 Programming

6.1 What Is Programming?

Programming is the means by which you issue detailed commands to the computer, manipulate the value of object attributes and change variables. A program is a list of instructions for the computer to perform.

In a sense, when you drag and drop icons into the Book Editor, you are building a program of sorts. The page contains the instructions that put objects in a window. Like a Program, the page is executed line by line, from top to bottom.

However, when people use the term "programming" they usually mean something at a lower-level...something that lets you get behind the scenes and do more nitty-gritty things. That's the definition we'll use here.

A-pex3

Everest users tell us that they create approximately 90% of their projects visually (via the icon drag-and-drop approach). But, when you need the flexibility to go beyond the icons, Everest has the power of a built-in language named A-pex3. The structure and syntax of A-pex3 is similar to that of Microsoft's Visual Basic (VB), so it is easy to learn and help is widely available.

6.2 Where Can I Program In Everest?

In Everest you can put programming instructions in a variety of locations. Extensive (i.e. large or complex) programming should be placed into a Program object. After you drag a Program object to a page, double-click on its line in the Book Editor to open the A-pex3 code window. It's a free-form text editor into which you can type your program.

Here's an example of a simple two-line program:

```
tries = tries + 1
IF tries > 3 THEN BRANCH feedback
```

Tiny one-line programs can be entered in the xxxAction attributes of the Wait object, such as Other1Action, and all xxxEvent attributes, such as ClickEvent.

You can put multiple commands on the same line; simply separate them with a colon (:).

A-pex3 Assistant

Everest comes with a built-in wizard, called the A-pex3 Assistant, that helps you construct simple programs, line by line. You can invoke the Assistant from both the Attributes window and the A-pex3 Program editor. Choose Assistant from the pull-down menu, or press F12.

The A-pex3 Assistant guides you through creating individual lines of programming. As you make choices from the lists provided, the Assistant displays the line of programming that you are constructing. The Assistant's OK button is enabled only if the line of programming is syntactically correct. Click OK when ready to insert the programming into the current attribute or Program.

The Assistant will not help you construct all possible programming statements, but many people find it helpful while they are learning the language. Give it a try.

Compiling Your Programs

Everest contains a "just-in-time" compiler that transparently and automatically compiles your A-pex3 Program objects at run time. Compiling means that Everest internally translates your Programs into a form that the computer can more easily understand. Compiled programs run faster than non-compiled ones. Since the compilation happens automatically, there is nothing you need to do to enable it. It happens so quickly that you probably won't even notice.

Don't Compile

In the rare situation that you receive an error from the compiler, or suspect the compiler is doing something wrong, you can tell Everest to not compile a particular Program object. This can help you debug the problem. To do so, put the following directive at the very top of the Program:

```
$$nocompile
```

The $$nocompile directive tells Everest to run the Program via its interpreter. The interpreter is a slower (and older) way to run Programs.

6.3 A-pex3 Command List

Commands make up a large percentage of the items in an A-pex3 program. Here is a list and a brief description of each:

Branching

BRANCH	go to another page
CALL	execute another page as a subroutine
GOSUB	execute a Program object as a subroutine
GOTO	redirect Program execution to a LABEL
JUMP	redirect Page execution to a JLabel Object
LABEL	mark the destination of a GOTO
OPEN	display a page in another window
RETURN	exit from a subroutine executed via CALL

Conditional

ELSE	otherwise clause in IF block
ELSEIF	test another condition in an IF block
ENDIF	end an IF block
IF	test a condition
THEN	mark the end of the condition in an IF or ELSEIF

Graphics

ARROW	draw an arrow
BOX	draw a rectangle
CIRCLE	draw an ellipse
COLOR	set color of graphics
FBOX	draw a filled rectangle
FONT	choose font of subsequent PRINT text
GFILL	fill with a graduated color
LINE	draw a line
LPRINT	send text to the printer
PAINT	fill an enclosed area with a color
POINT	draw a dot
POLY	draw a closed polygon
PRINT	display text
RBOX	draw a box with rounded corners
SCALE	customize window coordinate system
STYLE	set various graphics appearance attributes

Loop

DO	begin loop structure
LOOP	mark the end of a loop structure
OUTLOOP	exit from a loop structure
RELOOP	go back to the start of a loop structure

Other

DELVAR	delete variable or array
DIM	allocate an array
DPRINT	display text in Debug window
ERASE	remove object from window, and/or erase graphics
PAUSE	wait for a time period
REDIM	change the number of elements in an array
STEP	engage/disengage A-pex3 code step mode

Details about each command can be found in the technical reference/on-line help.

6.4 Simple Branching

Perhaps the most commonly used A-pex3 command is BRANCH. BRANCH causes the execution of your project to continue at a different page...BRANCH is the way your project "goes to" the next page.

Explicit BRANCHing

BRANCH is so simple to use, it hardly qualifies as programming, All you do is type:

```
BRANCH <page>
```

where <page> is the name of the page to which to branch. For example:

```
BRANCH page2
```

Flowchart BRANCHing

If you simply want to branch to the next page that appears in the Book Editor, you do not need to specify it by name. Instead, simply use the special @next keyword:

```
BRANCH @next
```

Similarly, if you want to branch to the page the appears above the current page in the Book Editor, you can use the special @back keyword:

```
BRANCH @back
```

If you plan to deliver your projects via the Internet, you will probably make them "granular," that is, break them into very small books. In such cases, avoid using BRANCH @next and BRANCH @back.

Where Does BRANCH Belong?

Where do you put the BRANCH command, and what triggers it? Most authors put BRANCH commands in the NextAction attribute of a Wait object. The Wait object waits for user input (an event) and takes the corresponding action. See the prior chapter for details about handling events. You can also place BRANCH commands in Program objects.

Why BRANCH is Best in NextAction

BRANCH can be used in several places within a given page. In fact, most pages BRANCH to several others, depending upon the user's action. In such a case, determine which of these page the user is most likely to visit next (usually, the user simply moves forward in your project, so the most likely page is the next one). Place the BRANCH command for this page in the Wait object's NextAction attribute.

We recommend doing so because, at run time, Everest preloads the page you specify in the NextAction attribute. This preloading feature can greatly enhance the execution speed of your project, particularly if it is being executed via the Internet.

6.5 Call and Return

The BRANCH command causes execution to go to another page, and continue branching from there. Sometimes it is useful to go to another page, and come back when done. This is what the CALL command does.

Technically, CALL runs a page as a subroutine. Everest remembers the location of the CALL, and returns to it when the other page performs a RETURN command. Like BRANCH, CALL is commonly used in the Wait object, or in Program objects.

It is important that your project eventually RETURN from a CALL. In the example above, if you were to BRANCH back to the mainmenu, rather use the RETURN command, Everest would think the CALL was still active. That would tie up memory. Always eventually RETURN from CALLs.

We recommend that you do not CALL pages that contain Wait objects; instead, use the OPEN command. The reason for this recommendation is that once you allow the user control of the page (via a Wait object) his/her actions might make it difficult to know when to RETURN. For example, if the user manually closes a window that was opened via CALL, the RETURN command might never be performed.

Everest remembers where to return from a CALL via the CALL stack in Sysvar(58). When Len(Sysvar(58)) = 0 there are no active CALLs.

6.6 Branching Across Books

By default, both the BRANCH and CALL commands look for the destination page in the current book (.ESL file). But what if you want to execute a page in a different book? The answer is: prefix the page name with the book name and a semicolon. For example:

```
BRANCH lesson2;intro
```

or

```
CALL lesson2;intro
```

which loads the book LESSON2.ESL (if it is not the one already in memory) and branches to the page named intro.

If the new book is located in a different subdirectory (i.e. not that of the current book), prefix the book name with a disk path. For example:

```
BRANCH C:\math\lesson2;intro
```

If you won't know on which disk drive the user will install your project, use:

```
BRANCH ?:\physics\lesson1;intro
```

which keeps the same disk drive as the current book.

Alternatively, if you want to employ the DOS default drive, use @ in place of the drive letter. For example:

```
BRANCH @:\physics\lesson1;intro
```

If you want to employ a path the StarPath entry in the EVEREST.INI file (a path your project's end user can edit easily), use * in place of the drive letter or path. For example:

```
BRANCH *:lesson1;intro
```

6.7 Branching to Multiple Windows

By default, at run time, Everest displays your pages in a single window (known as window number 1). To open another window, suffix the page name in a branching instruction with a window number surrounded by square brackets. For example:

```
BRANCH help[2]
```

```
CALL help[2]
```

or

```
OPEN help[2]
```

opens window number 2 (if not already open) and displays the page named help in it. The window number can range from 1 to 8, and need not be used in consecutive order. Up to 8 windows can be open simultaneously, however each consumes Windows resources; available resources can be monitored via the Fre() function.

See the technical reference for a description of the differences between BRANCH, CALL and OPEN.

You can express the window number with a variable. To do so, surround the variable name with { }. For example:

```
BRANCH help[{windno}]
```

Multiple windows give the appearance of running multiple projects. Each window operates independently of the others. More than one window can use a given book, or they can all use different books.

The user can highlight a different window simply by clicking on it. Everest stores the number of the currently highlighted window in the Sysvar(8) variable. Everest keeps track of all open windows via bits in the Sysvar(52) variable. The following program determines which windows are currently open:

```
windno = -2: openlist = ""
DO
  IF sysvar(52) ^? (windno+2) > 0 THEN
    openlist = openlist $+ windno $+ "   "
  ENDIF
  windno++
LOOP IF windno < 8
dummyvar = mbx("Open windows: " + openlist)
```

6.8 Using Object Attributes at Run Time

As you know, objects have attributes. At design time, you set the attributes to the desired values via the Attributes window. You can also read and change most object attributes at run time (those you cannot are so noted in the technical reference).

To read the value of an attribute in A-pex3 programming, you refer to the object's Class name, IDNumber and attribute, as in the following example:

```
IF Button(1).Value = -1 THEN x = x + 1
```

Button is the Class name of the Button object, the number 1 is the IDNumber, and Value is the name of an attribute. The IDNumber uniquely identifies the object within its class in the window.

Alternatively, if the object is located within the current page, you can refer to it by its Name attribute. This could resemble:

```
IF page2_button_A.Value = -1 THEN x = x + 1
```

If you plan to refer to objects by Name, consider changing the default name Everest assigns to one that is more meaningful. You can do so via the Book Editor's Edit menu, Rename feature.

Changing an Attribute

To change an attribute at run time, simply use it on the left side in an assignment statement. Examples:

```
Button(1).Value = 0
MenuButton.Value = 0          $$ by Name
Button(1).Visible = "Yes"
Button(1).Visible = -1        $$ faster than "Yes"
```

At run time, attributes are accessible from A-pex3 programming only for those objects that exist in the window at that time. Recall that pages are executed from top to bottom. If you refer to Button(1).Value in a Program object positioned before the Button object with IDNumber 1 (and there is no Button with IDNumber 1 already in the window from a prior page), Everest will report error 340. The solution is to drag the Program object to a location in the page after the Button.

6.9 Communicating Between Windows

When multiple windows are open, you may want to send information from one to the other. You can employ Everest variables (all are global in scope) to do so. Or, you can refer to object attributes.

In A-pex3 programs, Everest assumes objects to which you refer are located in the currently highlighted window. To refer to an object in a different window, prefix the object name as in this example:

```
Textbox(1).Text = Window(2)!Textbox(1).Text
```

which copies the text contents of a Textbox in window 2 into a Textbox in window 1.

If desired, you can express the window number via a variable or expression. For convenience, window number 0 refers to the currently highlighted window.

Since the user can click on a window to move the highlight to it at any time, you might be justifiably concerned that he will do so while your A-pex3 code is executing, which could result in programming references to attributes of the wrong object. However, this is not a problem because Everest does not actually move the highlight from one window to another until encountering either a Wait object in your page's Book Editor, or function Ext(101) in your project.

6.10 Closing Windows

You can close a window by referencing the Destroy attribute as in the following example:

```
ok = Window(2).Destroy
```

which closes window number 2. Upon closure, Everest removes all objects in the window from memory.

When a highlighted window is closed, the highlight moves to another. When the last window is closed, Everest treats it as a log out. Closing a window does not renumber the windows that are still open.

Remember that a user can close a window manually via its Control box. If you want to prevent this, remove the window's Control box via a Layout object's ControlBox attribute.

If you used a CALL command to open the window, you can close it via the RETURN command, but only if you include a space plus the optional window number parameter. For example:

```
RETURN [2]
```

returns from the CALL, and closes window number 2. If you simply want to close the current window, you can do:

```
RETURN [0]
```

Hiding a Window

Sometimes you might want to temporarily hide a window from the user to prevent his/her interaction with it. Rather than destroying the window and later reopening it, consider moving it off the visible area of the monitor. For example:

```
oldleft = Window(1).Left: Window(1).Left = -10000
```

moves window number 1 off the left edge of the monitor. Later, to restore the window's original position:

```
Window(1).Left = oldleft
```

6.11 Using GOTO, JUMP and GOSUB

The GOTO, JUMP and GOSUB commands provide other ways to control the flow of a project. The primary difference between them is scope.

GOTO

The GOTO command redirects program execution to a LABEL with the corresponding name. Both the LABEL and GOTO must appear in the same Program object. Label names can be of any length, and case is not significant. GOTO can only be used within Program objects.

GOTO also supports a keyword:

@exitprog	Ends execution of the current Program.

JUMP

JUMP also redirects execution, but on a broader scope. JUMP jumps to a JLabel object in the current page's Book Editor. In the JUMP command be sure to enter the full name of the JLabel (such as page1_jlabel_A). You might consider manually renaming the JLabel to assign it a more meaningful name (to do so, modify the JLabel's Name attribute via the Book Editor's Edit menu, Rename feature).

JUMP also supports two keywords:

@proceed	Causes execution to continue with the next object in the page. Handy in xxxAction attributes of Wait objects to force continuation of the page after performing another command.
@wait	Tells Everest to scan backwards in the page from the current object until it finds a Wait object, and wait there.

GOSUB

GOSUB performs a slightly different action. It executes a Program object as a subroutine. That means GOSUB runs the Program object, waits for that program to finish, then returns. GOSUBs can be nested up to the limits of memory (but we don't recommend more than 8 levels). Also, you should avoid nested GOSUBs if you will be making your project granular for delivery via the Internet or an intranet.

The Program object that is GOSUBbed to need not be in the current page. However, if it is, Everest will execute it more quickly.

IMPORTANT: Before GOSUBbing to a Program object located in a different page, be sure that Program object's SaveAsObject attribute is set to Yes. Some authors put all frequently used Program objects into one page for easy reference.

Most authors put the GOSUB command on a line of its own. But, if you want to put additional commands after it on the same line, you must separate them from the GOSUB with a colon and a space. For example:

```
GOSUB allsubs_program_A: count = count + 1
```

Inter-Book GOSUB

To GOSUB to a program object in a different book, prefix the object name with the book name. The following example executes the Program object named initvars in the book named central

```
GOSUB central;initvars
```

Program objects located in other books should not execute BRANCH, CALL, JUMP or RETURN commands.

6.12 Literals and Variables

Literals

Literals are either numbers or strings of characters that Everest interprets directly. In the following programming statements, the number -1 and the string Yes are literals.

```
Button(1).Visible = -1        $$ faster than "Yes"
Button(1).Visible = "Yes"
```

Note that literal character strings in Everest must be surrounded by double quotation marks.

Variables

A variable acts as a holder of information, be it either a number or a string of characters. All variables have names that you assign. When your programs run, Everest knows to substitute the value of the variable when needed. In the following programming, the word "showing" is a variable

```
Button(1).Visible = showing
```

All Everest variables are global; that means all can be accessed from anywhere in your project. There are two types of variables: author defined and system. Author defined variables are those you create. System variables are created automatically by Everest to hold information about the project.

Variable Names

You decide the names of your author defined variables. Names always start with a letter from a to z, and can contain letters from a to z, digits from 0 to 9, and the underline character (ASCII 95). The first eight characters in a variable's name are significant; that is, Everest ignores characters after the eighth. The case of letters is not significant.

The name of a variable cannot be a reserved word. Reserved words include all Everest commands (such as GOSUB), functions (such as Len), and Sysvar (which is used by system variables). Your project can contain up to 2500 uniquely named variables.

Defining Variables

Everest automatically defines variables upon their first use and initializes them to contain a null string. Alternatively, you can predefine variables. This is described in a following topic.

Contents

Variables can contain numbers from -3 x 10^38 to 3 x 10^38 or variable length strings up to 32,000 characters in length. The type of variable (numeric or string) is implied by its contents. Everest automatically converts numbers to strings, and vice versa, as needed by the usage context.

During project execution, the total amount of memory occupied by character strings in author defined variables is limited only by the memory available to Windows. However, if your project employs user records, at the time of user log off (bookmark writing) there is a limit of 64,000 bytes (characters) total.

Examples

To store literals in variables, use calculations such as:

```
barometer = 29.92
```

```
tendency = "rising"
```

Notice that character strings are surrounded with quotes, and numeric values are not. Calculations can be placed in Program objects, or in the xxxAction attributes of Wait objects.

You can place more than one calculation (or command) on a line. The previous example could be written:

```
barometer = 29.92: tendency = "rising"
```

Use a colon (:) as a separator. Placing multiple calculations and commands on one line makes your programming more difficult to read and debug, so avoid it where you can. A place you cannot avoid doing so is in the xxxAction attributes which have only one line available. The maximum length of each xxxAction attribute is 250 characters.

6.13 Arrays

Arrays are groups of variables with a common name. You access members of the group via a numeric index surrounded by parentheses; members are called "elements." Arrays have the same naming restrictions as do variables (see the previous topic). An array cannot have the same name as an existing non-array variable.

The REDIM Command

Most authors allocate (tell Everest how many members there will be in) arrays via the REDIM command. If you do not use REDIM, Everest allocates as many elements as are needed in the first reference to the array at run time. The following A-pex3 programming example allocates an array, and loads one letter of the alphabet into each element:

```
REDIM alphabet(26)
element = 1
DO
  alphabet(element) = chr(64 + element)
  element++
LOOP IF element < 26
```

Some other helpful commands and functions are (see the technical reference): DELVAR, DIM, Arr(), Lod(), Srt(), Sum().

The maximum number of elements in an array is 5000; multi-dimensional arrays are not currently supported. For compatibility with Summit for DOS, arrays with single letter names always contain at least 32 elements, plus a 0th element.

6.14 Predefining Variables and Arrays

While debugging your project, you might run one page separately from the others. If a Program object in that page uses an array that has been dimensioned on another page, Everest is likely to complain about an insufficient number of elements in the array, or a similar error. The solution is described below.

Predefining with @define

The best solution is to predefine your arrays. This is done by creating a Program object with the special name "@define" in any page in your book. Here are the steps:

1) Drag a Program object into any page in your book; many authors use the same page where they keep all Program objects that are used as GOSUB subroutines.

2) The name of the page is not significant, but the name of the Program object is. Everest assigns the Program object a default name (something like allsubs_program_A). Change the name attribute to @define manually via the Book Editor's Edit menu, Rename feature.

3) IMPORTANT: Set the Program object's SaveAsObject attribute to Yes.

4) Double click on the Program object in the Book Editor to open the A-pex3 program editor window.

5) Enter the array and variable declarations you want. You can even assign values to variables and array elements that are acting as constants; use the same syntax as in a regular Program object.

@define Execution

Each time you Preview a single page, run a lesson in debug mode, or run the lesson as a user, Everest automatically performs the @define Program object upon starting. By placing your DIM and REDIM commands in @define, your arrays will always be allocated.

Note that Everest also performs the @define whenever it loads or reloads a book. This occurs when you branch to a page in a different book, or open a new window. Since Everest performs the @define automatically, it is not necessary to place it on a page that gets executed. In fact, we recommend that you avoid doing so.

If you edit A-pex3 code to increase the number of elements in an array dimensioned with the DIM command, before re-running your project in debug mode, you should manually reset all variables. Do so via the Reset button in the Variables window. This is necessary because Everest ignores DIM commands for arrays that already exist. Alternatively, consider using REDIM in place of DIM.

6.15 System Variables

System variables are those that Everest allocates and uses. They contain information such as the name of the current page, the location of the mouse cursor, etc. All system variables are stored in an array named Sysvar.

Most system variables are read only. That means you should not change the value of the variable; if you do so, unpredictable results can occur. The contents of system variables are listed in the technical reference.

6.16 Viewing Variables at Run Time

The situation: you are testing your project. It is calculating the user's score on an exam. You display the score on the page: 167%? Somewhere a calculation went awry. Now you have to find where.

One of the most helpful debugging features of Everest is the ability to view the contents of variables at run time. From the main authoring Window pull-down menu, choose Variables. The names of all variables are listed. Click on a name to view its contents. You can edit the contents of a variable by clicking on the box that displays its contents.

Everest preserves the contents of variables while you are editing your project. The only time this might create difficulty is when you change the size of an array. Recall that Everest ignores DIM commands if the array already exists. To manually reset all author defined variables (i.e. delete them), click the "Reset All" button found in the Variables window.

6.17 Displaying Variables at Run Time

You might want to display the contents of a variable, such as an exam score, to the user when your project runs. To do so, simply enter the name of the variable surrounded by { } in a Textbox. For example, the contents of the Textbox at design time might resemble:

Exam complete.

Your score is: {scorevar}%

where scorevar is the name of the variable in which you previously stored the exam score.

This technique is called "embedding" variables. It can be used anywhere a non-quoted string is employed in Everest. For example, in the Attributes window for a Layout object, you can enter {scorevar} as the Caption attribute. At run time, Everest will display the contents of the scorevar variable as the Caption of the window.

Note that numeric attributes (such as Left, Top, Width, Height, etc.) cannot be set to variables in the Attributes window. You can, however, use a calculation in a Program object at run time to set these attributes.

Expressions as well as variables can be embedded for display. For example, in a Textbox:

All 10 exams complete.

Your average score is: {sum(scores(10))\10}%

Insert vs. Replace

You can tell Everest to either insert the value of the expression into the line of text (and automatically shift subsequent text left or right as needed), or replace the expression with the value (and do NOT shift subsequent text). To insert the value, use a lower-case letter after the {. To replace, use an upper-case letter immediately after {.

For example, at run time the expression:

```
{sum(scores(10))\10}%
```

could display a value of:

```
90%
```

But, if instead you type the expression with an upper-case letter, as in:

```
{Sum(scores(10))\10}%
```

Then at run time, Everest will display:

```
90                  %
```

Authors find that they use the insert approach most often. About the only time you use the replace approach is when you are trying to keep text aligned in particular columns.

When embedding variables in branching commands, we recommend you employ the insert approach (lower-case letters for variable names) to avoid confusing Everest with extra spaces in the page name.

6.18 Expressions and Operators

Constants and variables can be combined and manipulated in expressions via Operators. Details about Operators can be found in the technical reference/on-line help. Only a list of operators and some important subtle points appear here.

Numeric Operators

+	addition
-	subtraction
*	multiplication
/	division
\	integer division
$	string creation
^	exponentiation
^/	modulo
^\	step function
^?	bit test (returns 0 or 1)
++	increment (numeric variable)
--	decrement (numeric variable)
=	assignment

String Operators

+	concatenation
-	mid parse
*	search
/	right parse
\	left parse
$	string creation
^	replacement (see Notes below)
^*	count occurrences
^^	one middle character
^#	remove one or more characters
=	assignment

Operator Examples

Operators are represented in A-pex3 by the symbols shown above. For example, the addition operator is +. When + is used for two numeric values, Everest adds the two and returns the result. For example, the expression:

```
sum2 = 7 + 2
```

stores the value 9 in the variable named sum2.

The + operator can also be used with character strings to perform string concatenation. For example, the expression:

```
sport = "foot" + "ball"
```

stores the character string football in the variable named sport.

Everest determines at run time whether numeric addition or string concatenation should be performed by examining the operands (the values being operated upon). That's fine, except you must be careful to avoid ambiguity. For example, what is the result of the following expression?

```
tall = 7 + "foot"
```

Answer: the variable named tall is set to the character string 7foot. If either operand is a string, Everest performs string concatenation. What about the following?

```
complex = 7 + 2 + "feet"
```

Answer: the character string 9feet. Everest processes expressions from left to right, and performs operations on the pairs of operands as it goes. So, in this example, 7 + 2 are added numerically, then the string feet is concatenated.

This same left-to-right processing can be observed in the following expression that employs the * multiplication operator:

```
tricky = 3 + 4 * 2
```

Your algebra teacher says the result of this calculation is 11 (because multiplication has a higher precedence than addition, and is therefore computed first). However, Everest says the result is 14, because it calculates from left to right. If necessary, you can force precedence via parentheses. For example:

```
tricky = 3 + (4 * 2)
```

and now Everest and the algebra teacher agree on 11. Parentheses can be nested up to eight levels deep.

Relational Operators

=	equals
>	greater than
<	less than
#	does not equal
>=	greater than or equal to
<=	less than or equal to
=E=	equivalent (ignore case and spaces)
#E#	not equivalent
=P=	pattern match
#P#	not pattern match
=S=	phonetic sound-alike
#S#	not phonetic sound-alike
=T=	inside region
#T#	not inside region
=W=	word search
#W#	not word search

Relational operators are used in conditional expressions, such as those starting with the IF command. Some examples follow.

To test if a variable contains a numeric constant:

```
IF x = 1 THEN
```

To test if the contents of a variable match another:

```
IF x = y THEN
```

To test if the value of a variable is greater than a numeric constant:

```
IF x > 1 THEN
```

To test if the value of a variable is less than or equal to an expression:

```
IF x <= y * 2 THEN
```

To test if the string contents of a variable come alphabetically after a constant:

```
IF x > "m" THEN
```

To test if the contents of a variable phonetically sounds like a constant:

```
IF x =S= "machine" THEN
```

To test if the contents of a variable do not contain a particular word:

```
IF x #W# "everest" THEN
```

To test if the contents of a variable is a single letter that is a vowel:

```
IF x =P= "[aeiouAEIOU]" THEN
```

At Run Time

To evaluate an expression at run time, use Ext() function operation 125. For example

```
result = ext(125, "1+2")
```

stores the value 3 in the variable named result.

6.19 Strings vs. Numbers

There are situations in which you might want to force Everest to perform string concatenation rather than numeric addition. For example, say your project contains a very long survey, perhaps 300 questions, each of which is a multiple choice question with numbered choices. You want to save all the user's responses, but do not wish to store them in an array. So, after each question, you can concatenate the user's response into a long string of digits. You might try:

```
survey = survey + response
```

But, if both the survey and response variables contain a digit, Everest will perform numeric addition! There is an easy solution, prefix the operator with a $:

```
survey = survey $+ response
```

which tells Everest to handle the operands as strings, even if they are numbers.

Possible Overflow

However, there is another problem. While handling that long series of digits in the survey variable, Everest checks to see if it can be interpreted as a number. At some point after you have concatenated several digits, Everest will think the number is too large, and will either return an Overflow error or convert the number to scientific format.

The solution is to make Everest know that the survey variable contains a character string, not a big number. The easiest way is to put a dummy non-numeric character at the start of the variable. For example, before the start of the 300 questions, you might initialize the survey variable with a calculation such as:

```
survey = "x"
```

and concatenate the responses normally. Now, when Everest looks at the survey variable, it will see an x at the start, and therefore treat the whole variable as a string. Just be sure to allow for the x in any subsequent programming involving that variable.

What might you do with a string of 300 responses? You might count how many of each response (1 to 9) there are via the ^* "count occurrences" operator. Here's a program that does so, and stores the results in the howmany array:

```
DIM howmany(9)
which = 1
DO
  howmany(which) = survey ^* which
  which++         $$ faster than which = which + 1
LOOP IF which < 9
```

6.20 Using Functions

Functions are like built-in subroutines. You give a number or string to a function, it does some calculating for you, and returns a result. In Everest, all functions have 3 letter names. The value you give to a function is called a parameter.

One of the simplest functions is Len(). It returns the length of (number of characters in) a string parameter. For example:

```
numchars = Len("Everest")
```

sets the numchars variable to 7 because there are seven characters in the "Everest" parameter.

All functions return some sort of information (a number or a string). Therefore, it is imperative that you specify a place for the information to go (a variable, attribute, etc.). For example, the following has an illegal syntax:

```
Ext(19)                 $$ improper syntax!
```

but the following is OK:

```
dummyvar = Ext(19)      $$ correct syntax
```

A list of functions follows.

Numeric

Abs()	absolute value
Atn()	arc tangent
Cos()	cosine
Hex()	hexadecimal
Log()	natural logarithm
Lwr()	round down to nearest integer
Rgb()	combines red, green and blue
Rnd()	random number
Sel()	unique random number
Sin()	sine
Sqr()	square root
Tan()	tangent
Upr()	round to nearest integer

String

Asc()	ASCII value
Chr()	character
Fmt()	formatting for output
Len()	string length
Ltr()	remove leading blanks
Lwr()	convert to lower case
Mid()	substrings
Pik()	return one item from a list
Rpl()	replace or remove characters
Rtr()	remove trailing blanks
Upr()	convert to upper case
Val()	convert to numeric
Wrp()	word wrap

Variables and Arrays

Arr()	number of elements in an array
Lod()	copy values in an array
Srt()	sort an array
Sum()	sum elements of an array
Typ()	storage format of variable contents
Var()	test if variable exists

Disk/Operating System

Env()	environment
Fnt()	fonts available
Fre()	free memory
Fyl()	read/write disk files
Gdc()	Windows GetDeviceCaps
Gsm()	Windows GetSystemMetrics
Hlp()	Windows Help system
Ini()	read/write .INI files
Msg()	read EVEREST.MSG file
Pth()	disk path/file name parser
Rec()	read/write records files
Scn()	page exists

External Devices and Applications

Dde()	Windows Dynamic Data Exchange
Dll()	call a DLL routine
Mci()	Windows Media Control Interface
Mse()	mouse
Ply()	play musical notes
Shl()	execute external application

Miscellaneous

Cvi()	convert 2-character string to integer
Dat()	current date
Ext()	assorted
Ibx()	user input box
Key()	keyboard
Mbx()	message box
Mki()	convert integer into a 2-character string
Obj()	Everest objects
Reg()	regions
Tim()	current time

Details about each function can be found in the technical reference/on-line help.

Functions can be nested up to 8 levels deep. Nesting means using one function within another. For example, the following calculation uses functions nested several levels deep:

```
length = Len(Rtr(Ltr(response)))
```

6.21 Using IF Commands

The IF command is the basic method of testing a condition for validity. The IF command has two forms: single line and block. The syntax of each can be found in the technical reference. The following two examples do the same thing:

Single-Line IF

```
IF score < 70 THEN result = "failed"
```

Block IF

```
IF score < 70 THEN
  result = "failed"
ENDIF
```

With ELSEIF and ELSE

Block IFs allow an unlimited number of ELSEIF clauses. For example:

```
IF score < 70 THEN
  result = "failed"
ELSEIF score < 80 THEN
  result = "passed"
ELSEIF score < 90 THEN
  result = "good"
ELSE
  result = "excellent"
ENDIF
```

Single-line ELSE

The single-line version of the IF allows an ELSE clause (but not an ELSEIF). For example:

```
IF score<70 THEN grade="failed" ELSE grade="passed"
```

Multiple IF Conditions

You can test multiple conditions with an IF. To do so, separate the conditions with either of the symbols:

@	or
&	and

Examples:

```
IF count = 1 @ count = 2 THEN $$ either 1 or 2

IF age >= 30 & age < 40 THEN  $$ age is thirty something
```

IF Limitations

Some restrictions: LABEL commands are not allowed inside an IF block; conditions cannot be grouped in the IF (Everest processes them left to right).

6.22 Using DO...LOOP Commands

The DO...LOOP structure is handy when you need to execute a series of commands more than once. Everest performs the commands inside the structure until one of the following: 1) the IF specified on the same line as the DO is not true, 2) the IF specified on the same line as the LOOP is not true, 3) an OUTLOOP command is encountered.

The following example uses the built-in Ibx() input box function to obtain a response from the user, and checks that the response is within the range 1 to 4:

```
DO
  resp = Ibx("Enter a number from 1 to 4")
LOOP IF resp < 1 @ resp > 4
```

DO...LOOPs can be nested to eight levels deep. If you use a LABEL inside a DO...LOOP, do not GOTO it from outside the loop. To exit a loop, use OUTLOOP; do not use GOTO. To restart the loop from inside of it (and test the IF, if any, on the same line as the DO) use RELOOP.

6.23 Commenting Your Source Code

You can include comments on any line in a program. Simply prefix them with $$. Examples:

```
ptr = ptr + 1           $$ point to next item
```

```
cnt++        $$ increment item counter
```

6.24 Adding Objects at Run Time

For ultimate control of the contents of a window, you can add objects to it at run time. This is fairly complex, so it is best done by experienced programmers only. In theory, you could use this approach to build an entire project that contained only one startup page (and lots of A-pex3 programming code...each Program object can handle up to 32,000 characters).

To add an object, reference the Create attribute as in this example:

```
ok = Textbox(id).Create
```

where id is the IDNumber of the TextBox to create. Note that the object name is on the right side of the =, and a variable is on the left. This order is required for proper object creation.

If the object is created without error, the variable on the left side is set to -1. Otherwise, it is set to a numeric error code. An error will occur, for example, if the IDNumber is already in use in that object class.

The new object starts life in a pristine state. In fact, it is not even enabled or visible. Typically, your code will resemble:

```
ok = Textbox(1).Create
IF ok = -1 THEN                     $$ if no error
  Textbox(1).Move = "50,10,300,100"
  Textbox(1).ForeColor = 255
  Textbox(1).Text = "I love to write lots of code."
  Textbox(1).Enabled = -1
  Textbox(1).Visible = -1
ELSE
  dummyvar = Mbx("Error #" + ok + " upon create!")
ENDIF
```

As you can see, any attribute that is important for your application must be set in code. The Move attribute is a handy way to set the Left, Top, Width and Height attributes in one step.

6.25 Removing Objects at Run Time

Removing objects from with window via A-pex3 programming is much easier than adding new objects. To remove a particular object, reference its Destroy attribute in the following manner:

```
ok = Textbox(id).Destroy
```

where id is the IDNumber of the object. As with the Create attribute, Destroy appears on the right side of the =, and a variable on the left. Error codes are returned in the variable, or -1 for no error.

To quickly remove ALL objects from the currently highlighted window, use the ERASE command. Both ERASE (2) and ERASE (3) remove all the objects from the window.

You can also use Destroy to close a window. Everest automatically removes all objects from the window before closing it in order to release the memory occupied by those objects. To close window #2:

```
ok = Window(2).Destroy
```

6.26 Monitoring Objects via Obj()

If you are adding and removing objects at run time, it can be difficult to track the IDNumbers in use in the various object classes. Fortunately, Everest does so for you, and provides this information via the Obj() function.

The Obj() function can return a string of characters whose ASCII codes represent the IDNumbers of objects in the specified class. This string can be parsed via the ^^ operator and converted to a number via the Asc() function. This process is demonstrated in the following example that removes all Textbox objects from window #2:

```
ids = Obj(1, "Textbox",, 2)
idptr = 1
DO IF idptr <= Len(ids)
  thisid = Asc(ids ^^ idptr)
  ok = Window(2)!Textbox(thisid).Destroy
  IF ok # -1 THEN
    ok = Mbx("Error #"+ok+" Textbox("+thisid+")")
  ENDIF
  idptr = idptr + 1
LOOP
```

Note that the Obj() function needs the class name of the object as the second parameter. For easy reference, the class name of an object is displayed at the top of the ToolSet when you click once on its icon.

6.27 Debugging Your Programs

Even the best programmers make mistakes in their code. Fixing the problem is almost always easy. Finding the problem...well that's often quite another story. Everest offers several tools to help you find the mistakes.

First, there is the syntax checker in the A-pex3 code editor window. When you leave the editor after making a change, Everest scans your program and reports problems in syntax. It will discover variables with illegal names, illegal operators, misspelled commands, and more. These are typically easy to correct.

The Debug Window

But, the tough problems are execution errors: the program runs but does not do what you want. This is where the Debug window comes in handy. Open the Debug window by choosing it from the Windows list on the main menu.

The Debug window displays the name of the currently executing page and object. It also shows the A-pex3 programming command that is being processed, or was last processed. Often simply running your project with the Debug window open can reveal the source of trouble.

The Debug window will slow the execution of your project. When not needed, keep the Debug window closed.

6.28 Using the DPRINT Command

If your code flashes by too quickly in the Debug window, insert some DPRINT commands in it. DPRINT displays text (or a variable) in the Debug window. For example, insert:

```
DPRINT ("I'm here")
```

in a program and run it. When you see "I'm here" in the Debug window, you know your program executed at least that far. DPRINT is ignored when users run your project.

6.29 Using Step Mode

Another solution to code flashing by too quickly in the Debug window is the step mode feature. Step mode lets you execute one A-pex3 programming command at a time, and watch the contents of variables as you do.

You can engage step mode in any of several ways:

1) Select the Engage A-pex3 Step Mode item from the Debug window's Run menu. Then, run the page via the Author window's Run menu.
2) Press the Break key (the one labeled Pause on most keyboards) while the Debug window is open and an A-pex3 program is running.
3) Employ the STEP command in your program.
4) Enter a Break Expression in the Debug window (via its Settings pull-down menu). When the expression is true, Everest engages step mode.

When step mode is engaged, Everest pauses and displays the Debug window before performing each line of A-pex3 code. The caption of the Debug window includes the words "Step Mode" so you can distinguish step mode.

While paused, you cannot access features in other windows...only those in the Debug window itself. Via the pull-down menus, you can step through the code, halt execution, view and modify the contents of variables, and more.

6.30 Using Break Expressions

If you are trying to determine when a variable or expression reaches a certain value ("where does score get set to 167%?!?!") you can enter a Break Expression in the Debug window. Do so by selecting Break Expression from the Debug window's Settings pull-down menu.

An example of a valid Break Expression is:

```
score = 167
```

which tells Everest to check the score variable *before every line of A-pex3 code is executed*, and engage step mode as soon as it equals 167. As you might expect, a Break Expression will slow the execution of your project as Everest repeatedly checks its value. Even so, it is often the quickest way to find a problem.

Note that Everest checks the Break Expression only while the Debug window is open.

6.31 Using Watch Expressions

To continuously monitor the contents of a variable, or the result of a function, employ a Watch Expression. You can enter one by selecting Watch Expression from the Debug window's Settings pull-down menu.

An example of a Watch Expression is:

```
fre(0)
```

which displays and updates the amount of system resources still unused as you test run your project.

If you would like to watch several expressions at once, surround each with { }. For example:

```
{fre(0)}  {fre(1)}  {fre(2)}
```

6.32 More Debugging Hints

Another handy debugging technique is to add Mbx() function references in your code. The Mbx() function displays a message box, and waits for your input. By scattering them in your code, you can monitor how it is progressing. Use the opportunity to display the contents of a variable; for example:

```
DO IF counter < 10
  GOSUB routine
  dummyvar = mbx(counter)
  counter++
LOOP
```

To put special debugging code in your programs that will be ignored when the user runs it, reference Sysvar(15) (which contains 0 when debugging, and a value greater than 0 when the user runs the program). For example:

```
IF sysvar(15) = 0       $$ author run mode
  dummyvar = mbx("Result so far is: " + result)
ENDIF
```

If you still can't find the trouble, simplify your page. Use the Toggle feature in the Book Editor window to temporarily disable certain objects. Try building the page from scratch, testing it after every addition until the problem shows up again (then you know the last thing you added is the source of the problem).

7 Graphics

7.1 What Are Graphics?

Unlike in DOS, in Windows, everything is actually a graphic, even text. However, when most people think of graphics, they think of pictures and drawings. On computers, pictures are typically stored as "bitmaps." In a bitmap, the computer remembers the color of each dot that makes up the picture. In a drawing, the computer remembers the objects, such as lines and circles ("vectors"), in the image.

Everest supports both bitmap picture graphics and vector drawings. This chapter will help you decide where and when to use each.

7.2 Bitmaps vs. Drawings

Bitmapped Graphics

Bitmapped graphics do the best job of producing photographic quality images. Bitmapped graphics are produced in several ways: manually via a "paint" style graphics editor such as Windows Paintbrush, mechanically via an image scanner, or electronically via a digitizer. On a computer, bitmapped images are often saved in a file format that can be identified by the file name extension: .PCX. .BMP, .AVI, etc.

In Everest, the Picture, SPicture, Animate and PicBin objects all handle bitmapped graphics.

Vector Graphics

Drawings work well for charts and diagrams. Drawings are often called "vector graphics." They are usually produced manually via a draw-based graphics editor, or programming code. As with bitmapped graphics, drawings can usually be identified by file name extension: .DRW, .WMF, .XGR, etc.

In Everest, the A-pex3 programming language contains vector graphics commands (called Xgraphics commands). These commands can be used in Program objects.

Shapes and Lines

There is a third, intermediate type of graphic available in Everest: the Shape and Line objects. Shapes and Lines are similar to vector graphics, but do not require programming. You simply drag and drop them from the ToolSet, and adjust them visually.

Disk Space

Bitmapped graphics are notorious for occupying large amounts of disk space. An individual picture can easily use 50K of disk space. A few seconds of digitized video (such as in a Microsoft Video for Windows .AVI file) can occupy 2,000K (2 megabytes). The large size of bitmapped graphics often limits the extent of their use in a multimedia project.

Drawings occupy much less disk space. The average drawing built with Everest's Xgraphics vector drawing commands uses less than 2K of disk space. However, drawings generally are more difficult to create, and cannot produce photographic quality images.

Shapes and Lines occupy slightly more space than drawings because they are objects saved with a page's Book Editor.

7.3 Using Colors

All items in Everest (objects, bitmaps, vector graphics) employ colors. Proper color selection can give your project an attractive appearance. Therefore, it is important that you have a solid understanding of how Windows handles colors.

In DOS, there are many different graphics screen modes, each with its own color capability. Windows attempts to standardize color handling, but falls short of making the job simple. A complete discussion of how Windows produces colors is quite complex, and beyond the scope of this guide. The following paragraphs highlight the important items.

RGB

Each dot (pixel) on the monitor consists of a combination of red, green and blue. Windows allows each of these 3 colors to assume 256 different intensity settings (0 to 255). Something that has maximum red and blue (255 red and 255 blue) and no green (0 green) appears as bright purple. A color that is 64 red, 64 green and 64 blue appears gray (dim white).

So, theoretically Windows allows 16.7 million (256 x 256 x 256) possible different colors. Most current computers can only display 256 different colors at once, so Windows finds the best color match within a palette of 256 different colors it keeps in memory. If the computer can only display 16 different colors, then Windows chooses the best of the 16. The point is that Windows automatically does this color selection for Everest (and for your project). You have little control over how it picks the colors.

Dithering

If Windows cannot find a good color match, it may attempt to dither the color (mix two or more colors together). This tends to produce areas of color that look dotted or patterned. We recommend you try to avoid this as it rarely looks very good, particularly as a background for text. The trouble is, there is no good way to know if Windows is going to use dithering to produce a color.

One place dithering is very apparent is in Textbox objects. If you employ a dithered color as the background in a Textbox, Windows removes the dither pattern where text is written. This makes the Textbox appear to have more than one background color. If you observe this effect, you know that the background color is being dithered. To avoid this at design time, keep picking a different background color from the Color Dialog until the problem goes away.

At run time, you can assign colors via the Rgb() function in A-pex3 programming. For example:

```
Textbox(1).Backcolor = rgb(0, 0, 64)
```

produces a Textbox with a dark blue background. The Rgb() function has a special ability to generate the closest non-dithered (i.e. non-patterned) color. To do so, use a negative number for one or more of the Rgb() parameters; for example:

```
Textbox(1).Backcolor = rgb(0, 0, -64)
```

Recommendations

For choosing colors at design time, we recommend two general approaches:

1) employ Windows system colors (described in the next topic)

2) stick with the 16 default colors with which you are probably familiar from DOS; specifically:

Number	Color	Hex Value
0	black	&H0&
1	blue	&H800000&
2	green	&H8000&
3	cyan	&H808000&
4	red	&H80&
5	magenta	&H800080&
6	brown	&H8080&
7	white	&HC0C0C0&
8	gray	&H808080&
9	light blue	&HFF0000&
10	light green	&HFF00&
11	light cyan	&HFFFF00&
12	light red	&HFF&
13	light magenta	&HFF00FF&
14	yellow	&HFFFF&
15	bright white	&HFFFFFF&

These 16 colors are available (by default) in the "Custom Color" area of the Color Dialog box that appears when you double click on a color attribute in the Attributes window. Of course, if you have modified the colors in the Custom Color area, it may no longer have these 16 defaults. To restore the defaults, use a text editor or the INSTRUCT program to modify the EVEREST.INI file, and, in the CustomColor section of the file, enter the hexadecimal values shown in the table above.

7.4 Deciphering Hexadecimal Colors

At design time, after you choose a color from the Color Dialog box, Everest sets the corresponding attribute in the Attributes window. It represents the color via a number displayed in hexadecimal (base 16) format.

Recall that each color consists of three primary ones: red, green and blue. The hexadecimal value uses two characters to express the intensity of each of the primary colors. The order of the colors is backwards to what you might expect (go ask Microsoft), yielding the form BBGGRR. The &H is tacked on the front and the & on the end to designate hexadecimal format.

So, for example, a value of &H302010& means 30 (hex) blue, 20 (hex) green and 10 (hex) red. Translated into decimal values, that's 48 blue, 32 green and 16 red. A better way to think about it is that there is much more blue and green than red, so the actual color is likely to have a blue-green tint.

To employ a hexadecimal color in an A-pex3 program, use the Val() function, as in the following example:

```
Listbox(1).FillColor = val("&H8080")
```

System Colors

There are also several Windows system colors that are represented by 8 hex digit color values. Here is a table:

&H80000000&	scroll-bars gray area
&H80000001&	desktop
&H80000002&	active window caption
&H80000003&	inactive window caption
&H80000004&	menu background
&H80000005&	window background
&H80000006&	window frame
&H80000007&	text in menus
&H80000008&	text in windows
&H80000009&	text in title bar
&H8000000A&	active window border
&H8000000B&	inactive window border
&H8000000C&	application workspace
&H8000000D&	highlight background
&H8000000E&	highlight foreground
&H8000000F&	button background
&H80000010&	button shadow
&H80000011&	grayed (disabled) text foreground
&H80000012&	button text foreground

System colors are configured by the user via the Windows Control Panel. When you employ system colors, your project acquires the user's preferred color scheme. Everest's authoring programs work this way. To use a system color in your project, type the value from the table above into the Attributes window.

Be Careful Mixing System Colors

When using system colors, avoid mixing foreground and background from two different items, because you cannot be sure they produce contrasting colors. For example, it is not good to use &H80000012& (button foreground) on &H8000000D& (highlight background) because these could represent the same color (maybe not on your computer, but possibly in the user's preferred color scheme). A better combination is &H80000012& with &H8000000F& because these are both button colors, and should contrast.

Also, for the same reason, avoid mixing system colors and non-system colors on the same object. Since you cannot be sure what physical color the user has assigned a system color, you do not know how it will contrast with the non-system color.

7.5 Using Bitmaps In Everest

Currently, Everest does not contain a bitmapped graphics editor. To create bitmaps, we recommend you use your favorite editor (we are fond of ZSoft's PhotoFinish program), and save the pictures in one of the formats described below.

Bitmapped graphics are supported by several objects in Everest. To display background graphics within the window itself, employ the BgndPicture attribute. To display a bitmap within a defined area of a window, use a Picture or SPicture object. Everest supports many different graphics file formats,
including: .BMP, .DIB, .GIF, .ICO, .JPG, .PCX, .RLE, .TGA, .TIF and .WMF.

Background

The BgndPicture attribute of the Layout object lets you specify an image to display as the background of the window. All other objects, including Lines and Shapes, appear on top of a BgndPicture image.

Patterned and Tiled Backgrounds

Patterned backgrounds are very popular. This is easy to accomplish in Everest: simply load the desired image (most authors use one that is small in size) via BgndPicture, and enable the Layout object's Tile attribute.

Special Effects

To display the BgndPicture with extra style, use the Layout object's SpecialEffect attribute. If you'd like SpecialEffect to create a smooth visual transition from the prior page to this page's BgndPicture, place an Erase object in this page immediately before the Layout, and set its EraseType attribute to 18.

Picture Vs. SPicture

By default, the Picture object does not scale (resize) bitmaps to fit inside the object; the SPicture object does. If an image does not fit within a Picture object, and if no SpecialEffect is employed, the image is cropped.

Transparency

Via the TpColor attribute, the Picture object lets you specify a transparent color. If you also enable the CopyBgnd attribute, the background will show through in areas of the image that match the TpColor. The SPicture object has no similar features.

Authors often employ the transparency feature to overlay irregularly shaped images on a background.

Animation

The Picture object can act as a container for sprite animation, whereas the SPicture object cannot. See the Animation topic for details.

Special Effects

The Picture object has the ability to employ a special effect (slide, wipe, dissolve, etc.) when displaying a image. Choose one via the SpecialEffect attribute.

Transition from a Prior Image

To create an attractive visual transition from one Picture to another Picture, make sure the two Picture objects are of the same size (same Left, Top, Width and Height attributes) and make the IDNumber attributes identical. For best results, choose one of the overlay-style SpecialEffects (those numbered 18 and above).

Alternatively, you can employ A-pex3 programming to achieve the same visual transition without using a second Picture object. Such programming might resemble:

```
Picture(1).SpecialEffect = -18      $$ dissolve, deferred
Picture(1).PictureFile = "another.bmp"
```

SpecialEffect Limitations

Due to limitations in Windows, not all images can be displayed successfully with a special effect. We have learned that Windows often degrades the image quality and color if all the following are true simultaneously: 1) you use a SpecialEffect; 2) the Picture object is smaller than the actual image; 3) the image contains 256 or more colors; and 4) Windows is running in a 256-color mode.

Some possible workarounds: make the Picture object larger (enable the AutoSize attribute to have Everest automatically resize the Picture object to match the size of the image). Use an SPicture object (you won't be able to employ a SpecialEffect, but the image quality should be better).

Drawing on Pictures

Picture objects also accept Xgraphics drawing commands. This lets you draw vector graphics on top of a bitmap. This feature is described later in this chapter.

7.6 Using Bitmaps on Buttons and Other Objects

Everest can also display 16- and 256-color pictures on Button, Check, Frame, Gauge and Option (BCFGO) objects. In fact, if you want, it can display different pictures based on the state of the object (down, up, checked, unchecked, etc.).

The PicBin

For Picture and SPicture objects, you tell Everest the name of the file to load and display. BCFGO objects can operate somewhat differently: they can obtain their pictures via the PicBin object. The PicBin object acts as a holder for a bitmap that (typically) contains more than one clip-art style image.

This is best illustrated with an example. The samples that come with Everest contain a file named FLAGS.BMP. This single .BMP file contains dozens of tiny images of country flags from around the world. The individual flags are displayed adjacent to one another in the .BMP. If you showed the .BMP in a Picture object, you would see all the flags at once.

When you load a picture such as FLAGS.BMP into the PicBin object, Everest internally divides the image into a grid of cels, typically one image per cel. Think of this as a spreadsheet program with cels that can contain pictures...in this example, each cel would contain the picture of one flag.

Once you have the big image in the PicBin, it's very easy to display little pieces of it (the cels) in a BCFGO object. Simply set the object's Pic attribute to the number of the cel you want. The cel in the upper-left corner is number 1. Cel number 2 is the next one to the right, etc. To generate the proper cell number automatically, double click on the Pic attribute in the Attributes window; when Everest displays the PicBin image, click on the desired cel.

The Tricks

Where are the catches? The only tricky issue here is that when you load an image into the PicBin object, you must also specify the number of images it contains (otherwise, Everest doesn't know how much of the image to put in each cel). You do so by specifying the number of columns (across) and rows (top to bottom) of tiny images there are in the big .BMP file. If you enter the wrong values, the images that are displayed in the BCFGO object will appear shifted, or contain pieces of more than one. To determine the correct Columns and Rows, view the entire .BMP file (in an SPicture object, or your favorite editor) and simply count how many tiny images it contains left-to-right and top-to-bottom. Any extra space at the bottom and right sides of the picture must be counted as if it, too, were filled with tiny images.

7.7 Using Multiple Pictures on Button, Check and Option Objects

On Button, Check and Option objects you can display different images based on the state of the object. The process is the same as displaying a single picture (described in the previous topic).

To employ different pictures, enter the PicBin cell number for each Picxxx attribute. For example, the Check object has PicChecked, PicGrayed, PicPressed and PicUnchecked attributes. In your PicBin, appropriate images might be in cells 1 to 4, so you would set PicChecked to 1, PicGrayed to 2, etc.

IMPORTANT: when using multiple images, be sure to set the Pic attribute to 0 (or leave it empty).

To generate the proper cell number automatically, double click on the Picxxx attribute in the Attributes window; when Everest displays the PicBin image, click on the desired cell.

7.8 Sizing of Pictures on BCFGO Objects

You cannot control the location of the image on a BCFGO object, but you can control the size. Refer to the technical reference details for the Wallpaper attribute.

7.9 Palette Conflicts with Multiple Images

Some care must be taken when you display more than one image on the monitor at a time. The reason is that Windows determines its palette of displayable colors from those in the image *that has the focus*. When the focus moves from one image to another, the colors displayed may suddenly change. This is called palette shifting, and is usually unsightly.

Conditions that Produce Palette Shifting

Palette shifting will occur when all the following conditions are true simultaneously:

1) a total of more than one image is displayed in any window on the monitor
2) the color depth of at least two of these images is equal to or greater than the color depth at which Windows is currently running
3) the two images in question employ different color palettes

So, for example, if Windows is running in 256-color (8-bit) mode, and you display two different images, each with their own palette of 256 colors, Windows will shift its color settings to match the image that has the focus. The image that does not have the focus will likely be displayed with unusual colors.

Commonly, a place you can observe palette shifting is when branching from one page to the next. Images may flash in an unusual color as Windows is changing from one palette (that of the images used on the first page), to a different, second palette (that of the images used on the second page).

Solutions to Palette Shifting

Here are some ways to address the palette shifting issue:

1) Might Help Some: If the problem occurs only when branching between pages, try enabling the Layout object's LockUpdate attribute on the destination page
2) Easiest: run Windows at a higher color depth (perhaps 16-bit or 24-bit mode)
3) Compromise Visual Quality: use images with fewer colors (save them in 4-bit, 16-color mode)
4) Most Effort, But It Works: resave your images so they all employ the same palette; many software packages, such as Adobe PhotoShop, contain a "make identity palette" feature that takes several images and finds the best palette for all of them

5) Limit Access: don't allow users to run your project unless their copy of Windows employs greater than 256 colors; at run time, check the value returned by the Gdc(12) function; if Gdc(12) > 8 then the users will not encounter palette shifting for 256-color images

Option 4 above is what we recommend. Check our Web site for possible links to utilities that can help

7.10 Animation

Everest supports several forms of animation.

Animation Files

One of the most popular formats for animation is the so-called FLI-FLC format created by Autodesk Animator software. Everest can display .FLI, .FLC animation files via the Animate object.

Use of the Animate object is fairly straightforward. For the sake of performance, the Animate object is limited to one per page and per window. Also, the Animate object cannot be resized, or overlaid with a transparent background.

AnimPath

Everest's AnimPath feature provides more flexibility than the Animate object. Via AnimPath, you can make an object move around a window, and transparently overlay the image over the background.

Several objects, including Textboxes and Pictures, offer AnimPath-style animation; for those objects that do, you will find an attribute named AnimPath listed in the Attributes window. AnimPath has both basic and advanced capabilities. Some information is provided below; additional details can be found in the technical reference/on-line help.

Basic AnimPath Animation

At its simplest, the AnimPath attribute describes the path along which the object will move at run time. You can create the path by typing the numeric movement coordinates directly. However, it is often easier to have AUTHOR generate these values for you. To do so, double click on the AnimPath attribute, then follow the on-screen directions to move the object in the VisualPage editor along the desired path. When you run the page, Everest replays the path you recorded.

Transparent Animation

The basic animation describes above simply slides the rectangular Textbox, Picture or other object along the path you designate. If you would like an image to overlay the background during the animation, you should: 1) use a Picture object to hold the image to be animated; 2) enable the Picture object's CopyBgnd attribute; and 3) set the Picture object's TpColor attribute to the desired transparent color in the image.

Multi-Frame Animation

If you would like the animated image to change as the animation occurs, so that, for example, you could display a person walking, you can do so with help from the AnimCelStart and AnimCelEnd attributes. Refer to the technical reference/on-line help for instructions.

Sprite Animation

Sprite animation usually refers to animation that runs in the background, not the visual background, but the execution background, that is, while other events are occurring. Everest supports this via the AnimSpritePace attribute. In fact, you can even have more than one such sprite moving around the window while the user is doing other things. See the technical reference/on-line help for information.

Animation Via Programming

For simple, smooth movement of any object, use A-pex3 programming to adjust the values of the Left and/or Top attributes. The following example moves the Picture object with IDNumber 1 horizontally across the window:

```
maxx = Window(0).Width        $$ finish at
x = Picture(1).Left           $$ start at
Picture(1).Update = 1         $$ assure image plot
DO
  x = x + 2
  .Left = x
LOOP IF x < maxx
```

SpecialEffect

Though not really animation, interesting effects can be used with Picture objects via the SpecialEffect attribute. Refer to the technical reference/on-line help.

7.11 Multimedia Bitmaps

Yet another object that can load and display bitmapped pictures is the Media object. It can play back digitized video images stored in Microsoft Video for Windows .AVI format. When you add a Media object to the VisualPage, a row of buttons (play, fast forward, etc.) appears (you can hide these buttons via the ShowButtons attribute).

Playing Video for Windows .AVI Files

Here's how to play an .AVI motion video file (Microsoft Video for Windows file) in your project:

1) Put an object into the page that can act as a container for the .AVI; a Picture object works well.
2) Put a Media object after the Picture object.
3) Set DeviceType to AVIVideo.

4) Set DisplayIn to picture(1). The "1" is the IDNumber of the container object; your value might be different.

5) Set Command to Play.

6) Set FileName to the name of the .AVI file. You can double click on FileName to open the load file dialog box.

7) If you want to resize the video to fit the container, enable AutoScale.

8) Put a Wait object after the Media object.

9) Run a Preview of the page.

IMPORTANT: the object in which you display output from the Media object must already exist in the window...so be sure to place it above the Media object in the page as shown by the Book Editor.

7.12 Storage of Graphics Files

Graphics files used in your project can be either linked to or embedded in the .ESL book file. Appendix F in the technical reference/on-line help has more information about this topic, as well as many examples.

Linked Graphics Files

By default, Everest does not store a graphics file within the .ESL file. Instead, (via what is known as a linked approach) it stores the *name* of the graphics file, and later loads the file from disk only when needed. This means you must provide the graphics files with your project when you deliver it to the end user (Everest's Project Packager feature will gather the files for you).

The advantage to this approach becomes obvious when you need to edit a graphic. If Everest had incorporated it into the .ESL file, you wouldn't be able to retrieve it for editing. By keeping the file external, you can modify it freely, and let the changes appear in your project automatically.

Disk Paths for Linked Graphics Files

When you set the PictureFile, SPictureFile, and other attributes that specify external files, Everest allows you to prefix the file name with a disk path. However, we recommend that you do not do so. Here's why: if you prefix a disk path, Everest looks in that directory for the file...and will do so when the end-user runs your project. It is likely the end user will not employ the same disk drive and subdirectory names as you; Everest won't be able to find the file.

Better: while editing, copy the desired file to the same directory that contains the .ESL, and do NOT prefix a file name with a disk path. When you omit the path, Everest looks for external files in the same directory as the .ESL.

Also good: if you know that the name of the directory that contains the files now (while editing) is the same as that the end user will employ, but you are not sure the disk drive letter will also be the same, substitute a ? character for the drive letter. For example:

```
?:\pictures\ocean.bmp
```

The ? tells Everest to employ the drive on which the current .ESL file is located.

Perhaps the best: if the graphics files will be in a subdirectory off the directory that contains the .ESL, you can prefix the file name with just the subdirectory name. For example, if the book is in C:\lessons\ and the graphics are in C:\lessons\pics\, you can load a graphics file by entering

```
pics\mountain.pcx
```

This approach works regardless of the drive letter and book directory name.

Embedded Graphics Files

If you prefer, you can store graphics files within the .ESL book, and load them with objects such as Picture and SPicture. This is known as embedding.

To tell Everest you want an embedded file, manually prefix the file name with a | character (ASCII 124). When you do so in the Attributes window at design time, Everest checks if the file is already stored in the .ESL; if it is not, Everest prompts you to select a file to load.

Currently, the embedded graphics file feature is available for the AnimFile, BgndPicture, BMPFile, FileName, Pic, PictureFile and SPictureFile attributes.

Everest ignores a disk path specified after the | in an embedded file name (i.e. Everest always looks for the graphics file in the current book).

Embedded files have several advantages: 1) the files are stored in a compressed format (that works especially well on .BMP files); 2) the approach affords some copy protection to your graphics files; and 3) when copying and distributing the book, you need not also copy external graphics files.

Embedded files also have some disadvantages: 1) at run time they are slower to load because Everest must create a temporary file to transfer the image (this temporary file is written to the path you specify in Sysvar(152), or if Sysvar(152) is empty, then the path specified via the environment variable named TEMP, and if that is empty, then the path specified via the Sysvar(16) variable); and 2) when you modify an external copy of the graphics file, you must manually recopy it into the .ESL (this can be done via the Refresh feature of the Embedded File Manager window accessed via the Utilities menu in AUTHOR).

Your project can set Sysvar(152) to the desired disk path via A-pex3 programming, or you (and your project's end user) can do so by setting the TempPath entry in the EVEREST.INI file.

Deleting Embedded Files

To remove an embedded file from the book, in the Embedded File Manager window, highlight the name of the file to remove, and click Delete.

7.13 Drawing Vector Graphics

You can incorporate vector graphics in your project in two ways: via .WMF (Windows MetaFile files), or via A-pex3 Xgraphics commands. The Picture object and Layout object both accept .WMF files.

A-pex3 Xgraphics commands let you draw vector graphics via programming. Most authors enter the commands in a Program object, but they can also be used anywhere A-pex3 programming is allowed. Here are the Xgraphics commands, and what they do:

ARROW	draw an arrow

BOX	draw a rectangle
CIRCLE	draw an ellipse
COLOR	set color of graphics
FBOX	draw a filled rectangle
FONT	choose font of subsequent PRINT text
GFILL	fill with a graduated color
LINE	draw a line
LPRINT	send text to the printer
PAINT	fill an enclosed area with a color
POINT	draw a dot
POLY	draw a closed polygon
PRINT	display text
RBOX	draw a box with rounded corners
SCALE	customize window coordinate system
STYLE	set various graphics appearance attributes

More details about these commands follow in the topics below, or can be found in the technical reference/on-line help.

7.14 Layering Graphics

All graphics in Everest are displayed at one of four depths. The deepest graphics appear behind all others. The topmost graphics appear above all others. This layering is sometimes called the Z-Order.

Deepest: image loaded via BgndPicture attribute of the Layout object

Mid-Deep: A-pex3 Xgraphics commands in window

Mid-Top: Shape, Line and Frame objects

Topmost: all other objects with a visible component

Graphics displayed via objects, such as the Picture or SPicture objects, appear on top of everything else. When you have more than one such object, the one most recently executed in an Page appears on top by default. You can force the layering of objects within each of the four depths via the ZOrder attribute.

Transparency

Superimposed overlays within a Picture object can be obtained via the CopyBgnd and TpColor attributes; refer to the technical reference for help.

Hiding Objects at Design Time

At design time, if layering of objects in the VisualPage editor obscures the object(s) you want to edit, use the Hide feature in the Book Editor's pull-down menu to temporarily make the offending object invisible.

Drawing Xgraphics on Picture Objects

To draw Xgraphics on a Picture object, first set Sysvar(108) to the IDNumber of the desired Picture object. For example:

```
sysvar(108) = 1
CIRCLE (50, 50, 25)
sysvar(108) = 0
```

It is a good idea to set Sysvar(108) back to 0 when done. When Sysvar(108) is 0, Xgraphics are drawn in the window itself (on top of the BgndPicture, if any).

Each Picture has its own coordinate system (i.e. the upper-left corner of the Picture is 0, 0, unless you change it via a SCALE command).

7.15 Using Colors in Vector Graphics

Everest makes it easy to use the 15 standard (DOS-like) colors in your Xgraphics commands. Each command has an optional Color parameter, usually the last numeric parameter, that ranges from 1 to 15. When you include this parameter, Everest employs one of the 15 standard colors (which are stored in its internal palette). For the BOX command, this resembles:

```
BOX (0, 0, 100, 100, 15)
```

which employs color 15 (bright white, by default). Or,

```
BOX (0, 0, 100, 100, 1)
```

which employs color 1 (dark blue, by default).

Changing Everest's Palette Colors

If you want to alter a color in Everest's Palette, employ the COLOR command. For example, to use dim white in place of the default blue in palette slot 1:

```
COLOR (1, 64, 64, 64)
```

Now, subsequently drawn Xgraphics that employ color 1 will be displayed in dim white.

Omitting the Color Parameter

If you omit the Color parameter in an Xgraphics command, such as in:

```
BOX (0, 0, 100, 100)
```

then Everest employs the current foreground color (which can be selected via a COLOR command). Most of the time, you immediately precede the drawing command with a COLOR command. For example:

```
COLOR (-1, 64, 64, 64)
BOX (0, 0, 100, 100)
```

The -1 first parameter in the COLOR command tells Everest you are setting the default foreground drawing color. The other numbers select a dim white foreground color. Then the BOX command draws the rectangle using this color. If you have several Xgraphics commands that all employ the same color, preceding the group of commands with one COLOR command will produce faster drawing.

Color Number 0 (Color Black)

There is a 16th color: color number 0. Color 0 does not necessarily produce black...instead it produces the current background color. When the drawing color and the background color are identical, the graphics drawn are invisible!

So, for a black box, do NOT use:

```
BOX (0, 0, 100, 100, 0) $$ color 0, invisible!
```

instead, set the foreground (drawing color) and omit the color parameter in the Xgraphics command. For example:

```
COLOR (-1, 0, 0, 0)      $$ set black drawing color
BOX (0, 0, 100, 100)     $$ omit color parameter
```

Differences from Summit for DOS

In Summit for DOS, changing palette colors changes the colors *already displayed* on the screen. In Everest, it only changes the colors of *subsequently drawn* graphics. For example:

```
BOX (0, 100, 100, 100, 1)
```

employs palette color 1. That is blue by default, but if you have changed it via the previous COLOR command, instead the color is dim white. Everest remembers the settings of the 16 palette colors via Sysvar(30) through Sysvar(45).

Nearest Solid Color

A COLOR command such as:

```
COLOR (1, 64, 64, 64)
```

might produce a non-solid color (Windows dithers, or mixes, colors when there is no exact solid color match in its palette). To tell Everest to pick the nearest solid color, use a negative value for one or more of the color parameters. For example:

```
COLOR (1, -64, 64, 64)
```

assures a solid color will be produced.

7.16 Drawing Simple Vector Graphics

Once you understand how Windows handles colors, the only other tricky thing to know is that the drawing commands are influenced by the settings of previous STYLE commands. The STYLE command controls the pen width, mode, style and fill pattern. Refer to the technical reference for the details.

Drawing Dots

The simplest Xgraphics drawing command is POINT. It draws a dot (1 pixel) within the window. Here's an example:

```
POINT (200, 100)
```

which draws a dot 200 pixels from the left edge of the window and 100 pixels down from the top.

IMPORTANT: The parameter list of all Xgraphics commands is enclosed with parentheses, and is separated from the command by a space. Individual parameters can be constants, variables or expressions, and are separated from each other with a comma.

The POINT command also offers an optional third parameter that selects a color from Everest's 16-color palette. The example:

```
POINT (200, 100, 1)
```

uses color 1 (blue, by default).

Drawing Rectangles

There are three rectangle drawing commands: BOX, FBOX and RBOX. BOX draws the edge of a rectangle, FBOX always draws a solid (filled-in) rectangle, and RBOX draws a rectangle with rounded corners. The location of the rectangle is specified via four parameters.

The first four parameters of the three commands specify two opposite corners of the location of the rectangle within the window. The upper-left corner of a window is, by default, at X, Y location 0, 0. The lower-right corner of the window is set by the Layout object. A common value is 640, 480. For example:

```
BOX (10, 10, 400, 100)
```

draws a rectangle that starts close to the upper-left corner of the window, and extends to a location 400 pixels to the right of the left edge of the window, and 100 pixels down from the top.

An optional additional parameter controls the drawing color, as described in an earlier topic.

FBOX does the same as BOX, except it fills the inside of the rectangle with color. Note that it is possible to draw a filled rectangle with BOX: set the FillStyle via STYLE, as in this example:

```
STYLE (4, 0)                  $$ set solid fill
BOX (10, 10, 400, 100)        $$ now same as FBOX
STYLE (4, 1)                  $$ reset to transparent
```

The STYLE settings stick for subsequent Xgraphics commands, so it is often wise to restore the original values as done in the example above.

Drawing Lines

The LINE command draws a line between two points. Specify the two points, and the color, just as you do for rectangles. For example:

```
LINE (10, 10, 400, 100)
```

draws a line from X-Y location 10, 10 to location 400, 100. To draw from the previous location, include only one coordinate pair, as in:

```
LINE (400, 100)
```

Drawing Ellipses

To draw an ellipse, use the CIRCLE command. At minimum, you must include parameters for the X-Y center and radius. For example:

```
CIRCLE (150, 200, 50)
```

draws a circle that is 100 pixels wide (50 from edge to center, and 50 from center to opposite edge). To draw a filled circle, use:

```
STYLE (4, 0)                 $$ choose solid fill
COLOR (-5, 1)                $$ palette color number 1
CIRCLE (150, 200, 50, 2)     $$ edge is color 2
```

To draw an arc, include two additional parameters: starting angle and ending angle. Express in degrees, from 0 to 360. East (on a compass) is 0 degrees, North 90, West 180, and South 270. For example, to draw a half circle, use:

```
CIRCLE (150, 200, 50, , 0, 180)
```

Also notice that in this example, the color parameter is omitted (telling Everest to employ the default foreground color previously set via a COLOR command), but the corresponding comma is still included.

7.17 Drawing Polygons

To draw a closed, irregular, N-sided shape (a polygon), use the POLY command. The POLY command accepts from 3 to 100 endpoint coordinates, specified in a long string. For example:

```
POLY (0, "100,50;150,100;50,100")
```

which draws a triangle. Notice that the X-Y coordinate pairs are separated by a semicolon (;) while the X and Y are separated by a comma (,). The whole group of coordinates is surrounded by quotes (to make it a string).

The first parameter in the POLY command specifies the type of paint (fill) to employ:

0	do not paint inside
1	paint via alternate side process
2	paint via winding process

The two different paint types can produce different effects depending on the shape of the polygon. It's best to try both to learn the difference (for some shapes there is no difference).

A catch: remember that the STYLE command controls the style of paint employed (solid, pattern, etc.). By default, the paint is transparent (i.e. no paint at all). Typically, before a POLY command, most authors employ the STYLE command to choose a paint style. For example:

```
STYLE (4, 0)                  $$ choose solid paint
POLY (1, "100,50;150,100;50,100")
```

To pick the color of the polygon, and the color of the paint, employ the COLOR command before the POLY command.

7.18 Painting Areas

To fill an enclosed area, use the PAINT command. PAINT starts working at the spot you specify, and continues until it hits an edge of the color you also specify. An example:

```
PAINT (200, 100, 1, 9, 0)
```

fills starting at X-Y location 200, 100, and uses color 1 for the paint, stops when it hits an edge of color 9, and uses paint style 0 (solid). The last three parameters of PAINT are optional; see the technical reference for details.

Authors sometimes find the whole window filled with paint. This means the edge color was incorrect (so PAINT just kept going), or there was a tiny break in the edge, allowing the paint to leak out. If you watch carefully at run time, you might be able to see where PAINT escapes.

GFILL Command

The GFILL command does not paint within an area, instead it fills an object with a graduated color. GFILL is often used to create an attractive background. For example, to fill the background of the window with a graduated blue color, use:

```
GFILL (0, 0, 0, 0, 0, 255)
```

Due to possible color conflicts, GFILL is not recommended for use in windows that display bitmapped graphics.

7.19 Displaying Text

Textbox and Flextext

The easiest way to display text is via the Textbox or Flextext objects. You simply type the text into them, choose the desired font and color, and you are done.

Special Characters

To place special characters (such as a degree symbol) into a Textbox or Flextext object, first position the cursor at the desired location in the object, then from the Author window's Utilities menu, choose Character Table. Click on the character/symbol you want.

Textbox with DrawText

Both the Textbox and Flextext objects normally plot text in a destructive fashion, that is, they obscure whatever is underneath. However, via Everest's DrawText feature, you can display text without the obscuring rectangle. To do so, place a Textbox in your page, set its DrawText attribute to -1, and enter your text. At run time, Everest automatically converts the text within the Textbox into the PRINT commands described in the subtopic below. Everest draws the text onto either the window background itself, or onto a Picture object, if that Picture is directly beneath the Textbox.

For persistent display of DrawText text, set DrawText to -2, or enable the Layout or Picture object's AutoRedraw attribute.

DrawText with Drop Shadow

To plot the text with a drop shadow, set the DrawShadow attribute to the desired color. This works only when DrawText is in effect.

DrawText with Special Effect

To plot the text with a special effect (such as a dissolve), set DrawText to a value greater than 0. The drop down list in the Attributes window shows the possibilities.

DrawText with Pause

Sometimes you may want to display several lines of text, but pause briefly after each line. Often, this is an attractive way to present a series of bulleted items. This is accomplished easily with help from the DrawPause attribute. Specify the desired number of seconds in the DrawPause attribute. At run time, Everest looks for a CR (carriage return) character in the text to determine the end of the line. You can put such a character into the Textbox manually by pressing the Enter key where you want the text to break.

Combine DrawPause with a special effect setting in DrawText, and it's very easy to present a series of lines of text, each added to the display with a special visual effect.

Erasing DrawText

DrawText works by printing the text onto the object beneath the Textbox. If there is no object, the text is printed directly onto the window background. The text will remain there until you either change the BgndPicture of the Layout object, or erase it via the Erase object or ERASE command.

There is a way you can to erase the text without erasing or replotting the rest of the window background: put a Picture object underneath the Textbox. Make sure the Picture is listed before (i.e. above) the Textbox in the Book Editor. Resize the Picture to that of the Textbox (or slightly larger), and enable the Picture's CopyBgnd attribute. This will make Everest print the text on the Picture, rather than on the window's background. Later, to remove the text, use the Erase object or ERASE command to delete the Picture object; doing so will remove the text and expose the original window background.

Text via PRINT

For more flexibility displaying text, you can use the PRINT command in your A-pex3 programs. The PRINT command displays text. PRINT is handy when you don't want to be "boxed in" by a Textbox object. Via a preceding FONT command, you can choose the font, size and styles you prefer. PRINTed text appears in the same layer with other vector graphics (i.e. behind objects).

The following example draws large text in color 1:

```
FONT ("Roman", 28, -1, -1, 0, 0, 0)
PRINT (10, 10, 1, "Rome is a city in Italy.")
```

PRINTing onto a Picture Object

To print (non-destructive) text on a Picture object, first set Sysvar(108) to the IDNumber of the desired Picture object. For example:

```
sysvar(108) = 1
PRINT (10, 10, 1, "Rome is a city in Italy.")
sysvar(108) = 0
```

It is a good idea to set Sysvar(108) back to 0 when done. When Sysvar(108) is 0, Xgraphics are drawn in the window itself (on top of the BgndPicture, if any).

Each Picture has its own coordinate system (i.e. the upper-left corner of the Picture is 0, 0, unless you change it via a SCALE command).

Multi-Color Text

The text that appears in a given Textbox must be all one foreground color, one background color, and one size. For more flexibility, use the Flextext object. It is not as simple as a Textbox, but it has more capabilities.

You employ the special features of the Flextext object by embedding special codes within the text itself. You can find a description of these codes in the Text attribute topic of the technical reference/on-line help.

Changing Color of Flextext

To change the foreground color of a portion of text within a Flextext object, you can manually type the special codes mentioned above, or you can have Everest do it for you. To have Everest embed the proper color code, simply highlight the desired text within the Flextext, then choose "Flextext color" from the main Author window's Edit pull-down menu. The color dialog box will appear, from which you can make a choice.

7.20 Printing Text on a Printer

To print text on the (line) printer, instead of in the window, use the LPRINT command. LPRINT accepts up to two parameters. Unless you tell Everest otherwise, LPRINT simply sends the text to the Windows Print Manager, and advances to the next line on the page.

Most authors tell Everest to automatically insert a form feed when a page is full. To do so, before LPRINTing the desired text, first use LPRINT Operation 0 to specify the number of lines of text per page, as in this example:

```
LPRINT (0, 54)
```

which tells Everest to form feed after printing 54 lines of text via subsequent LPRINT commands. Everest remembers this value via Sysvar(21). In Sysvar(22) it counts how many lines have been printed so far. You can modify the values in Sysvar(21) and Sysvar(22) directly via A-pex3 programming, if you prefer.

When you want to print text, use LPRINT Operation 1:

```
LPRINT (1, "Print this on the printer.")
```

You can use object attributes to specify the text, for example:

```
LPRINT (1, Input(1).Text)
```

prints the text of the Input object with IDNumber 1. The Windows Print Manager scans such text for carriage return characters (ASCII 13) to determine where to wrap the text, and how many lines it contains.

Printing the Last Page

The Windows Print Manager spools your print job until you send it an "end of document" code, and only then does it actually begin printing. Use LPRINT Action 2 or -2 to send an "end of document" code. So, when you are done printing, use either

```
LPRINT (2, "this is my last line of text")
```

or

```
LPRINT (-2)
```

to tell the Windows Print Manager it is OK to release the page for printing.

Wrapping Text

If the text you are printing does not contain embedded carriage returns, the Windows Print Manager will simply let the text run off the right edge of the page. Fortunately, Everest has the Wrp() text wrapping function. You can tell the Wrp() function to insert a carriage return/line feed nearest a word break after 1) a certain number of characters, or 2) a certain printing length. When used with LPRINT, this can resemble:

```
LPRINT (1, wrp(70, Textbox(1).Text))
```

See the technical reference for details.

Controlling the Printer Font

The text prints in the current font of the printer. If the printer allows it, you can control the font (and related attributes) via the FONT command. The trick is to temporarily set Sysvar(8) to -1 which tells Everest to refer to the Printer. Here's an A-pex3 programming example:

```
sysvar(1) = -1          $$ clear any previous error
wassys8 = sysvar(8)     $$ save old sysvar(8) value
sysvar(8) = -1          $$ choose printer
FONT ("MS Sans Serif", 12, -1, -1, 0, 0, 0)
sysvar(8) = wassys8     $$ restore old sysvar(8) value
IF sysvar(1) = -1 THEN
  LPRINT (2, "This is MS Sans Serif size 12 font.")
ELSE
  dummyvar = mbx("Error " + sysvar(1) + " occurred!")
ENDIF
```

Everest passes your font and printing instructions to the Windows Print Manager, which then worries about performing the actual operation. If the font you choose is not installed for the printer, an error code will be set in the Sysvar(1) variable.

7.21 Printing Window Image

Everest also lets you send a copy of the image within a window to the printer. This is similar in concept to the DOS <PrintScreen> key, except that Everest prints only the contents of one window at a time. As with LPRINT, Everest simply sends the information to the Windows Print Manager, which then determines how to perform the printing on the currently selected Windows printer.

To print the current window, reference function ext(19) in A-pex3 programming, as in this example:

```
ecode = ext(19)
IF ecode # -1 THEN
  dummyvar = mbx("Error " + ecode + " upon ext(19)!")
ENDIF
```

To print a window other than the current one, temporarily set Sysvar(8) as in this example:

```
wassys8 = sysvar(8)     $$ remember current value
sysvar(8) = 2           $$ window number 2
ecode = ext(19)         $$ print window
sysvar(8) = wassys8     $$ restore previous value
```

All visible objects and bitmaps in the window are printed. Xgraphics are also printed if the AutoRedraw attribute of the Layout object is set to Yes when the Xgraphics are drawn. Refer to AutoRedraw in the technical reference for important details.

7.22 Erasing Vector Graphics

Since vector graphics are not an object (like the Shape object), they have no Visible attribute that you can set to No. To remove all vector graphics from the window, employ the ERASE command (or object). Examples:

```
ERASE (1)         $$ erase Xgraphics, leave objects

ERASE (3)         $$ erase Xgraphics and other objects
```

The only tricky item here is that the setting of AutoRedraw when the erase is performed must be the same as it was when the vector graphics were drawn. If the graphics do not erase as you intend, retry after changing the value of the Layout object's AutoRedraw attribute.

To remove selected Xgraphics from the window, consider redrawing them with a color that matches the background. You can also use an FBOX command to wipe a rectangular area.

Picture Object

To remove vector graphics from a Picture object, load another PictureFile.

7.23 Scaling Graphics

The default coordinate system of a window has X-Y location 0, 0 in the upper-left corner, and something like 640, 480 for the lower-right corner (depending on the size of the window). For special situations, you might want to temporarily change the coordinate system, draw some graphics, and change it back.

For example, say you have created a Program object named drawing that is normally displayed in the upper-left quadrant of a 640 x 480 size window, but you want to display it in the lower-right quadrant. The following example does this:

```
SCALE (0, -320, -240, 320, 240)
GOSUB drawing
SCALE (3)
```

The first SCALE command shifts the 0, 0 coordinate to the center of the window. The second SCALE restores the original coordinate system.

7.24 Displaying Vector Graphics in Another Window

Everest refers to the value in Sysvar(8) (the current window number) to determine the window in which to display Xgraphics. If you want to display graphics in a window other than the current, simply change the value in Sysvar(8) temporarily. For example:

```
wassys8 = sysvar(8)
sysvar(8) = 2               $$ set to window # 2
CIRCLE (100, 100, 50)
sysvar(8) = wassys8
```

Be sure to restore the original value of Sysvar(8) when done drawing; failure to do so is likely to produce unpredictable results.

7.25 Persistent Vector Graphics

Since vector graphics are not objects, Windows does not automatically refresh them when necessary. For example, when you have two overlapping windows, and close one of them, the objects (such as Textboxes) in the remaining window are automatically refreshed for display by Windows, but the text from PRINT commands is not.

If this creates a problem for your project, you have two workarounds: 1) AutoRedraw, and 2) Refresh.

Using AutoRedraw

AutoRedraw is an attribute of the window (the Layout object). When you set AutoRedraw to Yes, Everest saves a special bitmapped copy of vector graphics as they are drawn in the window. If the window later needs to be refreshed, Everest restores the image from this special copy.

AutoRedraw has drawbacks. The special bitmapped copy of the screen occupies an indeterminate amount of memory (proportional to the size and color depth of the window). When AutoRedraw is enabled, vector graphics are not displayed until there is a "break in the action" (i.e. a Wait object or ext(101) reference). Finally, to erase graphics drawn when AutoRedraw is enabled, you must use the ERASE command (or object) while AutoRedraw is still enabled.

Here's an example that uses AutoRedraw:

```
window(1).AutoRedraw = "Yes"
CIRCLE (100, 100, 50)
window(1).AutoRedraw = "No"
```

Later, to erase that CIRCLE:

```
window(1).AutoRedraw = "Yes"
ERASE (1)
window(1).AutoRedraw = "No"
```

Using Refresh

Another way to automatically refresh graphics is to enable the Refresh attribute of the Program object containing the Xgraphics. When Refresh is set to Yes, Everest automatically re-executes the Program object whenever Windows tells it that it needs refreshing. Only those Program objects in the currently loaded Page for the window are re-executed (i.e. if you BRANCH to a different page, the Program objects from the previous will not be re-executed for refreshing purposes).

Refresh does not consume memory the way AutoRedraw does. The drawback is that you must be sure to avoid placing commands that should NOT be repeated (such as BRANCH commands) in that Program object. Another drawback is that sometimes Windows gets carried away and tells Everest to refresh the window more often than is really necessary. The result is your graphics might appear to flicker, or might require extra time to replot.

7.26 Using Shapes and Lines

To draw simple shapes and lines without programming, employ the Shape and Line objects. Just drag them in from the ToolSet.

Shape objects can appear as rectangles, squares, ellipse, circles, rounded rectangles, and rounded squares. You control the appearance via the Shape attribute.

Shapes also have a special ability to recognize mouse clicks, even when hidden behind other objects. This lets you easily create invisible "hot spots" in an image. Details are given in the Interaction chapter here, or with the ClickEvent attribute in the technical reference/on-line help.

Due to a Windows limitation, Shape and Line objects do not cooperate well with vector graphics drawn via Xgraphics commands. The objects plot in a destructive fashion (thereby hiding nearby vector graphics).

7.27 Choosing Fonts

One of the most difficult issues you will encounter involves font selection. If you use fonts that are not supplied with Windows, you cannot be sure that the user's computer will have them. You might consider shipping the necessary fonts with your project, but then you must obtain the rights to do so, or create the fonts yourself. Our recommendation is to employ only those fonts that are shipped with Windows.

The following fonts are the ones included with Windows 3.1:

Arial

Courier

Modern

MS Sans Serif

MS Serif

Roman

Script

Symbol

Times New Roman

Wingdings

If the user's computer does not have the font you employ, Windows will pick another one it believes to be most similar. Usually the results are not very attractive. At run time, you can check if a particular font is available on the user's computer via the Fnt() function, which is described in the technical reference/on-line help.

The Font Size Issue

Another issue is that the user might be employing so-called "large fonts." At higher resolutions, the Windows Setup lets the user choose larger fonts. When large fonts are employed, Windows automatically makes text larger without also making its container (Textbox, Button, etc.) larger. The result is your carefully placed Captions on Buttons (for example) might become too long, etc.

Our recommendation is to employ only those fonts that are shipped with Windows. Also, test run your project in several Windows resolutions (you can choose the resolution via the Windows Setup program). Generally, allow a bit of extra horizontal room for Captions on Buttons and similar objects in case Windows makes them larger.

Dynamic Font Scaling

If you do not want to leave extra room in all your objects in case the user is running with "large fonts" then you should instead implement the following little trick: tell Everest to adjust the FontSize attribute at run time. To do so, set Sysvar(115) to the desired scaling factor. For example, if you have developed your project while working with normal size fonts, and the user runs with large fonts, the user will see fonts that are larger (relative to their container). If desired, you can compensate by scaling the fonts smaller via a calculation at the start of the project such as:

```
IF Sysvar(3) = 12 THEN  $$ if 12 twips/pixel (large)
  Sysvar(115) = .8      $$ make fonts only 80% as big
ENDIF
```

Of course, if you authored while Windows is set for large fonts, and the user runs the project with normal fonts, the text will appear smaller relative to its container. In that case, you can compensate by making the fonts larger:

```
IF Sysvar(3) = 15 THEN  $$ if 15 twips/pixel (normal)
  Sysvar(115) = 1.25    $$ make fonts 125% as big
ENDIF
```

If you discover other good techniques for coping with these issues, please let us know so we can share them with others via our newsletter.

7.28 Checking the Graphics Hardware

In your project, if you employ anything other than a 640 x 480 resolution in 16 colors, you should check that the user's computer has the necessary graphics capability. A good place for these checks is in a Program object on the @start and @restart pages. If the user's computer does not have sufficient capability, you should display a simple message box with this information, then exit gracefully.

The following A-pex3 programming example checks that the user is running in at least 1024 x 768 resolution with 256 colors:

```
IF gdc(8) < 1024 @ gdc(10) < 768 @ gdc(12) < 8 THEN
```

```
  txt = "We're sorry.  You need to be running Windows"
  txt = txt + "at 1024 x 768 x 256 resolution, minimum,"
  txt = txt + "in order to view this presentation."
  dummyvar = mbx(txt, 16)
  BRANCH @exit
ENDIF
```

A brief description of the functions used in the example above:

Gdc(8)	width of the display in pixels
Gdc(10)	height of the display in pixels
Gdc(12)	color resolution of the display, in bits per pixel (1 = 2 colors, 4 = 16 colors, 8 = 256 colors, 15 = 32K colors, 24 = 16.7M colors)

7.29 Smoothing Page-to-Page Transitions

As your project's pages are executed at run time, Everest adds and removes objects from the window. Depending on the speed of the computer, the user generally can see the changes occur step-by-step (i.e. a Textbox appears, followed by a Picture, then a Button, etc.). These changes are most obvious when branching from one page to the next (simply because that is typically when most visual changes take place). There are some techniques you can use to smooth these transitions.

Using LockUpdate

The changes to the contents of the window can cause flicker. To virtually eliminate this flicker, tell Everest to hold all visual changes in memory, then apply them to the window all at once. To do so, simply set the LockUpdate attribute of the Layout object to Yes, and put a Wait object later in the page.

When LockUpdate is Yes at run time, Everest changes the mouse cursor to an hourglass, continues to process the page (adding and removing objects "behind the scenes"), and when it encounters a Wait object, it updates the window to reflect all the changes at once, and finally, restores the previous mouse cursor.

Due to a Windows quirk, when LockUpdate is enabled, Xgraphics might not plot. If you encounter this problem, it usually can be solved by setting the Layout object's AutoRedraw attribute to Yes.

Be sure to see the technical reference's LockUpdate entry for more details. While there, also see DoEvents.

Forcing Update

As objects are added to a window, Windows tries to guess when it is best to actually display the contents of the object. It does so in an attempt to maximize performance. Sometimes this means Windows does not display the contents of objects until a Wait object is encountered in your page. This could cause you headaches, as in the following example:

Object	Name	Comment
(Icon)	**movepic**	
(Icon)	movepic_layout_A	window info
(Icon)	movepic_textbox_A	title
(Icon)	movepic_picture_A	image
(Icon)	movepic_program_A	move image
(Icon)	movepic_wait_A	wait for user

If the Program object in the example above performs a time consuming operation, such as animating the Picture object, Windows might display the animation before any prior objects in the Page, such as the Textbox.

Fortunately, Everest gives you two ways to handle such a situation:

1) Tell Windows to visibly update an object. To do so, set the object's Initially attribute to either 9, 11 or 15. In the example above, you would set Initially to 11 for both the Textbox and Picture objects. Notes: avoid enabling LockUpdate when using this method; this method is recommended over the next.

2) Allow a "break-in-the-action" to give Windows a change to process pending events. This is accomplished by inserting the following at the top of the Program:

   ```
   dummyvar = ext(101)
   ```

   When Windows receives the Ext(101) function, it will refresh the display. The Ext(101) function solves certain timing related problems, but can cause others. See the technical reference for some warnings.

7.30 Automatic Proportional Resizing

Many authors wish the pages of their project to occupy the entire display (thereby preventing the distraction of other Windows applications). The tricky part is maximizing the window and relocating objects to handle all the different Windows display resolutions. Fortunately, Everest makes this very easy.

Everest can automatically proportionally scale all objects (except Animate) to fit within the window, regardless of its size. To create a window that occupies the entire display and contains objects that are proportionally resized to match, simply set the Layout object's AutoResize attribute to Yes, and WindowState to 2 (Maximized).

If you also want to prevent the user from resizing the window manually, set WindowBorder to either 0 or 1. When WindowBorder is 0, no border elements (such as the caption bar) are displayed. If, while running such a page in the AUTHOR program, you find yourself without a way to exit, press the main menu hot key, Ctrl+M.

7.31 Creating Scrolling Windows

Another way to handle all display types and sizes is to allow the user to scroll the contents of the window. This technique works particularly nicely if you create windows that are large, say 1024 x 768, but need users to be able to display them even if their copy of Windows is running at a lower resolution.

To allow the user to scroll the contents of a window, put a Layout object in your page and enable the Scrollable attribute.   Also, set the VirtualWidth and VirtualHeight attributes of the window; these values represent the maximum scrollable area.   At run time, if the user's window is smaller than VirtualHeight and/or VirtualWidth, scroll bars will automatically appear and will be user-adjustable.

This feature works via Crescent software's QuickPak Professional product and is subject to its limitations.

8 Interaction

8.1 What Is Interaction?

Most computer software presents a tremendous amount of information to the user. In interaction, the user provides information to the computer software. That information might be a request to move to the next page, an answer to a question, etc.

In Computer-Based Training (CBT) projects, interaction is a key element. It engages the user and helps to keep things interesting. Interaction is also a means of determining the user's skill and retention levels.

Everest supports a wide variety of interaction and contains features for CBT applications, such as user response judging. The interaction features in Everest support the standards set by Windows.

8.2 Simple Interaction: A Button

One of the most common interactive items in any Windows application is a button. When the user clicks on a button, he expects an action to take place. Let's look at how to make this happen in Everest.

As described in the Events and Branching chapter, Everest lets you design your project to be event driven. This means your project "sleeps" until it detects an event trigger (that you specify) and then performs an action (that you also specify). To tell a page to sleep and wake up upon user interaction, you place a Wait object in it. Before the Wait object, you put all the possible objects with which the user might interact.

Button to Play .WAV Sound File Example

As an example, consider a project with a page named "page1" that contains a button. When the user clicks the button, you want to perform some multimedia, such as playing a .WAV sound file. Here's how:

1) create a new page; name it `page1`

2) drag a Button object into it

3) edit the Button's Caption property so it reads `Play Sound`

4) invent and enter a ClickEvent code for the Button, for example, `13` (the code for the Enter key)

5) drag a Wait object to page1 after the Button

6) set the NextActivator of the Wait object to match the ClickEvent of the Button, in this example, that's `13`

7) leave NextAction empty...that allows execution to fall through to the object(s) you place after the Wait

8) after the Wait object, place the multimedia you want (to play a .WAV file, you would drag in a Media object, set its DeviceType attribute to `WaveAudio`, set Command to `Play`, and set FileName to the name of the .WAV file you want)

9) to try it, run a Preview of the page, and click on the Button

What's Important Here

The important thing in the example above is that you specify the same event code in both the Button's ClickEvent attribute and the Wait's NextActivator attribute. That's how Everest knows what to do when the user clicks on the Button. The ClickEvent attribute of the Button specifies the event that is generated when the user clicks on the object. The Wait object's NextActivator attribute tells Everest to wake up when it sees this event. The NextAction attribute says what to do. The NextAction can be a branch to another page, a calculation, or even a single line of A-pex3 programming. If you leave NextAction empty, execution continues with the object(s) after the Wait.

Why ClickEvent 13?

Why did we choose 13 as the event code in the example? Because the same event code is generated when the user presses the Enter key. So, the example also plays the .WAV file if the user presses Enter. By using event codes in ClickEvent that match those generated by keypresses, it's easy to make your project both keyboard and mouse aware. Appendix A lists the event codes generated by various keypresses.

But I Want Something More Descriptive

If you feel numeric event codes too obscure, take heart! Everest also lets you specify any character string as the event. So, instead of 13 in the example above, you could use "clicked music button". Simply enter any word/phrase you want, *surrounded by double quotes*. Be sure you set both the ClickEvent and NextActivator to the same word/phrase.

I Want to go the Next Page When They Click the Button

In that case, put something in NextAction. For example, set NextAction to:

```
BRANCH @next
```

and your project will branch to whatever the next page is in the book.

8.3 Creating Pull-down Menus

Another form of interaction is a pull-down menu. Pull-down menus appear as a bar near the top of a window. The user makes a choice by clicking on it with the mouse or using an access key. As with the ClickEvent for a Button object, each item in a Menu generates an event code (that you specify) when selected. The events can be detected and actions performed with a Wait object.

To add a pull-down menu to a window, drag a Menu icon from the ToolSet to the page. To edit the contents of the Menu, double click on the Menu icon in the Book Editor. This opens the Menu Editor window, which is a free-form text editor, where you might enter something like:

```
&File, 1
  &New, 2078
  &Open, 2079
  &Save, 2083
  -
  &Quit, 2081
&Edit, 2
  Cu&t, 2088
  &Copy, 2067
  &Paste, 2086
```

There are several things to note in the example. Non-indented items will appear at the top of the menu (i.e. in the menu bar). Items indented with at least one space will appear in the pull-down area of the menu. To create an access key, prefix the desired letter with an ampersand (&). Use a hyphen (-) to place a separator bar in the menu. The number that follows the item is the event code to generate when the user selects the item; even though non-indented items cannot be selected by the user, they must have a dummy event code (such as 1 and 2 in the example) in order to enable the menu.

As with other objects in a window, the pull-down menu remains in the window until removed by an Erase object or command. Most authors want the menu to be present for the entire duration of the user's session. To do so, set the Menu object's IDNumber attribute to an "unusual" number, such as 99, and avoid including 99 within the EraseFromID/EraseToID range of the Erase object.

8.4 Modifying Pull-down Menus

You might want to modify an existing menu to, for example, add a check mark next to an item, or gray it out (disable it), etc. To change an item in a menu, you need to know its location in the menu, specifically its column and row. The leftmost column is column number 1. The topmost item (the one in the menu bar itself) is in row 0; the first item in the pull-down area is in row 1.

For example, to disable the first item in the pull-down area of the second to left column, you would enter the following in a menu object:

```
,,0,,2,1
```

An explanation: the leading commas act as place holders for the item name and event code. The 0 disables the item, the extra comma is a place holder for the check mark indicator, the 2 is the column and the 1 is the row.

Or, to put a check mark next to this item:

```
,,,1,2,1
```

Notice there are 3 leading place holder commas. These commas skip past the item name, event code and enabled status parameters. When you omit a parameter, Everest does not change that aspect of the menu item.

You can add new items to an existing menu by setting the column and row values to a high enough number. Everest allows a maximum of 8 columns with 20 rows of items in each.

Also, when updating an existing menu, you must be sure to set the NewMenu attribute of the Menu object to No. When NewMenu is Yes, Everest removes all menu items before processing your menu specifications. That's great if you are creating a totally new menu, but not good if you are simply modifying an existing one.

8.5 Obtaining Input Via Input and Mask Objects

Another very common type of user interaction involves obtaining a fill-in-the-blank response. The Input and Mask objects do exactly that. The Input object is very similar to a Textbox object; the most significant difference is the user can edit its contents. Mask is similar to Input, except it allows you more control over the user's entry.

To quickly try an Input object, drag its icon from the ToolSet and drop in on the VisualPage editor. Add a Wait object to the page after the Input. Then run the page via Preview.

The Input object works well, and it's easy too. The slightly tricky part comes when you want to process the user's response in order to do something. You need a way to retrieve what the user typed into the Input or Mask object; you have two choices: 1) reference the object's Text property via programming (discussed in this topic), or 2) perform answer judging on the user's response (discussed in the next topic).

ResponseVar Tip

Some authors are confused by the operation of the ResponseVar attribute of the Input object. They wonder why the user's response is not being stored in the variable. What they fail to realize is that Everest copies the user's response into the ResponseVar *only when answer judging is performed* via a Judge object. Answer judging is discussed in the next topic in this chapter.

Without Judging

We recommend that you use the answer judging approach (even when all responses are correct) because it simplifies handling the response. But, for now, let's look at how you can retrieve the response without answer judging. A typical page to do this resembles:

Object	Name	Comment
Page	**q1**	
Layout	q1_layout_A	window stuff
Erase	q1_erase_A	erase previous objects
Textbox	q1_textbox_A	the question text
Input	q1_input_A	user fill-in
Wait	q1_wait_A	wait for response
Program	q1_program_A	branch upon response

On many pages, you put branching instructions in the Wait object. In this example, you do not because you need room for an IF...THEN block in which to examine the user's response.

In the example, you would set the Wait object's NextActivator to 13 (the code for the Enter key), and leave the NextAction empty.

Falling Through

This is the key: when you leave NextAction empty, and the Wait object detects a NextActivator, it simply lets execution of the page "fall through" to the next object(s). In the example above, execution falls through to the Program object and continues there.

In the Program object, you would put A-pex3 code that resembles:

```
response = Input(1).Text      $$ copy response into var
IF response =E= "yes" THEN
  BRANCH q2
ELSEIF response =E= "no" THEN
  BRANCH mainmenu
ELSE
  BRANCH q1
ENDIF
```

The program above retrieves the user's response by examining the Input object's Text attribute. It copies the response into a variable named response for convenience and speed (Everest processes variables more efficiently than attributes). Then it branches to different pages based on the response. The =E= operator tells Everest to compare the two items and ignore differences in case and spacing.

8.6 Retrieving Input Via Judge Object

Another way to retrieve and access a user's response in an Input or Mask object is via answer judging. Answer judging is the process by which Everest compares a user's response with a list of anticipated answers that you, as author, provide.

To perform answer judging, place a Judge object in the page and set the JudgeActivator attribute of the Wait object. Your page might resemble:

Object	Name	Comment
Page	**q1**	
Layout	q1_layout_A	window stuff
Erase	q1_erase_A	erase previous objects
Textbox	q1_textbox_A	the question text
Input	q1_input_A	user fill-in
Wait	q1_wait_A	wait for response
Judge	q1_judge_A	perform judging
Program	q1_program_A	branch upon response

In the Input object's ResponseVar attribute, enter the name of the variable in which to store the response. For example, you might use a variable named userresp. Also, set the Wait object's JudgeActivator to `13` (the code for the Enter key).

Jumping to Judge

When the Wait object detects a JudgeActivator event, execution of the page jumps to the next Judge object and continues there. When employing answer judging, most authors leave the Wait object's NextActivator empty, and only enter event codes for the JudgeActivator.

When Everest runs a Judge object, it performs answer judging. In this example, no anticipated answers have been specified with the Input object, and therefore, Everest judges all user responses as incorrect. But, in this example we only want to retrieve the user's response easily, so the judgment does not matter. Upon judgment, the response is placed in the variable you specify in the ResponseVar attribute. In this example, you set ResponseVar to the variable named userresp. The Program object might now resemble:

```
IF userresp = "yes" THEN
  BRANCH q2
ELSEIF userresp = "no" THEN
  BRANCH mainmenu
ENDIF
```

Note that there is no need to reference the Input object's Text attribute (as shown in the prior topic) because Everest has automatically placed the user's response in the variable you specified via the ResponseVar attribute.

Jump Back to Wait Object

In the Program object above, if the user's response is not yes or no, neither IF condition is matched, and execution falls through to the next object in the page. However, in this example, there is no next object. Everest handles this situation in a special way: it searches for the last Wait object in the page, and automatically jumps back to it. This feature makes it easy to wait for another response from the user.

Jump Back to Wait Object

In the Program object above, if the user's response is not yes or no, neither IF condition is matched, and execution falls through to the next object in the page. However, in this example, there is no next object. Everest handles this situation in a special way: it searches for the last Wait object in the Page, and automatically jumps back to it. This feature makes it easy to wait for another response from the user.

8.7 Judging Input

The previous topic shows how to use the Judge object to easily obtain a user's response. From there, it is a simple step to also judge the response for accuracy. You can enter anticipated (correct) answers with each interactive object, such as Input, Check, etc. You can also detect anticipated incorrect answers (in order to, for example, provide custom feedback).

Using Answers Attributes

All interactive objects offer either 2 or 8 Answers attributes. In these attributes you enter the correct answers(s) to the question. When there is more than one possible correct answer, you can use either of two approaches. With the first approach, you type all the possible correct answers into the same Answers attribute (Everest allows up to 250 characters per attribute). To do so, use the | character to separate the correct answers.

However, many authors employ the second approach: spread the possible correct answers across several Answers attributes (rather than list them all in one attribute). We hope you are wondering why, because we're about to tell you.

When Everest performs answer judging, it compares the user's response with the list of answers you provide in the Answers attributes. This is the key: when it finds a match in a particular Answers attribute, it stores the number (1 to 8) of that attribute in the Judgment attribute. Many authors use this number to easily determine which of the possible answers was matched.

An example: your question page asks the user to enter the name of a state in New England. In the Answers attributes of the Input object, you enter:

Answers1	`maine|me`
Answers2	`newhampshire|nh`
Answers3	`vermont|vt`
Answers4	`massachusetts|ma`
Answers5	`connecticut|ct`
Answers6	`rhodeisland|ri`

In this example, if the user enters "Vermont" as a response, Everest judges it as correct, and stores the numeric value 3 in the Judgment attribute because the Answers3 attribute contains the match. Many authors find such numeric values easier to use in programs, or to determine response feedback.

Everest searches the answers list from top to bottom until it finds a match. If Everest does not find a match in any Answers attribute, it stores 0 in the Judgment attribute.

Some notes: do not use upper-case letters or spaces in the Answers attributes if the AdjustResponse attribute is set to Yes (the default). Also, note in the example above that each attribute contains two possible answers: the name of the state, plus its postal abbreviation. Everest judges the user's response as correct if it matches either. The | character (ASCII 124) separates possible correct answers.

ResponseVar vs. JudgeVar

As a product of answer judging, Everest stores the user's response in the variable whose name you enter in the ResponseVar attribute, and puts the judgment result in the Judgment attribute. To summarize:

ResponseVar	Everest stores the user's response here
Judgment	Everest stores the judgment (correct/incorrect) here

The ResponseVar and Judgment attributes are provided solely as a convenience. Upon judging, Everest simply stores information in the variables you specify. It is up to you to make use of this information as you see fit. For example, you might embed the ResponseVar variable in a Textbox object to echo the user's response in feedback. The contents of such a Textbox might resemble:

```
Sorry, {userresp} is not the correct answer.  Recall that
the...
```

You might employ the Judgment attribute in A-pex3 programming to determine the appropriate branching destination (remediation for an incorrect response, advancement for a correct one). Such a program might resemble:

```
IF Input(1).Judgment = 0 THEN $$ no match
  BRANCH remedial
ELSEIF Input(1).Judgment < 0  $$ anticipated incorrect
  BRANCH feedback
ELSE                    $$ correct. move ahead
  BRANCH page2
ENDIF
```

8.8 Using Anticipated Incorrect

Everest's question objects also have AntIncorrect attributes; you can use them trap anticipated incorrect responses. You enter anticipated responses just as you do for Answers attributes. When the user's response matches an AntIncorrect answer, Everest negates the value it places in the Judgment attribute.

For example, if the user's response matches one in the AntIncorrect2 attribute, Everest stores -2 in the Judgment attribute. It does so to help you distinguish such matches from matches with the correct Answers attributes.

Everest compares the user's response with AntIncorrect answers after it exhausts comparisons with the Answers answers.

8.9 Using Ignore

In addition to Answers attributes, all question objects also have an Ignore attribute. Ignore is a handy way to tell Everest to disregard a response from the user. You specify the Ignore attribute using the same syntax as you do for the Answers attributes.

When a user's response matches one in the Ignore attribute, Everest does not perform answer judging, does not score the response and does not collect CMI data about it.

How might you use Ignore? When you want to disregard an empty response (i.e. no response) in an Input object, set Ignore to

```
""
```

(which matches an empty response). In a multiple choice question, you might want to disregard responses that are outside the range allowed. For example, if the range is A to D, you could set Ignore to

```
<a|>d
```

You can provide custom feedback for an ignored response by detecting the situation via the contents of the JudgeVar, which Everest sets to a null string ("").

8.10 More Answer Judging

At some time you will need more flexible answer judging than exact match. You might want to allow for misspellings, or for a numeric range. Everest offers several features to help; they are documented in the technical reference. Only the most commonly used ones are discussed here.

When you need to match any response from the user, enter * as the anticipated answer. The asterisk is a wild card that works much as it does in DOS or Windows for filename directory commands. Another example: to match any response that starts with the letter c, enter c* as an anticipated answer.

To accept a range of letters, use the pattern match feature. Activate pattern match by prefixing the anticipated answer with =P= as in this example:

```
=P=[a-dA-d]
```

which matches any letter from a to d, in both lower- and upper-case forms.

To accept a response that contains a word, use the =W= word search feature, as in this example:

```
=W=cartesian
```

which matches any response that contains the word cartesian.

To accept a response that sounds like a word, use the =S= phonetic match feature, as in this example:

```
=S=machine
```

which matches any response that sounds like the word machine.

Do Not Combine Judging Features

The various special features cannot be combined. For example, the following is an illegal anticipated answer:

```
=W==P=[a-d]*
```

8.11 Answer Judging Objects Other Than Input

While most of your answer judging will involve Input objects, it is also possible to judge user responses in other objects. Simply enter the anticipated answers with the same syntax as you use for Input objects.

It is important to know the form (number, character string, etc.) of the possible user responses generated by the various objects. Here is a list:

Object	Response Type/Range	Also In Attribute
Button	0 (up), -1 (down)	Value
Check	0, 1 or 2	Value
Combo	text in the top box	Text
HScroll	number from Min to Max	Value
Input	character string	Text

Listbox	list of items selected	TaggedList
Mask	character string	Text
Option	0 (empty), -1 (selected)	Value
VScroll	number from Min to Max	Value

For example, if the correct user response is the press of a certain Button object, you would enter -1 as the Answers1 attribute for that Button.

To learn more about the possible ranges, look in the technical reference/on-line help for the Attributes listed in the table above.

8.12 Providing Feedback

In Computer-Based Training, feedback is an important element. If the user enters an incorrect response, some type of corrective message is typically displayed. Perhaps the user is branched to remediation pages.

In Everest, you can build any type of feedback you want on a separate page, and branch to it based on the ResponseVar and Judgment values. But, a more elegant (and more easily maintained) approach is to place the feedback in the same page as the question.

To continue the example started in a previous topic, if you want to display a paragraph of text about the New England state whose name the user enters in the Input object, create a separate Textbox object for each paragraph. Place these Textbox objects after the Judge object. The page would now resemble:

Object	Name	Comment
Page	**q1**	
Layout	q1_layout_A	window stuff
Erase	q1_erase_A	erase previous objects
Textbox	q1_textbox_A	the question text
Input	q1_input_A	user fill-in
Wait	q1_wait_A	wait for response
Judge	q1_judge_A	perform judging
Textbox	q1_textbox_B	Maine description
Textbox	q1_textbox_C	New Hampshire
Textbox	q1_textbox_D	Vermont
Textbox	q1_textbox_E	Massachusetts
Textbox	q1_textbox_F	Connecticut
Textbox	q1_textbox_G	Rhode Island

Now, you might be wondering how Everest will know which of the six Textboxes to display. It's very easy. You tell it via the Condition attribute of each Textbox.

Recall that in this example the state names are listed in the anticipated Answers attributes. Upon judging, Everest stores the number of the Answers attribute that contains a match with the user's response. If the user enters "Vermont" (which is in the Answers3 attribute of the Input object) Everest sets the Judgment attribute of the Input object to 3.

Therefore, you would enter the following as the Condition attribute of the Textbox containing the description for Maine:

```
IF Input(1).Judgment = 1
```

because Maine appeared in the Answers1 attribute. Note that you omit the word THEN in the Condition attribute. For the New Hampshire Textbox:

```
IF Input(1).Judgment = 2
```

and so on, for each Textbox.

How Condition Works

At design time you see all six Textboxes. At run time, Everest only displays the one with the true Condition. The Condition attribute gives you a very powerful way to control which page objects are displayed, and which are not. Don't miss it!

Feedback via Programming

If you have many different possible feedback messages, it can be tedious to create a Textbox for each. As an alternative, create just one Textbox, and use a Program object to set its Text attribute as needed. In the New England States example, the Program object would resemble:

```
IF Input(1).Judgment = 1 THEN
  Textbox(2).Text = "Maine is the northernmost of..."
ELSEIF Input(1).Judgment = 2 THEN
  Textbox(2).Text = "New Hampshire's capital is..."
.
.
.
ELSEIF Input(1).Judgment = 6 THEN
  Textbox(2).Text = "Rhode Island is our smallest..."
ENDIF
```

Recall that when Everest reaches the end of a page, it searches for the Wait object nearest the end, and jumps back to it to wait for more input (i.e. for the user to try again). Of course, if you prefer to prevent this, do not let Everest run the page to the end...instead JUMP back to a previous object, or put another Wait object at the end.

8.13 Counting Tries

In a CBT project, you may want to offer the user a certain number of attempts to answer a question correctly. The Tries attribute lets you do so.

Each time Everest judges a user's response (via the Judge object) the Tries attribute for the question is decremented. When Tries reaches 0, Everest disables the question by setting its Enabled attribute to No.

A very common CBT design provides feedback to users after an incorrect response, then allows them to try the question again, if tries remain. This is quite easy to create in Everest. In the New England states example, you add a JLabel object and a JUMP command. You would also set the Tries attribute of the Input object to something like, say, 3. Your page would resemble:

Object	Name	Comment
Page	**q1**	
Layout	q1_layout_A	window stuff
Erase	q1_erase_A	erase previous objects
Textbox	q1_textbox_A	the question text
Input	q1_input_A	user fill-in
JLabel	again	jump here for another try
Wait	q1_wait_A	wait for response
Judge	q1_judge_A	perform judging
Textbox	q1_textbox_B	Maine description
Textbox	q1_textbox_C	New Hampshire
Textbox	q1_textbox_D	Vermont
Textbox	q1_textbox_E	Massachusetts
Textbox	q1_textbox_F	Connecticut
Textbox	q1_textbox_G	Rhode Island
Program	q1_program_A	jump to again if more tries

In the Program object, enter A-pex3 programming that resembles:

```
IF Input(1).Tries > 0 THEN JUMP again
BRANCH q2               $$ else
```

Remember, Everest decrements Tries upon each attempt. While Tries is greater than 0, additional attempts remain. The JUMP command causes Page execution to jump back to the JLabel, which then continues with the Wait object (waiting for another attempt).

When the user answers correctly, Everest sets Tries to -1 and disables the question.

Offering Hints

If you want, you can provide feedback customized to the number of attempts that user has made. For example, after the first attempt you might give a tiny hint, and after the second, a more descriptive hint, etc. To do so, simply reference the Tries attribute in either A-pex3 programming or in an object's Condition attribute. The only tricky part is to remember that Tries is decremented. For example, if you set Tries to 3 with the object, after the first incorrect attempt, Tries becomes 2. After the second, Tries becomes 1.

8.14 Keeping Score: Level 1

In Computer-Based Training and similar projects, you might want to keep track of the users score across a series of questions. Everest can automatically count the number of questions a user attempts to answer, as well as the number he answers correctly, and compute a total score.

To activate Everest’s built-in scoring system, all you need to do is tell it when to start counting.

Level 1 vs. Level 2

Everest offers two scoring systems, called Level 1 and Level 2. The Level 1 scoring system (the default) is easy and automatic. Level 2 requires a tiny bit of programming with each question page, but is more flexible.

To use Level 1 scoring, your questions must allow the user exactly one try to answer correctly. If you allow the user multiple tries, you must employ the more complex Level 2 system (described later in this chapter), or a scoring system of your own.

Employing Level 1 Scoring

Here's the process to employ Level 1 scoring.

1) Set the Tries attribute to `1` for each question you want to score. Leave the Tries attribute empty for interactive objects, such as navigation Buttons, that you do not want to score.
2) If your questions are of the multiple choice variety where the user clicks one of several Button or Option objects, set the Judge object's Grouped attribute to `Yes`.
3) Prior to asking the user the very first question, activate Level 1 scoring by resetting several Sysvars() via calculations as shown here:

```
sysvar(106) = 0      $$ enable Level 1 scoring
sysvar(5) = 0        $$ clear previous # correct
sysvar(6) = 0        $$ clear previous # attempted
sysvar(7) = 0        $$ clear previous score
```

   in a Program object. You might put this Program object on a page that precedes the first that contains the questions you want to score.

How it Works

With Level 1 scoring, upon judging the user responses via a Judge object, Everest totals the number of questions attempted in Sysvar(6), and the number answered correctly in Sysvar(5). It also computes a current score, and stores it in Sysvar(7). At the end of the exam, you might use an A-pex3 program similar to the following to branch to a page that displays the results to the user:

```
IF sysvar(7) >= 70 THEN
  BRANCH passed
ELSE
  BRANCH failed
ENDIF
```

It is important to realize that Everest keeps incrementing the values in Sysvar(5) and Sysvar(6) with each question, even if those questions appear on multiple pages. This allows you to easily construct an exam that consists of multiple questions on separate pages.

Just the Current Page

If you simply want to know how many questions the user answered correctly and attempted for the most recent Judge object, then reference Sysvar(175) and Sysvar(176) respectively immediately after judging. This approach requires no special settings in Sysvar(106), or Sysvar(5) to Sysvar(7).

8.15 Keeping Score: Level 2

If your project's questions allow the user more than one attempt to answer correctly, and you want Everest to keep score for you, you must use the Level 2 scoring system. For simplicity, Level 1 assumes that questions will be asked only once. If a question has multiple Tries, Level 1 does not know how many attempts to count.

With Level 2 scoring, you tell Everest when it can tally the results. If you want, you can tell it to do so after each attempt, only after Tries are exhausted, or whenever. For example, in an exam with questions on several pages, you might tell Everest to tally the results just before branching to the next question page.

Employing Level 2 Scoring

Here's how to enable Level 2 scoring

1) Before the start of the series of questions, use a Program object to set several Sysvars(). You might put this Program into a page that introduces the exam.

```
sysvar(105) = 0            $ clear prev total correct
sysvar(106) = -1           $ enable Level 2 scoring
sysvar(107) = 0            $ clear prev total score
```

2) During the exam, each time you want to tally the results of a question page, reference function ext(105) after the Judge object has been performed. You can put this in a Program or Wait object:

```
newscore = ext(105)
```

If you are tallying results, and branching to the next question page in one step, you can do this on the same line (in a Wait object's NextAction, for example):

```
newscore = ext(105): BRANCH q2
```

What Ext(105) Does

The Ext(105) function does several things at once: 1) it accumulates the total correct and attempted questions in the Sysvar(105) and Sysvar(106) variables, 2) it resets the Sysvar(5) and Sysvar(6) variables to 0 (which held the results of the most recent Judge object), 3) it stores the current score in Sysvar(107), and 4) it returns the score computed so far.

8.16 Doing Your Own Scoring

Both of Everest's built-in scoring systems (Level 1 and Level 2) count every question on every page when figuring the score. Perhaps you want to count each "pageful" of questions as one question. Or, perhaps you want to vary the weight of the questions when figuring the score.

In such situations you need to perform your own scoring. If you are familiar with A-pex3 programming, scoring is actually quite easy. After judging each response, use a Program object to tally the results as you need in one or more variables.

8.17 Creating a Multiple Choice Question

A multiple choice question can be created in any one of several ways. Here are some ideas:

1) Use an Input object to accept a letter or number response from the user. Set the TextLength attribute to `1`, and the InputTemplate to either `a` or `9`. If you want to judge the user's response immediately upon input, set the EOFEvent attribute to the same code as you specify for the JudgeActivator attribute of the Wait object.
2) Place a Button object next to each choice displayed in a Textbox. More details are given below.
3) Put the choices in a Listbox. Set the ClickEvent attribute to the same event code you specify for the JudgeActivator attribute of the Wait object. If you want the user to choose only one item in the list, set TagStyle to `0`. The response returned is a string of spaces and X characters indicating which item(s) were chosen. See the TaggedList attribute for more information.
4) If only one choice is allowed, use several Option buttons. Set the ClickEvent attribute to the same event code you specify for the JudgeActivator attribute of the Wait object.

Multiple Choice Via Buttons

Many authors employ Button objects for multiple choice questions. The following details describe how to create and score a question with four possible choices (A to D) when C is the correct choice.

1) Start a new page. Place four Textbox objects on it, one for each choice. Enter text that describes each choice.
2) Put four Button objects on the page. Position each next to a Textbox.
3) Set the Caption attribute of the first Button (the one with IDNumber 1) to `A`, of the second (the one with IDNumber 2) to `B`, etc.

4) When the user clicks any of the Buttons, you want to judge the response. To trigger judging, each Button needs a ClickEvent. To later help identify which Button was clicked, use a different event code for each. So, set the ClickEvent of the first Button to `-65`, of the second to `-66`, of the third to `-67` and of the fourth to `-68`. Use negative numbers to avoid confusion with keypress event codes, which are always positive numbers.

5) When a Button is clicked, its value (for judging purposes) is -1. In this example, the correct answer is C. So, set the Answers1 attribute for Button C to `-1`.

6) To enable Everest's built-in scoring system, you must set the Tries attribute. Set Tries for all four Buttons to `10` (infinite retry).

7) Add a Wait object. Make sure it appears after the Buttons in the page.

8) Set the Wait object's JudgeActivator1 to `-65,-66,-67,-68` (the ClickEvents of the Buttons).

9) Add a Judge object. Make sure it appears after the Wait object in the page.

10 Important: upon answer judging, by default Everest examines all interactive objects independently. Therefore, it will score the four Buttons separately. This is undesirable for this question because the user will click only one Button (per try). The solution: tell Everest to judge all the Buttons as a unit. To do so, set the Judge object's Grouped attribute to `Yes`.

11) To see the results of the automatic scoring, add a Textbox to the page. Make sure it appears after the Judge object in the page. Inside the Textbox, embed three system variables by entering:

```
Correct: {sysvar(5)}

Attempts: {sysvar(6)}

Score:  {sysvar(7)}
```

12) Save your page to disk, then run a Preview.

13) During the Preview, click in sequence on Button A, B, C and D. You should observe the number of attempts incrementing with each click, and the number correct incrementing when you click on Button C.

Note that if you run the Preview again, system variables 5 through 7 will start with their previous values. If you want to reset them each time this page is run, add a Program object to the top of the page, and enter the following programming into it:

```
sysvar(5) = 0: sysvar(6) = 0
```

Recognizing Keypresses

Perhaps you would like to allow the user the option of pressing A, B, C or D on the keyboard. Here's how:

1) For all four Buttons, set the HoldDown attribute to `Yes`. This will make the Button remain depressed when the user selects it. This setting is also necessary to make the following steps work properly.

2) In the Wait object, set the Other1Activator to `65,66,67,68`. These are the event codes for the letters a, b, c and d on the keyboard.

3) Set the Other1Action to

   ```
   Button(sysvar(12)-64).State = -1
   ```

   This little trick works because Sysvar(12) contains the event code of the most recent event. Sysvar(12) is going to be either 65, 66, 67 or 68 since these are the events codes trapped via Other1Activator. Event code 65 is the letter "a" key. Subtracting 64 from that leaves 1, which is the IDNumber of the first Button, Button A. When you set the Button's State to -1 via programming, Everest thinks you clicked with the mouse to depress the Button. Doing so triggers the Button's ClickEvent, and that produces answer judging as before.

Collecting CMI Data

Enabling the CMIData attribute tells Everest to automatically record the user's response, as well as the answer judgment rendered. To make this work, enable CMIData for each Button. However, be sure you also enable the Judge object's Grouped attribute. If you forget to enable Grouped, Everest will dutifully save the state of all four Buttons, as well as the answer judgment for all four, just as if there had been four separate questions on the page.

Randomizing Choices

To discourage users from exchanging the answers to multiple choice questions, many authors randomly scramble the order of the choices. You can do this by swapping the text of the choices, as well as the answer judging attributes of the Buttons. The following A-pex3 code illustrates (place it in a Program object located in the page immediately before the Wait).

```
numq = 4          $$ number of choices
fromobj = 0
DO IF fromobj < numq    $$ loop once per choice
  fromobj++
  toobj = rnd(numq) + 1 $$ random number 1 to 4
  IF fromobj # toobj THEN

    $$ swap choice descriptions
    was = Textbox(fromobj).Text
    Textbox(fromobj).Text = Textbox(toobj).Text
    Textbox(toobj).Text = was

    $$ swap anticipated incorrect
    was = Button(fromobj).AntIncorrect1
    Button(fromobj).AntIncorrect1 = Button(toobj).AntIncorrect1
    Button(toobj).AntIncorrect1 = was

    $$ swap anticipated answers
    was = Button(fromobj).Answers1
    Button(fromobj).Answers1 = Button(toobj).Answers1
    Button(toobj).Answers1 = was
  ENDIF
LOOP
```

The A-pex3 program above assumes the choice Textboxes and Buttons on your page have IDNumbers 1 to 4. The swapping of the Textbox contents might be seen (as flicker) at run time; to avoid this flickering, enable the Layout object's LockUpdate attribute.

Easier Randomization

If your question page simply displays the possible choices via the Caption attributes of the Buttons (thereby removing the need for a separate object, such as a Textbox, to display each choice), there is a much easier way to randomize. Simply use this program in place of the one above:

```
dummyvar = sfl("Button", 1, 4, "Caption", "Answers1")
```

This program employs the shuffle function, which automatically does all the random swapping for you. See the Sfl() function documentation for details.

8.18 Creating a True/False Question

A true/false question is actually just a restricted multiple choice question. There are several ways you can create a true/false question. A few are described below, along with the names of attributes that are often useful (see the technical reference/on-line help for more information).

Input Object

The Input object allows the user to type any response. You can limit the entry to a single character by setting TextLength to 1. You might also set InputTemplate to the letter a to force a character (rather than a digit) to be entered.

Still, the user might enter a letter other than T or F. For answer judging, tell Everest to ignore anything that is not T of F. Simply set the Ignore attribute to

```
#t#f
```

To invoke judging immediately after the user presses either T or F, employ the EOFEvent attribute.

Mask Object

The Mask object is very similar to the Input. For true/false questions, it offers no advantage over Input.

Check Object

The Check object is not often used for traditional true/false questions. However, you might, for example, use one next to each item of a list in a question that asks "Mark the following statements that are true."

Option Object

The Option object is more suited to true/false type questions. Use two Option objects for each question, and put the pair in the same Group. Set the Caption of one of the objects in the pair to "True" and the other to "False."

Using and Judging Option Objects

If you are using Everest's built-in scoring systems, here are step-by-step details on how to build a true/false question with judging. The following technique applies to any multiple choice question, including those with more than just two choices.

1) Start a new page. Place your normal objects in it as needed (Layout, Erase, etc.).
2) Put a Textbox on the page, and type the question into it.
3) Put two Option objects on the page.
4) Set the Caption attribute of the first Option object to `True`, and that of the second to `False`.
5) When the user clicks one of the Options, you want to judge the response. To trigger judging, each Option needs a ClickEvent. Set the ClickEvent of both Option objects to `"clicked"` (include the quotes).
6) To enable Everest's built-in scoring system, you must set the Tries attribute. Set Tries for both Options to `10` (for infinite retry).
7) To prevent one of the Option objects from being in a highlighted state initially, set the Preset attribute for both to `0`.
8) When an Option is clicked, its value (for judging purposes) is -1 (you know this from the documentation for the ResponseVar attribute). Let's say the correct answer for this question is True. So, set the Answers1 attribute for the True Option to `-1`.
9) Add a Wait object. Make sure it appears after the Options in the page.
10) Set the Wait object's JudgeActivator to `"clicked"` (include the quotes).
11) Add a Judge object. Make sure it appears after the Wait object in the page.
12) Important: upon answer judging, by default Everest examines all interactive objects independently. Therefore, it will score the two Options separately. This is undesirable for this question because the user will click only one Option (per try). The solution: tell Everest to judge both Options as a unit. To do so, set the Judge object's Grouped attribute to `Yes`.
13) To see the results of the automatic scoring, add a Textbox to the page. Make sure it appears after the Judge object in the page. Inside the Textbox, embed three system variables by entering:

    ```
    Correct: {sysvar(5)}

    Attempts: {sysvar(6)}

    Score:  {sysvar(7)}
    ```
14) Save your page to disk, then run a Preview.
15) During the Preview, try clicking both Options. You should observe the number of attempts incrementing with each click, and the number correct incrementing when you click on the True Option. Stop the Preview when you are done testing.
16) In actual use, you don't want the user to be able to repeatedly click the Option buttons. To do so, change the Tries attribute for both Option objects to `1`. Also, set the Judge object's DisableObjs attribute to `Yes`.

Note that if you run the Preview again, system variables 5 through 7 will start with their previous values. If you want to reset them each time this page is run, add a Program object to the top of the page, and enter the following programming into it:

```
sysvar(5) = 0: sysvar(6) = 0
```

Recognizing Keypresses

Perhaps you would like to allow the user the option of pressing t or f on the keyboard. Here's how:

1) In the Wait object that appears before the Judge, set the Other1Activator to `84` (the event code for the letter t) and Other1Action to:

   ```
   Option(1).State = -1
   ```

2) Also, set the Other2Activator to `70` (the event code for the letter f) and Other2Action to:

   ```
   Option(2).State = -1
   ```

This little trick works because when you set the Option's State to -1 via programming, Everest thinks you clicked with the mouse to depress it. Doing so triggers the Option's ClickEvent, and that produces answer judging as before.

Branching After Judgment

If you simply want to branch to the next page after the user makes a selection, place a Wait object (that will make two within the page) after the Judge. Set the Wait object's Pause attribute to `N` (for no pause) and the NextAction attribute to the desired BRANCH command.

8.19 Randomizing Exam Questions

A common exam approach in CBT is one that involves asking questions chosen at random from a bank of available questions. To accomplish this, Everest uses a technique very similar to that found in Summit for DOS. Here are the steps:

1) Create a pool of question pages with names that consist of a letter followed by a number. For example, name the first page `q1`, the second `q2`, etc.

2) In each question page, set the Wait object's NextAction to branch to a main page. For example, if you name the main page `selectq`, the NextAction should read `BRANCH selectq`.

3) Create an exam initialization page. You could name it `initexam`. Set it to branch to selectq. The initexam page will be where you start the exam.

4) In a Program object in the initexam page, initialize a variable to contain blank spaces. The number of spaces in the variable should equal the total number of questions in the pool (not the number to extract, which is less than or equal to the number in the pool). This variable will be used by the Sel() function to track which question pages have been used previously. You might name the variable qflags. Also initialize a question counter variable; name it qcount. If the pool has 100 questions, your Program object should resemble:

```
qflags = 32$100   $$ 100 ASCII 32 (space) flags
qcount = 0        $$ question counter
BRANCH selectq    $$ start asking questions
```

If you are employing Level 1 or Level 2 scoring, you can initialize the appropriate Sysvars() here too.

5) In a Program object in the selectq page, check if the qcount variable has reached the number of questions you want in the exam (say, 20), and if so, branch to a score display page (which you need to create). Otherwise, use the Sel() function to generate a random number and employ the number in a BRANCH command. Your programming should resemble:

```
IF qcount = 20 THEN BRANCH score
qcount++                $$ increment question count
qno = sel(qflags)       $$ a unique question number
qno = "q" $+ qno        $$ page name letter
BRANCH {qno}            $$ branch to that question
```

Using Variables in Branching Commands

The example above shows the variable named qno embedded in a BRANCH command. We recommend you do not make extensive use of this feature because it tends to make your project difficult to debug. Also, Everest cannot branch test such commands because the name of the page to which to branch is determined only at run time. For best results, employ only lower-case letters for the names of variables embedded in branching commands.

Randomizing Choices

For an example of how to randomize the choices within a particular multiple choice question page, refer to the prior topic titled "Creating a Multiple Choice Question."

8.20 Easy Interaction Via Ibx() and Mbx()

For brief messages or quick input, who wants to create an entire page? Two functions come to the rescue: Ibx(), the input box, and Mbx() the message box. Both display a small message (of your choice) and return user input. The Ibx() function returns a text string. The Mbx() function returns a number that indicates which button (OK, Cancel, Yes, No, etc.) the user clicked.

You can use both Mbx() and Ibx() anywhere A-pex3 programming is allowed, such as in Program objects, the xxxAction attributes of Wait objects, or xxxEvent attributes.

For example, if you wanted to verify a user's action when he clicks on a Button, in the Button's ClickEvent, you would enter something similar to (all on one line):

```
IF Mbx("Are you sure?", 36) = 6 THEN ERASE (3)
```

Refer to the technical reference for operational details about Ibx() and Mbx().

8.21 Handling Point & Click Interaction

Another common form of interaction is the so-called "point and click." It is employed when you want the user to click the mouse at a certain location or object on the page. Everest makes it easy to detect clicks on objects because it lets you determine the ClickEvent that is generated, as well as the action to perform. This is described in the Events chapter.

Highlighting Objects

A common user interface technique involves changing the color of an object when the user moves the mouse over it. This is easy to implement in Everest thanks to the MouseOverEvent and MouseLeaveEvent attributes. The MouseOverEvent triggers when the mouse cursor enters an object, and MouseLeaveEvent triggers when it exits.

For example, if you want to change the background color of the Button with IDNumber 1 to red when the mouse is positioned over it, set its MouseOverEvent to

```
Button(1).FillColor = rgb(255, 0, 0)
```

To change the background color to gray when the mouse cursor moves off the Button with IDNumber 1, set its MouseLeaveEvent to

```
Button(1).FillColor = rgb(128, 128, 128)
```

Specific Click Locations

The ClickEvent fires when a user clicks the mouse on an object. If you want to determine where the user clicked WITHIN an object, you'll need to do a bit more work. For example, if you display an image in a Picture object and want to know on what portion of the picture the user clicked, you can do so via Shape objects or A-pex3 programming.

Via Shape Objects

Everest processes user mouse clicks on Shape objects first, even if the Shapes are visually obscured by objects layered on top of them. You can take advantage of this feature to easily detect mouse clicks on portions of a Picture (or any object).

Using the VisualPage editor, simply place one or more Shape objects in the desired locations on the Picture, set their ClickEvent attributes to unique values, and detect these values with a Wait object. While editing, you will only be able to see the sizing handles around each Shape (since the Shapes themselves will be obscured by the Picture object). To edit a different Shape, first click on it in the Book Editor.

If you would like to visually prompt the user when they move the mouse over a special area, tell Everest to display a different mouse cursor via the ShapePointer attribute. The ShapePointer attribute has the highest priority for mouse cursor appearance.

If you are using Shapes over a non-obscuring image (such as the BgndPicture), you can make the Shapes invisible (if desired) by setting their OutlineStyle to 0, and FillStyle to 1.

Via A-pex3 Programming

Instead of Shape objects, you can employ A-pex3 programming to detect where the user clicks on an object such as a Picture; this approach is more difficult. As an example, say you want to know if the user clicked in the upper-left, upper-right, lower-left or lower-right quadrant of a Picture object. Your page might resemble:

Object	Name	Comment
Page	**click**	
Picture	click_picture_A	display picture
Wait	click_wait_A	wait for click
Program	click_program_A	process click

In the Picture object, invent an event code and enter it for the ClickEvent. Enter this same event code in the Wait object's NextActivator attribute. Leave the NextAction attribute empty so execution will fall through to the Program object. In the Program object, enter A-pex3 code that resembles:

```
clickx = sysvar(9) - Picture(1).Left
clicky = sysvar(10) - Picture(1).Top
areax = Picture(1).Width / 2
areay = Picture(1).Height / 2
clickspot = reg(clickx, clicky)
IF clickspot =T= reg(0, 0, areax, areay) THEN
  area = "Upper-left"
ELSEIF clickspot =T= reg(areax, 0, 2*areax, areay) THEN
  area = "Upper-right"
ELSEIF clickspot =T= reg(0, areay, areax, 2*areay) THEN
  area = "Lower-left"
ELSEIF clickspot=T=reg(areax,areay,2*areax,2*areay) THEN
  area = "Lower-right"
ELSE
  area = "None!  Something is wrong."
ENDIF
dummyvar = mbx(area, 64)
```

The program above retrieves the mouse click location from the Sysvar(9) and Sysvar(10) variables. The values in these variables represent the location (in pixels) of the most recent mouse click, relative to the upper-left corner of the display area of the window. To determine the location relative to the Picture object, the program subtracts the location of the Picture object from the mouse click location. Next, the width and height of the 4 quadrants are calculated and stored in the areax and areay variables, respectively. With help from the Reg() function and =T= operator, it is easy to determine the quadrant the user clicked.

Via Color

Another technique you can employ, and one that works especially well on irregularly shaped areas, is to retrieve the color of the pixel. The following obtains the RGB color of the pixel at which the user last clicked the mouse.

```
colr = ext(1, sysvar(9), sysvar(10))
```

Use this technique with care because the RGB value returned can vary based on the current Microsoft Windows color depth setting.

Balloon Help

Shape objects have yet another special attribute: MouseStayEvent. The MouseStayEvent fires if the user leaves the mouse pointer over the Shape for one second (to alter the amount of time, set Sysvar(173) to the desired number of seconds). MouseStayEvent is handy for creating so-called "balloon help" in your project. Balloon help is the term used to describe a help message that appears only after the user leaves the mouse cursor over a certain area for a period of time.

8.22 Using Timers

Perhaps you want to monitor the value of a system variable that Everest updates automatically. For example, you might want to check the location of the mouse cursor relative to the window. To do so, employ the Timer object to repeatedly generate an event at a particular time interval, then do what you want upon the TimeEvent.

For example, to constantly display the current location of the mouse cursor, put a Textbox with IDNumber 1 on the page, then put a Timer object in the Page, set its Period attribute to .1 (one tenth of a second), and set its TimeEvent attribute to:

```
Textbox(1).Text = "X="+sysvar(9)+", Y="+sysvar(10)
```

When you run this page, the Timer generates an event every 0.1 seconds. The TimeEvent attribute updates the Textbox to display the current mouse cursor location.

You can choose to generate events more or less frequently than every 0.1 seconds. However, not that if you generate events too frequently, the computer will not be able to process them fast enough to keep up.

Displaying the Current Time

To display a digital clock that updates continuously, put a Textbox with IDNumber 1 on the page, followed by a Timer object. Set the Timer's Period to 1 (which means 1 second), and the TimeEvent to:

```
Textbox(1).Text = tim("")
```

8.23 Creating a Drag and Drop Interface

Via Everest's drag and drop features, you can allow users to move an object on the page via the mouse, and detect where they dropped it. For example, you might have a page with three draggable Picture objects and two Textbox objects. You can detect when the user drops a Picture on a Textbox.

STEP 1: allow the objects to be dragged. In this example, that's done by setting the DragMode attribute of each Picture (assume they have IDNumbers1, 2 and 3) via A-pex3 programming as shown:

```
Picture(1).DragMode = 1
Picture(2).DragMode = 1
Picture(3).DragMode = 1
```

STEP 2: invent a DragDropEvent code. When the user drops an object (by releasing the mouse button), Everest generates the DragDropEvent code you specify with the Layout object. Add a Layout object to the Page (if one does not yet exist), and set its DragDropEvent attribute to -100 (or whatever event code you prefer).

STEP 3: detect the DragDropEvent. Add a Wait object to the Page, and set the NextActivator to the DragDropEvent code (-100), Leave the NextAction attribute empty to tell Everest to fall through to the next object in the Page.

STEP 4: process the DragDropEvent. Put a Program object after the Wait object. In the Program, you can determine which object was dragged and where it was dropped by examining certain Sysvars. Then you can perform any desired action. For example, the following A-pex3 example clears the image from the Picture box if the user drops it on the Textbox with IDNumber 1:

```
onobj = sysvar(111): onid = sysvar(112)
dragobj = sysvar(113): dragid = sysvar(114)
IF onobj = 84 & onid = 1 & dragobj = 86 THEN
  Picture(dragid).PictureFile = ""
ENDIF
```

The values 84 and 86 in the example above are code numbers that represent the Textbox and Picture box object classes, respectively. A list of object class codes can be found with the Obj() function in the Technical Reference, and is reproduced below for convenience:

Object Number	Class Name
50	Layout (the window)
65	Animate
66	PicBin
67	Program
70	Frame
71	Gauge
72	HyperHlp
73	Line
74	Judge
75	Timer
76	JLabel
77	Media
78	Flextext
79	OLE
80	SPicture
83	Special (now obsolete)
84	Textbox
85	Erase
86	Picture
87	Wait
88	Include
90	Shape
98	Button
99	Check
101	Mask
104	HScroll
105	Input
107	Combo
108	ListBox
109	Menu
111	Option

118	VScroll

Moving an Object to a Dropped Location

It is possible to move an object to the location at which the user dropped it. This is trickier than it might sound because you must adjust for the position of the mouse when the user initially drags the object. Everest stores the initial mouse position in Sysvar(159) and Sysvar(160). The following example, typically used in response to a DragDropEvent, shows how to move the Picture object with IDNumber 1 to the location at which the user dropped it:

```
Picture(1).Move = reg(Picture(1).Left + sysvar(9) -
sysvar(159), Picture(1).Top + sysvar(10) - sysvar(160))
```

(all on one line).

8.24 Using Single-Column Listboxes

The Listbox object is a very handy container for a list of items. Listboxes allow the user to scroll through items and easily choose one or more. Everest's Listbox object supports both single-column lists, and multi-column lists. Single column lists are discussed in this topic, multi-column lists can be found in the next.

Adding Items at Design Time

Perhaps the most important attribute of a Listbox is the ItemList. At design time, you use ItemList to specify the list of items. To do so, enter a the items separated by a unique delimiter character of your choice; specify the delimiter as the first character. For example:

```
;Robins;Blue Jays;Cardinals;Red Finches;Juncos
```

which uses ; as the delimiter between types of birds.

Controlling Number Selectable

By default, the Listbox allows the user to select (i.e. highlight or tag) one item in the list. When the user selects another, the highlight is automatically removed from the prior item.

If you want the user to be able to select more than one item, do so by setting the TagStyle attribute to either 1 or 2.

Detecting Which Line Was Clicked

To determine which Listbox line the user most recently clicked on, employ the ItemIndex attribute. For example, you might set the Listbox's ClickEvent to:

```
dummy = mbx("You clicked line " + Listbox(1).ItemIndex)
```

Detecting Which Items Are Selected

If your Listbox allows the user to highlight more than one line at a time, you can use either of two techniques to determine which lines the user selected: 1) examine the Tagged attribute for each line, one by one, or 2) retrieve the TaggedList, a list that indicates which are selected.

Using the Tagged Attribute

The following example examines each item in the Listbox with IDNumber 1 and, if it is selected, copies that item into the array named "chosen". Note how the first item in the list is number 0.

```
REDIM chosen(Listbox(1).TaggedCount)
ptr = 0: max = Listbox(1).ItemCount
count = 0
DO IF ptr < max
  Listbox(1).LookAt = ptr
  IF Listbox(1).Tagged # 0 THEN
    count++
    chosen(count) = Listbox(1).Item
  ENDIF
  ptr++
LOOP
```

Using TaggedList

The following example does the same as the previous, but employs the TaggedList attribute instead. TaggedList returns a string of X and O characters, one for each item in the list. An X means the corresponding item was selected. For processing a large list of items, the following example runs faster than the previous.

```
REDIM chosen(Listbox(1).TaggedCount)
max = Listbox(1).ItemCount
marked = Listbox(1).TaggedList
ptr = 1: count = 0
DO IF ptr <= max
  IF marked ^^ ptr = "X" THEN
    count++
    Listbox(1).LookAt = ptr - 1
    chosen(count) = Listbox(1).Item
  ENDIF
  ptr++
LOOP
```

You can also assign a value to TaggedList at run time to set the selection status of the items. Refer to the technical reference for details.

Adding Items at Run Time

Items can be added to Listboxes at run time. This is best illustrated via examples. To replace the entire list with another, you can use the ItemList attribute as follows:

```
Listbox(1).ItemList = ";Robins;Blue Jays;Cardinals;Red
Finches;Juncos"
```

(all on one line).

To add items without removing those already in the Listbox, use the AddItem attribute, multiple times if necessary. For example:

```
Listbox(1).AddItem = "Robins"
.AddItem = "Blue Jays"
.AddItem = "Cardinals"
.AddItem = "Red Finches"
.AddItem = "Juncos"
```

Note that "Listbox(1)" was omitted on lines 2 to 5. This is simply a faster way to access the last referenced object.

Deleting Items at Run Time

Use the RemoveItem attribute to delete an item from the list. Set RemoveItem equal to the number of the item to remove (the first item in the list is number 0). The following example removes the items that are selected (highlighted) in the Listbox with IDNumber 1:

```
marked = Listbox(1).TaggedList
ptr = len(marked)
DO IF ptr > 0
  IF marked ^^ ptr = "X" THEN
    Listbox(1).RemoveItem = ptr - 1
  ENDIF
  ptr--
LOOP
```

Notice how the example above works backwards (i.e. starting with the last item in the list). If you started with the first, upon removing an item all the remaining items would shift location, and thereby confuse future RemoveItem references.

To remove all the items quickly, set RemoveItem to -1, for example:

```
Listbox(1).RemoveItem = -1    $$ delete all
```

Finding Items at Run Time

To find an item within a Listbox, use the FoundIndex and FindString attributes. The following example looks for the string "Cardinals" in the Listbox with IDNumber 1 and replaces it with "Mockingbirds":

```
Listbox(1).FoundIndex = -1    $$ start at top
Listbox(1).FindString = "Cardinals"
IF Listbox(1).FoundIndex >= 0 THEN  $$ if found
  Listbox(1).LookAt = Listbox(1).FoundIndex
  Listbox(1).Item = "Mockingbirds"
ENDIF
```

The Pik() Function

For additional, handy list processing features, refer to the Pik() function in the technical reference.

8.25 Using Multi-Column Listboxes

Use a multi-column Listbox when you want to display items in a table or grid form. In general, the techniques discussed in the previous topic for handling single-column Listboxes also apply to multi-column Listboxes. This topic discusses issues unique to multi-column Listboxes.

Creating the Columns

By default, Listboxes contain one column. To tell Everest you want additional columns, use the ColCount attribute.

Same Sized Columns

If all the columns in your Listbox have the same width, then simply set ColCount to the number of columns (2 to 49) you want.

Differently Sized Columns

There are two ways you can tell Everest that you want columns with different widths.

By Pixels

In the ColCount attribute, simply enter the width of each column (counting from left to right), in pixels. So, for example, if you set ColCount to `30,40,50` you'll get a three-column Listbox with columns that are wider left-to-right.

By Percentage

In the ColCount attribute, enter the width of the columns expressed as a percentage of the width of the Listbox. Express via negative numbers to distinguish this method from the previous. The numbers you enter should total 100 (or more accurately, -100). So , for example, if you set ColCount to `-35,-65` you'll get a two-column Listbox in which the right column is wider than the left.

Entering the Items

As with a single-column Listbox, you can specify the items to display via the ItemList attribute. However, you must pick a character that will act as a separator between items you want to appear on the same line in the Listbox. You specify the character of your choice by entering its ASCII code in the ColChar attribute. Many people use a comma, which is ASCII code 44, so you would set ColChar to 44.

Specifying the ItemList

When specifying the items via the ItemList attribute, not only do you need to employ the character that separates the lines (many people use a semicolon), you also need to employ the

ColChar character. The ItemList for a two-column inventory Listbox that uses a comma for ColChar might resemble:

```
;pads,14;pencils,7;pens,4
```

Loading from a File

You can also load items into the Listbox from a disk file with help from Fyl() function operation 9.

Comma-Delimited Format

Many simple, text database files are stored in so-called "comma-delimited format." This simply means there are commas between the items. For example, if you viewed such a text file, call it TOOLS.TXT, via a word processor or text editor, it would resemble:

hammers,22,$4.95
screwdrivers,10,$8.49
wrenches,15,$12.95

Here's how to easily load a comma-delimited file, like the fictitious TOOLS.TXT above, into a multi-column Listbox: in the Attributes window for the Listbox, set ColCount to the number of columns, ColChar to `44`, and ItemList to:

```
{fyl(9, 1, "tools.txt", chr(13))}
```

If you do this via a Program object, your A-pex3 code would resemble:

```
Listbox(1).ItemList = fyl(9, 1, "tools.txt", chr(13))
```

Note that Fyl() function operation 9 is limited to files under 32000 bytes in size.

Accessing Cells

When a user clicks on a line in a Listbox, as you might expect, the ClickEvent is generated. It's easy to determine the line on which they clicked: use the ItemIndex attribute. But, how do you determine the column? Immediately after a click, Everest sets Sysvar(177) to the column number. The following sample programming code uses help from the Pik() function to display the contents of the grid cell on which a user clicked:

```
getlyn = Listbox(1).ItemIndex: getcol = sysvar(177)
Listbox(1).LookAt = getlyn
contents = pik(getcol+1, Chr(.ColChar) + .Item)
crlf = chr(13) + chr(10)
dummy = mbx("That cell contains" + crlf + contents, 64)
```

8.26 HyperText via the Flextext Object

In addition to its abilities to display text in multiple colors, sizes and fonts, the Flextext object also offers hypertext "hot spot" features. At run time, the appearance of the hot spots is similar to those in the Windows Help system.

There are two types of hot spots you can define: Jump and Popup. A Jump type hot spot is displayed with a solid underline at run time. A Popup type hot spot is displayed with a dashed underline at run time. Via event codes, you define what happens when the user clicks on a Jump or Popup. Typically, Jumps move to a new location (such as a different page), and Popups display a message (such as a definition) within a window.

Jumps

Jumps and Popups are designated in much the same way as the Flextext's colors and styles. To specify a Jump, surround the word with \J and \j. For example:

```
For visitor information, click on \JScotland\j.
```

Actually, it's a bit more complicated that than. You must also specify the event (key) code to generate when the user clicks on the Jump. To do so, include a numeric value surrounded by \K and \k. For example:

```
Click on \J\K-123\kScotland\j.
```

In the example above, when the user clicks on the Jump hot spot "Scotland", Everest generates event -123. Trap such event codes with a Wait object, just as you do any event code. If you prefer, you can use a BRANCH command instead of an event code. For example:

```
Click on \J\KBRANCH scotmenu\kScotland\j.
```

Use this technique with caution because such BRANCH commands are NOT checked by Everest's Branch Test utility.

Popups

Popups are designated similarly: instead of \J and \j, you use \P and \p. For example:

```
The \P\K-321\krectifier\p is an integral component.
```

A-pex3 calculations and commands are allowed in place of event codes. In the case of the previous example, you can use this ability to display a message box that contains a description of a rectifier. For example, if you previously stored a textual description into the variable named "rectdesc" you would use a calculation such as:

```
\P\Kdummyvar=mbx(rectdesc)\krectifier\p
```

For details regarding the use of multiple colors and fonts in the Flextext object, consult the technical reference's description of the Text attribute.

Assistance

Constructing these Jumps and Popups can sometimes be tedious. To have Everest build them for you, highlight the desired word or phrase, then choose one of the Flextext items listed on the main Author window's Edit pull-down menu.

8.27 Usage Flags

Many authors like to display "done markers" or usage flags in their projects, often next to the items on a menu page. These markers let users know which sections of your project they have completed. There are many ways to accomplish this. What follows below is a discussion of three possible methods.

To illustrate by example, let's assume you have a page that acts as the main menu for your project. From this main menu, the user can choose any of five different sections. The page contains a Button object for each section. You have designed it so that when the user clicks a Button, the menu page branches to the appropriate section.

Easy, No Frills

Here are the steps for easy, no frills usage flags.

1) On the menu page, place a small Textbox object next to *each* Button.
2) In the Textbox you want to associate with section 1, enter

   ```
   {section1}
   ```

   This is known a embedding a variable. In this example, section1 is the name of the variable being embedded. At run time, Everest will display the value of the variable.
3) Similarly, in the Textbox for section 2, enter

   ```
   {section2}
   ```

4) Repeat for each Textbox, using variables section3 to section5 (or however many sections you have). Save the menu page when done.
5) Next, open the first page of section 1. Add a Program object to it; somewhere near the top, perhaps after the Layout, will do.
6) Into this Program object, enter the following programming:

   ```
   IF section1 #E# "done" THEN section1 = "started"
   ```

   which tells Everest to store the word "started" into the section1 variable, but only if the variable does not already contain the word done.
7) Save the page. Next, open the last page of section 1, add a Program object to it, and enter the following:

   ```
   section1 = "done"
   ```

8) Save the changes. Repeat this process for the first and last page of each section. Just be sure to use variable section2 for the second section, section3 for the third, etc.

Fancier (With Graphics)

If you want something flashier than the Textbox approach above, try the following.

1) Create a graphics image to display as a done flag. Perhaps you already have suitable clip art, such as a fancy check mark, or something similar.
2) On the menu page, place a Picture object next to *each* Button.
3) For *each* Picture, set the PictureFile attribute to the name of the image file from step 1. This will load and display the image.
4) For the Picture you want to associate with section 1, edit the Condition attribute to read

   ```
   IF section1>0
   ```

   omit the "THEN" normally found with IF statements. This tells Everest to, at run time, display the Picture only if the section1 variable has a value greater than 0.
5) Similarly for the second Picture, set Condition to

   ```
   IF section2>0
   ```

6) Repeat for each Picture, using variables section3 to section5 (or however many sections you have). Save the menu page when done.

7) Next, open the last page of section 1. Add a Program object to it; somewhere near the top, perhaps after the Layout, will do.

8) Into this Program object, enter the following programming:

```
section1 = 1
```

9) Save the changes. Repeat this process for the final page of each section. Just be sure to use variable section2 for the second section, section3 for the third, etc.

Ultimate (With More Graphics)

You may have noticed the prior method only displayed a "done" image, and did not display a "started" image. You could modify the method to include two Picture objects for each section, and design it to display the appropriate one. However, Picture objects consume a fair amount of Windows resources, even when empty. If possible, it is best to avoid using many Pictures at once.

The following technique is a bit more complex to create, but it can display either of two images while still using only one Picture object per section.

1) As in the prior method, create a graphics image to act as the done flag. For example, you might name the file DONE.BMP.

2) Also create an image to display as the started flag. You might name the file STARTED.BMP.

3) On the menu page, place one Picture object next to *each* Button.

4) For the Picture you want to associate with section 1, edit the Condition attribute to read

```
IF section1>0
```

omit the "THEN" normally found with IF statements. This tells Everest to, at run time, display the Picture only if the section1 variable has a value greater than 0.

5) While still working with the Picture you want to associate with section 1, set the PictureFile attribute to the following expression

```
{pik(section1+1, ";;started.bmp;done.bmp")}
```

Yes, the list of file names starts with two consecutive semicolons. At run time, if the value of the section1 variable is 1, the Pik() function will return STARTED.BMP; and if section1 is 2, Pik() will return DONE.BMP. The +1 and double semicolons are to avoid passing the Pik() function a value of 0, which it makes it count the number of names in the list (something you don't want to do here).

6) Set the Condition and PictureFile attributes similarly for the second Picture, except use the section2 variable.

7) Repeat for each Picture, using variables section3 to section5 (or however many sections you have). Save the menu page when done.

8) Next, open the first page of section 1. Add a Program object to it; somewhere near the top, perhaps after the Layout, will do.

9) Into this Program object, enter the following programming:

```
IF section1 = 0 THEN section1 = 1
```

which tells Everest to set the section1 variable to 1 (which indicates "started"), but only if the variable is still 0 (which means the user has not previously started or completed the section).

10) Next, open the last page of section 1. Add a Program object to it; somewhere near the top, perhaps after the Layout, will do.

11) Into this Program object, enter the following programming:

```
section1 = 2
```

12) Save the changes. Repeat this process for the first and last page of each section. Just be sure to use variable section2 for the second section, section3 for the third, etc.

9 Multimedia and External Devices

Multimedia means the combination of text, graphics, animation, audio and/or motion video. While the word multimedia was coined relatively recently, it describes something our authoring systems have offered ever since the first one was released in 1986.

This chapter also briefly discusses communication with external programs via Windows DDE and OLE technologies.

9.1 Comparing the Media Object and Mci() Function

Just as with many things in Everest, there is more than one way to incorporate multimedia in your project.

The Media Object

The Media object provides a high-level, no programming approach. Basically, you drop a Media object onto your page, then set the DeviceType attribute. The Media object can play .WAV audio, CD audio, videodiscs, MIDI music files, and Microsoft Video for Windows .AVI files, among others.

The Mci() Function

For ultimate low-level control of multimedia devices, A-pex3 programmers can employ Everest's Mci() function (MCI stands for Media Control Interface, a Windows standard). Virtually all multimedia devices respond to commands sent to them via the MCI. The commands vary based on the device and manufacturer, but they usually resemble phrases such as "load file" or "play from 1 to 5" etc.

Obtaining Drivers

For both the Media object and Mci() function to address various multimedia devices, you generally need software called a "driver" to be previously installed into Windows via the Windows Control Panel's Setup utility. These drivers are not packaged with Everest; contact the device's hardware manufacturer to obtain them.

9.2 Using the Media Object

When you drop the Media object on your page, a row of buttons appears. These buttons are designed to allow the user to control the device manually (play, pause, fast forward, etc.). If you do not want the user to see and manipulate these buttons, simply set the ShowButtons attribute to No.

Choose DeviceType First

Before setting other attributes, set the DeviceType. The DeviceType attribute tells Everest whether you are using a CD, videodisc, etc. It needs to know this up front in order to correctly handle the other attributes.

Set the Command

Next, set the Command attribute. The most common setting is the word Play. If you leave Command empty, Everest simply opens the DeviceType, and does nothing more (i.e. you won't see or hear playback of your media element).

Choose the File

In the FileName attribute, enter the name of the file that contains the multimedia content. This might be a .WAV file, or .AVI file, or another file depending on how you set the DeviceType attribute. The easiest way to locate the desired file is to double click on FileName to open Everest's Load File window.

Playing Segments

Most of the time you can leave the StartAt and EndAt attributes empty. However, if you only want to play a segment of a media element, for example, the first 10 seconds of the third track on an audio CD, you do so via StartAt and EndAt.

You can manually type in values for StartAt and EndAt, but usually it is easier to let Everest generate them for you. When you double click on StartAt or EndAt, Everest opens the Multimedia Peek window. Within this window you can control the device you previously selected via DeviceType, and visually or audibly choose a starting and ending location.

You can set the form of the data in StartAt and EndAt by controlling the TimeFormat attribute. Via TimeFormat you tell Everest the starting and ending locations and clock times, frame numbers, etc. Not all TimeFormats are available for every DeviceType...you will need to experiment to see which can be used with your device.

Coordinating Multimedia

The Media object makes it quite easy to coordinate various multimedia elements, for example, start an animation after the first second of a .WAV audio file has played. To do so, you would set UpdateInterval to the time period to wait (in milliseconds), and UpdateEvent to the event code you wish to generate after that interval has elapsed. Use that event to trigger the start of another element.

DoneEvent

Similarly, you may want to perform an action when the multimedia element has finished playing. For example, you might want to replay a sound file, or branch to the next page. This can be accomplished via the DoneEvent attribute.

9.3 Viewing Video Via the Media Object

For multimedia DeviceTypes such as AVIVideo and MMMovie, you need to tell Everest where to display the images. To do so, put an object (such as a Picture) on the page that can act as a container for the images. The class of the object does not matter, as long as it has a visible component in the VisualPage editor.

Also important: put the container object before the Media object in the Book Editor. If you put it after the Media object, the container will not yet exist when the Media object attempts to display the images. Recall that you can reorder an object in the Book Editor by dragging it while holding down the right side mouse button.

Here are step-by-step instructions on how to play an .AVI motion video file (Microsoft Video for Windows file) in your project:

1) Put an object into the page that can act as a container for the .AVI; a Picture object works well.
2) Put a Media object after the Picture object.
3) Set DeviceType to `AVIVideo`.
4) Set DisplayIn to `picture(1).` The "1" is the IDNumber of the container object; your value might be different.
5) Set Command to `Play`.
6) Set FileName to the name of the .AVI file. You can double click on FileName to open the load file dialog box.
7) If you want to resize the video to fit the container, enable AutoScale.
8) Put a Wait object after the Media object.
9) Run a Preview of the page.

Sizing the Image

If you would like to resize the image to match the size and shape of its container, set AutoScale to yes.

9.4 Using the Mci() Function

If you prefer, you can control multimedia devices directly via Everest's Mci() function. The Mci() function sends your commands to the devices via the Windows Media Control Interface. Specifically, Mci() Action 0 calls the Windows API MciExecute function, and Action 1 calls the Windows API MciSendString function.

The following example uses the Mci() function to play the Microsoft Video for Windows COWBOY.AVI file in the current window:

```
ecode = mci(0, "open cowboy.avi alias anyname type AVIVideo")
IF ecode = 0 THEN
  ecode = mci(0, "window anyname handle " + window(0).hwnd)
  size = window(0).width $+ " " $+ window(0).height
  ecode = mci(0, "put anyname destination at 0 0 " + size)
  ecode = mci(0, "play anyname wait")
  ecode = mci(0, "close anyname")
ELSE
  dummyvar = mbx("MCI error " + ecode)
ENDIF
```

The commands (such as open, put, play and close) in the example above are defined by the Microsoft Video for Windows driver. Each device driver supports several standard MCI commands, as well as some unique to the device. Consult the device's documentation to learn what commands are supported. The Microsoft Multimedia Programmer's Guide is also helpful.

9.5 Playing a .WAV File

There are two ways to play .WAV sound files

Media Object

The easiest way to play a .WAV sound file is via the Media object. Drag a Media object into your page, set DeviceType to WaveAudio, Command to Play, and FileName to the name of the .WAV file. Then run a Preview.

Mci() Function

Operation 3 of the Mci() function also lets you play a .WAV sound file. The following example plays the TADA.WAV file (one typically found in the \windows directory):

```
ecode = mci(3, "c:\windows\tada.wav")
```

If you do not know in advance the name of the subdirectory that contains Windows, use Everest's &: disk drive wildcard; for example:

```
ecode = mci(3, "&:tada.wav")
```

The &: tells Everest to automatically determine the location of Windows at run time. A complete list of drive wildcards can be found in the technical reference/on-line help for the Pth() function.

If an error occurs during play back, it is returned as a non-zero number. You can use Action 2 to retrieve a description of the error. For example:

```
ecode = mci(3, "c:\windows\tada.wav")
IF ecode # 0 THEN
  mess = mci(2, ecode)
  dummyvar = mbx(mess, 48)
```

```
ENDIF
```

9.6 Using the Media Object and Mci() Together

For special purposes, you might want to use the Mci() function to operate a device that was opened by the Media object. To specify the device name (required by most MCI commands) use the Media object's FileName.

You cannot use the Media object to control a device that was opened via the Mci() function.

9.7 Checking Device Availability

As previously mentioned, most multimedia devices need a driver loaded into Windows in order to operate. Everest, through Windows, communicates with the drivers, rather than with the devices directly. If your project employs multimedia, but the user's computer does not have the corresponding drivers installed, the user will likely receive an error message, or no multimedia element at all.

Checking for Sound Availability

To check if the computer is capable of playing .WAV files, employ the Ext(114) function. It returns the number of sound devices installed in the computer. You might use A-pex3 programming similar to the following at the start of your project:

```
IF ext(114) = 0 THEN
  dummy = mbx("Your computer cannot play audio.", 16)
  BRANCH @exit
ENDIF
```

Checking for Other Devices

Fortunately, you can check for the presence of most drivers by examining the Windows WIN.INI file. When a driver is installed, it adds a line (its "signature") in the WIN.INI file. You can read (and write) INI file contents via the Ini() function. The example for Ini() in the technical reference shows how to determine if the Microsoft Video for Windows drivers are loaded.

To determine the signature of a particular driver, contact the manufacturer of the multimedia device.

9.8 Using the HyperHlp Object

As with multimedia, there are two ways you can link to a hypertext help system from Everest: via the HyperHlp object, or via the Hlp() function. The HyperHlp object is handy when you do not want to program, while the Hlp() function is there for A-pex3 programmers.

Both the object and function make calls to the Windows help system for the purposes of loading and displaying .HLP files. You can create your own .HLP files with the Microsoft

Help Compiler for Windows. A complete discussion of how to do so is beyond the scope of this manual; consult a Windows Help guide.

To offer hypertext without use of the Windows Help system, consider the Flextext object (described in the previous chapter).

9.9 Running External Applications

For special purposes you might want to run other applications from within your project. This can be accomplished with the Shl() function. You specify the name of the program to run, and Everest invokes it as if the user had done so manually. Your project continues to run while the other program does.

Programmers often use the Shl() function to start an application in preparation for establishing a DDE or OLE link with it. Consult the technical reference for the Shl() function's syntax.

9.10 Using Windows DDE

DDE stands for Dynamic Data Exchange, and is a method for exchanging information between two running Windows programs. The DDE features in Everest are intended for experienced DDE programmers. The features are accessed via the Dde() function.

An example of DDE use would be a training application for Microsoft Excel. Via Everest, you could instruct the user how to perform a certain operation in Excel, let him try it, then check the result by requesting the value of a spreadsheet cell via DDE. Basically, your project establishes a link with Excel, then asks it to send the cell's contents. The Dde() function returns the contents, and your project can store them in a variable, check them for accuracy, etc.

In the example above, Excel acts as server in the DDE conversation, and Everest as the destination. Everest can also act as server for another application. For example, if you have written a database application in Visual Basic (VB) that needs to retrieve exam scores from Everest, you can design the VB program to request a link with Everest (the technical reference tells you how). Once the link has been established, the VB program can request the value in a Everest variable...it does so by sending the name of the variable surrounded by { }.

In fact, that same program can send your Everest project A-pex3 commands, such as BRANCH or BOX. In theory, the other program could totally control the presentation of your project by sending the appropriate commands.

9.11 Using the OLE Object

OLE stands for Object Linking and Embedding. You can think of OLE as a way to run another application inside a window within your Everest project. OLE is typically employed to let the user see (and possibly edit) the data as presented by another application. Compare this with DDE which is typically employed to access the data in another application.

The OLE features in Everest are intended for experienced OLE programmers. A complete discussion of OLE would probably require several chapters, and is beyond the scope of this book. Consult an OLE technical guide, or the Microsoft Visual Basic Programmers Guide for details.

As of this writing, Everest supports OLE 1 and cannot act as an OLE server.

9.12 Reading Text Files

Via the Fyl() function, your project can read the contents of a text file (ASCII sequential with variable length records, as are most text files). The following example uses Fyl() to read a file named CONTENTS.TXT and count the number of times the word "Everest" is found:

```
ecode = fyl(1, 1, "contents.txt")   $$ open the file
found = 0                         $$ init number found
IF ecode = -1 THEN                $$ -1 means no error
  DO
    IF fyl(11, 1) * "Everest" > 0 THEN found++
    IF sysvar(1) # -1 THEN ecode = sysvar(1): OUTLOOP
  LOOP IF fyl(-11, 1) = 0         $$ = 0 when file has more
ENDIF
dummyvar = fyl(0, 1)              $$ close the file
IF ecode # -1 THEN
  dummyvar = mbx("File error: " + ecode)
ELSE
  dummyvar = mbx("Found " + found + " occurrences.")
ENDIF
```

Read Whole File

If you'd like to quickly read in a whole text file (up to 32K) from disk in one step, rather than line by line as in the example above, use Fyl() function 10. The following example loads the C:\AUTOEXEC.BAT file into the Textbox with IDNumber 1:

```
Textbox(1).Text = fyl(10, 1, "C:\autoexec.bat")
```

Read Whole Embedded File

Text files (or any type of file, actually) can be embedded within the book via the Embedded File Manager. Such files are stored in a compressed format (but do occupy at least 512 bytes of space in the book). To read such a file, use Fyl() function -10, as in the following example:

```
Textbox(1).Text = fyl(-10, 1, "myinfo.txt")
```

9.13 Writing Text Files

Generally, to write a variable record length text file to disk, you must either start writing at the beginning of the file, or append to the end. Except for special purposes, it is not practical to write text into the middle of the file because DOS does not make room for it (you would overwrite old text and end-of-line markers). However, with either fixed record length files (known as random access files) or linked list files, you can write anywhere in the file; such files are discussed in later topics in this chapter.

The following example writes to disk user responses which have previously been stored in the array named resp:

```
ecode = fyl(2, 1, "response.txt")   $$ open for output
cnt = 1                   $$ initialize counter
DO IF ecode = -1          $$ if no error
  ecode = fyl(22, 1, resp(cnt))     $$ write the data
  cnt++                   $$ increment element pointer
LOOP if cnt <= arr("resp")    $$ for each array element
dummyvar = fyl(0, 1)      $$ close file
IF ecode # -1 THEN dummyvar = mbx("File error " + ecode)
```

9.14 Using Random Access Files

In random access files, all records have the same length (number of characters) which you define. The following example reads in the data in a random access file named NAMES.DAT with record length 80, sorts the data alphabetically, then writes it back to disk:

```
ecode = fyl(4, 1, "names.dat", 80)        $$ open file
IF ecode = -1 THEN                        $$ if no error
  maxrec = fyl(-2, 0, "names.dat") \ 80   $$ # of records
  REDIM holder(maxrec)                    $$ allocate
  cnt = 1                                 $$ init counter
  DO IF cnt <= maxrec                     $$ each record
    holder(cnt) = fyl(14, 1, cnt)         $$ load array
    cnt++                                 $$ point next
  LOOP IF sysvar(1) = -1                  $$ if no error
  IF sysvar(1) # -1 THEN ecode = sysvar(1): GOTO uhoh
  dummyvar = srt(holder(cnt))             $$ sort
  cnt = 1                                 $$ init counter
  DO IF cnt <= maxrec                     $$ each record
    ecode = fyl(24, 1, holder(cnt), cnt)  $$ write record
    cnt++                                 $$ inc counter
  LOOP IF ecode = -1                      $$ if no error
  DELVAR holder                           $$ release mem ENDIF
LABEL uhoh
dummyvar = fyl(0, 1)                      $$ close file
IF ecode # -1 THEN dummyvar = mbx("File error " + ecode)
```

9.15 Using Linked List Files

Linked list files combine the convenience of variable length records of sequential text files with the ease of random access files. Everest's .ESL book files and .EUR user records files employ a linked list storage format. In fact, the Fyl() function references the same routines Everest does internally for reading/writing .ESL and .EUR files.

In a linked list file, every record must be assigned a unique name from 3 to 504 characters in length. Records are retrieved from the file by this name.

The following example writes the contents of the variable named moduleflags into a linked list file named COMPLETE.LL located in the same directory as the Everest user records file. Records are identified by the unique user number found in Sysvar(130). The example also demonstrates how to allow for multi-user access on a network:

```
filename = pth(3, sysvar(56))       $$ user rec dir
filename = filename + "complete.ll" $$ append file name

LABEL hourglass
tries = 0                           $$ init try counter
dummyvar = mse(0, 11)               $$ hourglass cursor
LABEL retry
ecode = fyl(6, 1, filename, 5)      $$ attempt to open
IF ecode = -1 THEN                  $$ if no error
  ecode = fyl(26, 1, moduleflags, sysvar(130))
ELSEIF ecode = 70 & tries < 10 THEN $$ 70 = file locked
  now = tim("")
  DO IF now = tim("")               $$ wait one second
  LOOP
  tries++                           $$ count tries
  GOTO retry                        $$ try again
ELSEIF ecode = 70 THEN              $$ 10 tries already
  dummyvar = mse(0, 0)              $$ restore cursor
  mess = "Network is busy.  Do you want to keep trying?"
  choice = mbx(mess, 36)            $$ keep trying?
  IF choice = 6 THEN GOTO hourglass $$ yes
  ecode = -185                      $$ user canceled
ENDIF
dummyvar = mse(0, 0)                $$ restore cursor
IF ecode # -1 THEN dummyvar = mbx("File error " + ecode)
```

10 Record Keeping

10.1 User Records and CMI Data

Both user records and CMI Data are means by which to collect information about the use of your projects. Everest allows you to create a wide variety of projects, some of which typically need record keeping features, and others which do not. Here are a few examples of projects that often do not need user records or CMI:

demonstration disks
slide show presentations
kiosks

Here are a few examples of projects that typically do need user records:

computer-based training
question exam banks
any project of long duration that will be completed in steps

User Records

In Everest, the term "user records" refers to information about the user's place in the project. This is often called a bookmark. Via the user records system, a user can exit your project, and later resume where he left off. A good real world analogy is a photograph; when the user logs off, Everest makes a photograph that's available for recall later.

Most authoring systems lack a built-in bookmark feature. Such a feature is essential if you want users to be able to stop anywhere in your project, then resume later where they left off.

CMI Data

Continuing the analogy, the CMI data collection system is like a videotape. As a user proceeds through your project, the CMI system remembers his/her path, responses to questions, the amount of time spent, etc. CMI is good when you need very detailed information about the use of your projects. You enable CMI via the CMIData attribute of interaction objects.

Both user records and CMI are optional...your project can employ none, both, or just user records. Your project cannot employ CMI data collection without also employing user records. Later topics in this chapter tell you how to enable/disable these options.

10.2 Disengaging User Records

By default, user records are enabled. To try it yourself, simply run your project with the ERUN program, and Everest will prompt you for log on information. It will record where you log off, and allow you to resume there the next time you log on.

Certain types of projects do not need user records. In such projects, you simply want the user to be able to start running without logging on.

Disengaging User Records

Via EVEREST.INI

To disable user records (and the log on process), add an item to the EVEREST.INI that resembles

```
Logon=0
```

which should appear within the [Settings] section.

When user records are disabled, the ERUN program always begins running your project at the page named @start. An error occurs if the book specified via the ProjectFile item of the EVEREST.INI does not contain an @start page.

Detecting at Run Time

At run time, your project can determine whether user records are enabled by examining Sysvar(130). When Sysvar(130) is greater than 0, user records are enabled. The value in Sysvar(130) is a unique number associated with the user's record.

If you want to prevent your project from being viewed without user records, use programming similar to the following in your @start page:

```
IF sysvar(130) <= 0 THEN
  d = mbx("Sorry, you must first log on.", 16)
  BRANCH @exit
ENDIF
```

10.3 Locating User Records

By default, Everest stores user records in a file named USERREC.EUR in the same directory as the project (defined via the ProjectPath entry in the EVEREST.INI file). Everest creates the file if it does not yet exist.

The EVEREST.INI file can be edited with the INSTRUCT program.

If you want to store the user records in a different directory and/or file name, do so by setting the UserRecs entry in the EVEREST.INI file to the desired directory and file name (the file name extension must be .EUR). At run time, Everest stores this custom name in the Sysvar(56) variable for reference. Do not change the value of Sysvar(56) via A-pex3 programming.

Upon user log off, Everest also saves images from the Pic, PicChecked, PicDown, PicGrayed, PicPressed, PicUnChecked and PicUp object attributes in individual files with names that resemble 1B011.SP0 and 11.EPW.

10.4 Customizing User Log On: Level 1

There are two levels at which you can customize user log on. Level 1 involves simple text editing in the EVEREST.MSG file. Level 2, intended for experienced authors, allows you to customize the whole log on process via A-pex3 programming.

At Level 1 you can customize the text you see in Everest's built-in default log on and password windows. This is useful for small changes or for language translation. Start by using a text editor, such as the Windows Notepad or DOS EDIT program, and load the EVEREST.MSG file.

The EVEREST.MSG file contains many messages, each on its own line, and each prefixed with a number. Those messages with numbers greater than 0 are messages produced by Microsoft's software. We have included them in the EVEREST.MSG so you can customize them for non-English language users, if desired.

Messages with numbers less than 0 are unique to Everest. Those between -002 and -050 are used during the log on sequence. You can change the wording as you see fit. Here is description of each message:

Number	**Purpose**
-035	if it exists, it is displayed if ERUN cannot locate the EVEREST.INI file, and lets the user enter a location (disk path or URL) at which to retry
-034	if it exists, it is displayed as the caption of the "are you sure you want to log off" window
-033	displayed when the user closes the last open window (implied log off) to ask for verification of the action
-032	displayed when the user activates the QuitActivator to ask for verification of the action
-031	prompts user to enter a comment
-030	displayed when user records are temporarily locked by another station on a network; asks if user wants to continue to wait; asks every 10 seconds
-024	displayed when user logs on after the first time; asks if user want to resume where they logged off last time
-023	tells user they must enter at least a last name in order to log on
-022	displayed when the user's name is not in the user records file, and the author does not want Everest to automatically create a new record for the user
-021	displayed when the user records file is full
-020	displayed when the user's name is not in the user records file, and the author wants Everest to automatically create a new record for the user
-018	displayed when the user asks to change his password
-017	displayed on the button that allows a user to change his password
-016	displayed as the caption of the Password window

-015	displayed when the user enters an incorrect password for the 4th consecutive time (exhausted tries)
-014	displayed when the user enters an incorrect password up to 3 consecutive times (try again)
-013	displayed when, during new password verification, the user enters two different passwords
-012	prompts the user to re-enter the new password for verification
-011	asks the user to enter a new password
-010	asks the user to enter their password
-009	displayed as the caption of the Log on window
-008	displayed on the Log on & Password window's Cancel button
-007	displayed on the Log on & Password window's OK button
-006	prompts for the 4th Log on input field (ID#)
-005	prompts for the 3rd Log on input field (Group)
-004	prompts for the 2nd Log on input field (Last name)...the user must enter something here in order to log on
-003	prompts for the 1st Log on input field (First name)
-002	displays brief instructions near the top of the Log on window

As mentioned, you can change the wording of the text. However, by removing the text of certain messages (but leaving the number), you can also customize some operational aspects of the log on. Here is a list of what deleting the text does:

Number	Effect of Removing Text (But Leaving Number)
-003	does not prompt for user's first name
-005	does not prompt for group
-006	does not prompt for ID#
-011	does not prompt for password (user needs no password)
-020	Everest will not automatically create a record for a new user; an instructor will have to create a record manually via the INSTRUCT program; message number -022 is displayed instead
-024	user is forced to restart project at @start page upon next log on
-031	disables @comment user comment feature globally
-032	does not ask for verification of quit
-033	does not ask for verification of quit when the user manually closes the last open window
-035	does not allow the user to manually enter the location of the project he wants to run

10.5 Customizing User Log On: Level 2

With Level 2 customization, you can completely change the appearance of the user log on window. You do so by adding a page named `@logon` to the book. When Everest sees a page named @logon in the first book (i.e. the one specified via the ProjectFile item of the EVEREST.INI), it executes it instead of the default log on window.

Creating the @logon Page

Create the @logon page as you would any other page in your project (you can include picture boxes to display a logo, etc., buttons, input fields, whatever). Put a JLabel object at the top of the page, and change its name to "redo."

Collect the user's log on information (perhaps via Input objects), and when ready to allow the user to log on, copy that information into the Sysvars indicated below:

Sysvar(131)	user's first name (optional)
Sysvar(132)	user's last name
Sysvar(133)	user's group (optional)
Sysvar(134)	user's ID# (optional)
Sysvar(129)	password (optional...if left empty, Everest performs its default password process)

An A-pex3 program to do this after the user had entered the information would resemble:

```
sysvar(131) = Input(1).Text
sysvar(132) = Input(2).Text
sysvar(133) = Input(3).Text
sysvar(134) = Input(4).Text
```

Then reference the Rec() function; the Rec() function returns a numeric code that indicates the success of the log on attempt. The following A-pex3 programming example illustrates proper use:

```
ecode = rec(7)          $$ load record
IF ecode = 0 THEN       $$ wait! rec does not exist
  txt = "Your record does not yet exist." + chr(13)
  txt = txt + "Do you want to create a new record?"
  choice = mbx(txt, 36, "Log on")
  IF choice = 7 THEN txt = "": JUMP redo
  ecode = rec(8)        $$ initialize record
ENDIF
IF ecode = -2 THEN      $$ an error has occurred
  txt = "Error " + sysvar(1) + " has occurred!"
ELSEIF ecode = -1 THEN  $$ successful logon
  IF sysvar(127) = 0 THEN OPEN @start[1]
  dummyvar = Window(8).Destroy      $$ close logon window
  txt = ""
  $$ check if greater than first log on for user
  $$ and if an @restart page exists
  IF sysvar(135) > 1 & scn("@restart") > 0 THEN
    sysvar(128) = sysvar(18)        $$ save current name
    BRANCH @restart
  ENDIF
ELSEIF ecode = 1 THEN   $$ canceled password entry
  txt = "You canceled entry of your password."
ELSEIF ecode = 2 THEN   $$ exhausted password tries
  txt = "Exhausted tries at password."
ELSEIF ecode = 9 THEN   $$ wrong password in sysvar(129)
  txt = "That password is not correct."
ENDIF
IF len(txt) > 0 THEN
  choice = mbx(txt, 16, "Log on")
  JUMP redo             $$ to JLabel redo Page top
ENDIF
```

Everest runs the @logon page in window number 8. Consequently, if you are employing a custom @logon page, your project must not use window number 8. Note that the example above closes window number 8 (via the Destroy attribute) upon a successful log on. If you neglect to close window 8, the log on window will remain visible even as your project begins running.

Also, upon successful log on, Everest automatically reopens all windows that were open when the user last logged off. If no windows were open at the time, or if this is the first time the user is logging on, the Sysvar(127) variable will equal 0. If no windows are open, the user will have nothing to do! The example above checks for this situation by examining Sysvar(127), and if it is 0, runs the @start page in window number 1.

If your project contains multiple books, only the first book (i.e. the one declared via the ProjectFile item in the EVEREST.INI) needs an @logon page.

Usage with the Internet

If you make your project granular, do not create a separate book for @logon; instead, put @logon in the first book (normally @START.ESL).

Also, to increase start up performance, ERUN does not automatically check for an @logon page if your books are hosted on an Internet/intranet site.   To force ERUN to recognize and use your @logon page, you must set the LogOn item in the EVEREST.INI file to a value of 5.

10.6 Using @restart and @preempt

When a user resumes where they logged off during a prior session, Everest automatically reopens the appropriate windows, restores the objects within each, and reloads the system and author defined variables.   For special purposes (such as a "welcome back" type message) you might wish to intercept resumption.   You can do so by including in the book a page with the special name @restart or @preempt.

@restart

When Everest sees that an @restart page exists, it runs it after reopening windows and reloading variables that existed when the user logged off.

To branch from the @restart page to the page at which the user had logged off, employ the A-pex3 command:

```
BRANCH {sysvar(128)}
```

Before Everest executes an @restart page, it stores into Sysvar(128) the name of the page at which it would have resumed had there been no @restart page.

The @restart page is an optional feature (i.e. creation of an @restart page is not mandatory). Note that @restart is book specific: if your project contains multiple books, put an @restart page in each book in which you want the @restart feature.

@preempt

When Everest sees that an @preempt page exists, it runs it after reloading the user's variables, but before reopening windows that existed when the user logged off.   Authors typically use an @preempt page to offer a custom welcome back message, and ask the user if he wants to resume where left off.   As with @restart, @preempt is book specific: if your project contains multiple books, put an @preempt page in each book in which you want the feature.

When ready to branch from the @preempt page to the page at which the user had logged off, execute the following commands in a Program object in @preempt:

```
ERASE (11)
IF rec(6) = -1 THEN BRANCH @wait
BRANCH @start
```

To continue elsewhere (i.e. to NOT resume where left off), simply branch to the desired page, such as:

```
BRANCH mainmenu
```

@preempt With Custom @logon

If you are using a custom @logon page (see the previous topic), you should modify the @logon programming to allow for @preempt. Also, you should use Rec(107) instead of Rec(7).   The example below illustrates:

```
ecode = rec(107)        $$ load record & skip resume? box
IF ecode = 0 THEN       $$ wait! rec does not exist
  txt = "Your record does not yet exist." + chr(13)
  txt = txt + "Do you want to create a new record?"
  choice = mbx(txt, 36, "Log on")
  IF choice = 7 THEN txt = "": JUMP redo
  ecode = rec(8)        $$ initialize record
ENDIF
IF ecode = -2 THEN      $$ an error has occurred
  txt = "Error " + sysvar(1) + " has occurred!"
ELSEIF ecode = -1 THEN  $$ successful logon
  IF sysvar(135)<=1 @ sysvar(127)=0 @ sysvar(145)=0 THEN
    OPEN @start[1]
  ELSEIF scn("@preempt") > 0 THEN   $$ does @preempt exist?
    OPEN @preempt[{sysvar(144)}]    $$ if so, open it
  ELSEIF sysvar(135) > 1 & scn("@restart") > 0 THEN
    sysvar(128) = sysvar(18)        $$ save current name
    OPEN @restart[{sysvar(144)}]
  ELSE
    OPEN @start[{sysvar(144)}]
  ENDIF
  dummyvar = Window(8).Destroy      $$ close logon window
  txt = ""
ELSEIF ecode = 1 THEN   $$ canceled password entry
  txt = "You canceled entry of your password."
ELSEIF ecode = 2 THEN   $$ exhausted password tries
  txt = "Exhausted tries at password."
ELSEIF ecode = 9 THEN   $$ wrong password in sysvar(129)
  txt = "That password is not correct."
ENDIF
IF len(txt) > 0 THEN
  choice = mbx(txt, 16, "Log on")
  JUMP redo             $$ to JLabel redo IconScript top
ENDIF
```

Several system variables are used in the programming above.   Sysvar(127) reveals which windows were open when the user last logged off, Sysvar(135) is the number of times the user has logged on, Sysvar(144) indicates which window had been active when the user last logged off, and Sysvar(145) says whether the user has chosen to resume where left off.

10.7 Accessing User Records

User records can be accessed with the Everest INSTRUCT administrator program. Use INSTRUCT to:

add new records

delete existing records

reset a user's password

view user records (.EUR files)

print user records

run a custom user records report

generate an Internet access list (.EAL files)

export user records to dBASE

view user comments (.ECM files)

The operation of INSTRUCT is straightforward. Basically, with help from the pull-down menus, you open the desired file, highlight the desired items, then choose the operation to perform.

Creating and Using Custom Reports

Everest lets you, the author, create custom user records reports. To create the reports, build one or more pages with the AUTHOR program and embed the variables you wish to display (for example, Sysvar(132), the user's last name) within the objects of the page. For instructions on how to embed variables, see the topic "Displaying Variables at Run Time" in the A-pex3 Programming chapter.

To print custom reports on the printer, employ the LPRINT command and/or ext(19) in A-pex3 programs within your custom report pages.

What the Administrator Does

In the INSTRUCT program, highlight the user records for which to generate a custom report, select Custom Report from the User Records pull-down menu, and choose the page at which to start. The INSTRUCT program then loads each user's record, and runs the page, performing calculations and branching just as the ERUN program does normally. A report counter can be found in Sysvar(169).

10.8 Enabling CMI Data Collection

As previously described, Everest's CMI data collection system works like a videotape in that it records a user's progress through a project. By default, Everest does NOT collect CMI data. To tell it to do so, you must perform the following steps:

1) In the pages of the project, set the CMIData attribute to Yes for those objects for which you want to automatically save CMI data. Alternatively, use the Rec() function to save CMI data manually via A-pex3 programming.

2) Enable the global SaveCMI flag in the EVEREST.INI file. This can be done via the INSTRUCT program. Edit the EVEREST.INI file and make sure the file contains the item:

   ```
   SaveCMI=1
   ```

3) Run the project via ERUN with user log on. (If you disable log on, Everest cannot collect CMI data.)

Where the Data is Saved

The CMI data is saved in a file named CMIFILE.DAT. Everest puts this file in the same directory as the user records file. You can configure this via the UserRecs entry in the EVEREST.INI.

10.9 Processing CMI Data

To make use of the data stored in the CMIFILE.DAT, employ the SUMCMI program. SUMCMI is a DOS-based dBASE program. It accumulates the data from one or more CMIFILE.DAT files into a master database, then lets you generate several simple reports.

The SUMCMI.EXE program provided with Everest is upwardly compatible with the one provided with Summit for DOS. This means you can mix CMI data collected with both Summit for DOS and Everest.

SUMCMI is not a very fancy program. Its purpose is to illustrate the variety of information that can be collected and reported.

Licensed users of Everest can obtain the source code of SUMCMI at no additional charge by request. The source code is written in dBASE, and can be compiled with Clipper. We offer the source code so you can customize SUMCMI to meet your particular needs.

11 Extensibility

11.1 What is Extensibility?

Extensibility refers to your ability to enhance Everest's functionality via the use of third-party add-on component objects, or simply, external objects. Such add-on components can operate very similarly to those built into Everest. For example, the Textbox object is a component that is shipped with Everest. Perhaps your project needs a different type of Textbox, or a completely different object, such as a spreadsheet. Thousands of such add-on objects already exist in the form of .VBX files, and the newer .OCX/ActiveX.

.VBX files are typically used with Microsoft's Visual Basic 3.0 for Windows product, though an increasing number of other applications can also employ them. There are .VBX files for animation, database access, image handling, spreadsheets, fancy displays, scientific applications, and much more. All can be added to Everest so that they appear in your projects.

This chapter describes how to use external components in your project, as well as how to create your own.

11.2 Using External Objects

To use an external object in your project, you must first load it into Everest. This is very simple, and is done via the Add External item on the Author window's File pull-down menu. The steps will be illustrated by example.

Three sample external objects are included with Everest: Drives, DirList and FileList. They are packaged together in the file named ICFILES.EXE. These objects display lists of disk drives, directories and file names, and allow the user to select from the lists.

Loading External Objects

To load these objects into Everest, select Add External from the Author window's File pull-down menu. Everest's Load File window will display, and show a list of *.EDF files. For each Everest add-on, such as ICFILES.EXE, there is a corresponding .EDF file, in this case ICFILES.EDF. The .EDF extension stands for "external description file"; the file contain a brief description of the objects in the .EXE. If you peek at the .EDF file from the Load File window, Everest displays the description.

You can find ICFILES.EDF in the directory that contains the Everest AUTHOR.EXE file. When you locate it, double click on it to load it. Upon doing so, you will notice that icons for

Drives, DirList and FileList are added to the ToolSet window (you might need to enlarge the ToolSet window to see them).

At this point, you can employ these objects in your project just as you do the built-in objects: just drag and drop them into your pages.

Distributing External Objects

When distributing your project to end users, you should include the necessary external object .EXE files, plus, in turn, the support files they require (if any). Everest's Project Packager utility will automatically copy the necessary files for you. Be sure you are licensed to distribute the external objects, as well as any support files they require.

Impact on EVEREST.INI File

The EVEREST.INI file remembers which external objects are loaded, and automatically reloads them the next time you start the AUTHOR program. After loading the desired external objects, you should update the .INI file (this can be done manually via the Save .INI item on the Author window's File menu).

Be sure that the copy of the EVEREST.INI you provide with your project to end users also contains the references to any external objects. The references appear as Extern#= entries.

11.3 Creating External Objects

You can create your own external objects for use with Everest. To do so, you will need any software development tool that supports Dynamic Data Exchange (DDE). You can use C/C++, Visual Basic, as well as many other development tools and programming languages. The remaining discussion in this chapter will focus on the use of Microsoft's Visual Basic (VB) 3.0. (VB version 4 can also be used.)

If you choose VB, you will be able to tap into the library of third-party .VBX add-on components quite easily. To use a .VBX with Everest, an interface module is required. The ICFILES.EXE sample add-on program provided with Everest is an example of an interface module. If you have a particular .VBX you want to use, we can provide the interface module to you (in fact, we may already have created the one necessary for the .VBX you have in mind). Alternatively, you can create the interface module yourself with VB.

The discussion that follows is fairly technical, and is intended for programmers that have used VB and DDE. Please note that due to the complex nature of .VBX files, we cannot provide technical support for those not created by us.

Examining ICFILES

The best way to create an interface module is to start by examining the one for ICFILES. Inside the Everest \SAMPLES directory, you should find the files ICFILES.BAS, ICFILES.FRM and ICFILES.MAK. Load the ICFILES.MAK into VB.

You will find several procedures in the ICFILES.BAS module. Here is a brief description of each:

copx	Control operations. Gets and sets object properties.
extmgrx	Processes incoming DDE messages from Everest.

FixColorx()	Function that converts hexadecimal color attribute values.
FixDirx$()	Function that assures directory path ends with : or \.
FixDriveVx$()	Same as FixDrivex, but accepts incoming Variant.
FixDrivex$()	Function that handles disk drive letters specified with @, ? or &.
fnCompx$()	Converts incoming variant into a string form.
fnExtx()	Reverse of fnCompx$().
init	Initialization. Object descriptions go here.
mapid%()	Function that translates an ID # into the value used internally.
objmgrx	Object manager. Adds and deletes objects.
parseint	Parses a comma delimited list of numbers into an array.
sendevent	Sends information via DDE to Everest.

When adding your own objects, you will need to modify the init, objmgrx and extmgrx procedures. You may also need to modify the copx procedure.

A Brief Look at How ICFILES Works

When you use Everest's Add External feature to load ICFILES, Everest run the program, then establishes a DDE conversation with it.

1) Initially, Everest sends requests to ICFILES to determine what objects the module contains. Those requests come in the form of execute strings that trigger the ICFILE.FRM's Form_LinkExecute procedure.
2) The Form_LinkExecute procedure sends the incoming message to the extmgrx procedure for processing.
3) The extmgrx procedure dissects the incoming message. The first thing extmgrx does is restore ASCII chr 0 to the proper locations in the message (DDE cannot transmit chr 0, so Everest substitutes a different character).
4) Extmgrx then parses the 36-byte message header. The header contains information that identifies the desired object and the routine (extt.rout) to perform.
5) The first few messages from Everest request the descriptions of objects and attributes (extt.rout = 0 and extt.rout = -1). The extmgrx procedure returns the values of the attpick$ and att$ global variables, as well as arrays that define the objects. You set the values of these variable in the init procedure.
6) The other routines called by extmgrx are used when objects are added to the window (such as when you drag an object's icon from the ToolSet, and drop it on the VisualPage editor), and when the value of attributes change.
7) When an object event occurs, such as a File1_Click, ICFILES sends this information to Everest in the form of a coded message via the sendevent procedure.
8) When Everest stops running, it sends a shutdown message to ICFILES, which then also shuts down.

Creating an Interface Module

To create your own, new interface module, we recommend that you copy ICFILES.FRM and ICFILES.BAS to new files, then edit them with VB. Here are the steps we recommend:

1) Make copies of the ICFILES.FRM and ICFILES.BAS files, and load the newly named copies into VB.

2) Into VB, also load any additional external modules your project requires, such as .VBX files. The .VBX controls must have an hWnd property to work with Everest.

3) Add the desired control(s) to the form (the one that has the Drives, Directory and File list boxes from ICFILES...do not delete these old controls yet).

4) Set the Index property of the new control(s) to 0.

5) Set the Visible and Enabled properties of the new control(s) to False.

6) Set the other control properties to the default values you prefer.

7) For each new control, add an Image control to the Image1 control array. This control holds the icon that will appear in Everest's ToolSet and Book Editor windows. Make the new Image's Height and Width match those of the Images already present. Load the desired icon into the control via its Picture property.

8) For each new control, on paper make a list of the properties you want to be able to edit at design time in Everest. These are the properties (object attributes) that will appear in Everest's Attributes window.

9) For each property you listed in the previous step, scan the copx procedure to see if it appears. For example. the Top property appears as follows:

```
Case 28756 '"Tp"
   vary = con.Top \ Screen.TwipsPerPixelY
```

Note the two letter code included as a comment on the line with the Case statement. Everest represents each property with a two letter code. The code for the Top property is Tp. Most codes are somewhat mnemonic. Write down the two letter code next to each property on your list.

10) If a property on your list is remarked out in the copx procedure, unremark it and its corresponding Case statement. Note there are two places most properties appear in copx (one sets vary = the property, the other sets the property = vary). Unremark both.

11) If you cannot find the property in copx, you can add it. To do so, invent a unique two character property code that consists of a digit from 1 through 9, and a lower-case letter from a to z. Write the new code on your list. In copx, find the programming that is shown below, and add your new property to the Select Case block:

```
Select Case cviat%
Case cvi("1d")
  vary = con.Drive
Case cvi("1p")
  vary = con.Pattern
Case cvi("2p")
  vary = con.Path
End Select
```

Similarly, add your new property to the Select Case block located further down in the copx procedure. It resembles:

```
Select Case cviat%
Case cvi("1d")
  con.Drive = vary
Case cvi("1p")
  con.Pattern = vary
Case cvi("2p")
  con.Path = vary
End Select
```

Note that the difference between these two Select Case blocks is that the first reads the value of the properties, and the second sets them. For properties that are read-only (and therefore cannot be edited in the Attributes window), you would only modify the first Select Case block shown above.

12) Now move to the init procedure. Near the top of init, you will find a line that resembles:

```
atts$ = "|1d.Drive|1p.Pattern|2p.Path|"
```

Add your new, custom attributes (i.e. those for which you invented a two-character code, if any) to the list assigned to the atts$ variable. Use the format <Code>.<Name>, with each attribute separated by the | character (ASCII 123). The attribute name must be at least 3 letters long and may not start with a digit. Also, be sure the contents of the atts$ variable ends with a | character.

If a custom attribute is a Boolean (has a Yes/No value only) include a question mark after its name.

13) Further down in the init procedure you will find array elements that contain information about the objects in this interface module. Recall that each interface module can hold up to 8 objects. The array elements for the Drives List object of ICFILES resemble:

```
' Drives
  objn$(0) = "Drives"
  oats$(0) = "TpLtWhHtIdCoXXIyFcBcF1TsHeYY1d"
  roats$(0) = "VeMp"
  support$(0) = "icfiles.exe,vbrun300.dll"
  help$(0) = ""
```

Change the values assigned to the array elements to match your object(s). Here is a description of the arrays

objn$()	The name of the object (up to 8 characters).
oats$()	A string of the two-character codes that represent the attributes of the object. The string of codes MUST include the codes Id (ID #), Co (Condition) and Iy (Initially). Case is significant. Even though as of this writing, Everest does not support the SaveAsObject feature for external objects, use the XX code to separate what would be local (instance) attributes from non-instance attributes. If there are attributes whose value you want Everest to save with the object, but should not appear in the Attributes window, put their two-character codes at the end after the special code YY.
roats$()	These are read/write attributes whose value you want to save in the user records between user log off/log on, but which should not otherwise be saved with the object.
support$()	This is a list of files that are required to run your external object (including .VBXs, .EXEs, etc.). Everest's Project Packager utility will copy them to the destination disk. Separate multiple file names with a comma.
help$()	This is the name of the .HLP on-line help file for the object. When an author requests help in Everest, this is the file that will be displayed. Set to a null string if there is no custom help file.

For the first object, store the desired values in element 0; for the second object (if there is one), store the desired values in element 1, etc. Remove any unused settings to these arrays left over from ICFILES.

14) Optionally, you can create attribute descriptions that Everest will display in the Attributes window, and which the author can cycle through by double clicking. Put the descriptions into the attpick$ variable in the format described in the init procedure.

15) Next, in the procedures for each object event you want to transmit to Everest, you must place some VB programming code similar to the following:

```
Call sendevent("He", 0, index, xtra$)
```

The sample shown above is from the ICFILES Drive1_Change event procedure. The first parameter is a two-character code that represents the event. The second parameter is the object number (from 0 to 7). This is the same number you used when setting the array elements that describe this object in the init procedure. Change this number as necessary. The third parameter is the control array index. The fourth parameter is used by certain events to send extra information (see the samples for ICFILES). The quoted two-character string ("He" in the example above) identifies the event. For proper operation while authoring, it is important that for each object you CALL sendevent for each of the events in the list below (if the object has them):

Ce	click event
Dc	double click event
Dd	drag drop event (xtra$ information)
Gf	got focus event
He	change event
Lf	lost focus event

Md	mouse down event (xtra$ information)
Me	mouse over event
Mf	mouse leave event
Mm	mouse move event (xtra$ information)
Mu	mouse up event (xtra$ information)

Certain events transmit extra information (typically X-Y coordinates). The mouse move event also checks that the mouse really has moved (to avoid sending too many messages to Everest); refer to the MouseMove examples in ICFILES.

The simplest thing to do is to copy the event handling code for each event from one of the ICFILES objects, and (if necessary) change the second parameter in the sendevent call to the proper object number.

If there are other events you want to transmit to Everest for run-time processing, invent a unique two-character code consisting of a digit (1 to 9) and character (a to z), and call sendevent. Be sure to describe the two-character code in the list of attributes in the init procedure just as you do for properties.

16) In the objmgrx procedure, for each object, copy and modify the Select Case block shown below:

```
Select Case obj%
Case 0                              ' E, Extern
  If newcon% Then
    localid% = mapid%(1, wto%, obj%, cto%)
    Load icfiles.Drive1(localid%)
    X% = SetParent(icfiles.Drive1(localid%).hWnd, extt.hwn)
    icfiles.Drive1(localid%).Tag = newtag
  End If
  If Action% <= 2 Then                  ' from disk
    Call copx(1, obj%, icfiles.Drive1(localid%), o$, opt&, vwaste)
  Else
    icfiles.Drive1(localid%).Move Screen.TwipsPerPixelX * atx, Screen.TwipsPerPixelY * aty
  End If
  If enable% Then
    icfiles.Drive1(localid%).Enabled = yes
    icfiles.Drive1(localid%).Visible = yes
  End If
```

Change the occurrences of "icfiles.Drive1" in the VB programming code above to the name of your form and control.

17) Further down in the objmgrx procedure, find the following VB programming code:

```
  Select Case obj%
  Case 0
    oldtag$ = icfiles.Drive1(localid%).Tag
    Unload icfiles.Drive1(localid%)
  Case 1
    oldtag$ = icfiles.Dir1(localid%).Tag
    Unload icfiles.Dir1(localid%)
  Case 2
    oldtag$ = icfiles.File1(localid%).Tag
    Unload icfiles.File1(localid%)
  End Select
```

Change the occurrences of "icfiles.Drive1" to the name of your form and control.

18) Similarly, in the following block of VB programming code found in the extmgrx procedure, change the occurrences of "icfiles.ObjectName" to the name of your form and control:

```
Select Case extt.rout                         ' desired routine
Case 2                                        ' call control operations
  localid% = mapid%(2, wind%, obj%, ind%)
  Select Case obj%                            ' diff call needed for each object
  Case 0
    Call copx(op%, obj%, icfiles.Drive1(localid%), buf$, waste&, vary)
  Case 1
    Call copx(op%, obj%, icfiles.Dir1(localid%), buf$, waste&, vary)
  Case 2
    Call copx(op%, obj%, icfiles.File1(localid%), buf$, waste&, vary)
  End Select
  If (op% And 1) Then nochange% = yes         ' no new info to return
```

If your interface module has only one object, you can delete the Case 1 and Case 2 sections in the programming code above.

19) Finally, you can delete the Drives, Directory and File lists from the form.

20) Make an .EXE file from VB. If VB complains about "Property or control not found", comment out the offending code. If VB wants a startup form or Sub Main, use the form.

21) Create an .EDF (external description file) with the Windows Notepad or DOS EDIT program, or any text editor for your interface module. Put the .EDF file in the same directory as your .EXE.

22) Try your module with Everest.

Debugging Your Interface Module

In a perfect world, your interface module will work perfectly the first time. However, in the real world, the results are likely to be different. The interface module can be difficult to debug because 1) the bulk of the programming is foreign, and 2) it tends to be event driven and timing oriented.

The best debugging technique we have found is to scatter MsgBox statements in the code (that display values of critical variables), recompile, and retry. If Everest is running concurrently, be sure to unload the external module, and reload the new one.

Another technique is to comment out the Me.Visible = 0 statement in the Form_Load procedure so that the interface module's form will be visible. Then, add statements that assign values to the Label1 and Label2 controls on the form (their sole purpose is for debugging). You'll be able to watch the values change as Everest communicates with your module.

One more idea: scatter Beep statements in the code to produce audible notification that certain sections of it are executing.

11.4 Calling DLLs

DLL stands for Dynamic Link Library. DLLs typically contain useful pre-programmed routines that your project can employ (for example, routines that operate an external device, such as a videodisc or scientific instrument, or routines that are built into Windows itself).

Your project can call DLL routines via the Everest Dll() function. The Dll() function invokes Everest's master DLL driver program, EDLLDRV.EXE, which performs the actual DLL call and returns information to Everest. If you employ the Dll() function in your project, be sure to include EDLLDRV.EXE among the files you distribute to your end users.

EDLLDRV.EXE is a program created with Microsoft Visual Basic 3.0 for Windows. As of this writing, it contains references to the following routines in the Windows API: GetDriveType, GetKeyboardType, and GetSysColor. See the Dll() entry in the technical reference for the proper syntax for calling these routines.

You can modify the EDLLDRV.EXE file to call other routines. To do so, you will need a copy of Microsoft Visual Basic 3.0 or 4.0 for Windows. Use Visual Basic to edit the EDLLDRV.MAK project. The section of code to modify is in the dllmgr procedure, and resembles the following:

```
Select Case LCase$(p(1))
Case "getdrivetype"
  i% = p(2)
  vary = GetDriveType(i%)
Case "getkeyboardtype"
  i% = p(2)
  vary = GetKeyboardType(i%)
Case "getsyscolor"
  i% = p(2)
  vary = GetSysColor(i%)
Case "**shutdown**"
  shutdown% = yes
Case Else
  vary = "No such DLL routine defined: " & p(1)
End Select
```

If you add calls to new routines, note that you must also declare them (via the Visual Basic DECLARE statement).

Alternatively, we can enhance the EDLLDRV.EXE program for you (so it calls the additional routines you wish) for a nominal fee. It is possible that we have already added the routines for someone else, and all you might need is an updated copy of the EDLLDRV.EXE file. Contact us for details.

11.5 Programming in VB and C/C++

For ultimate flexibility, you can program in Microsoft Visual Basic or C/C++ and call the routines from Everest. Actually, any programming language that supports DDE can be used.

Using Visual Basic

We recommend Visual Basic for your custom programming because the necessary interface module is supplied with Everest. The suggested approach is to add your custom programming to the EDLLDRV.BAS program (as described in the prior topic). You can then invoke your programming via Everest's Dll() function.

Using C/C++ and Other Languages

You can also program in other languages. You will need to write the necessary DDE communication routines to accept incoming execute strings from Everest, and send replies. We recommend that you model your program after the EDLLDRV.BAS program included with Everest.

Help for Non-Programmers

Need a custom feature or routine, but do not have the programming experience necessary to create it? We can help. Please call us for details.

12 Authoring Tips

Don't miss the SAMPLES.ESL and EXAMPLES.ESL books found in the \EVEREST\ SAMPLES directory. Other tips:

12.1 Main Author Window

1) Drag the bottom of the main Author window (the one that has the File pull-down menu) down a bit to reveal the current display coordinates. The coordinates do not become visible until you click inside the VisualPage window, or resize an object there. The numbers represent either 1) the X-Y location of the last mouse click in the VisualPage window, or 2) the Left, Top, Height and Width of the last VisualPage object that was highlighted.
2) While editing, if a window you want (Attributes, Book Editor, etc.) is buried beneath other windows, select its entry in the Windows pull-down menu, or press its hot key. That brings the selected window to the top. If the main Author window is buried, press Ctrl+M to bring it to the top.
3) While test running a project, and the project is waiting for input, you can bring the main Author window to the top by pressing the AuthorHotKey (defined in the EVEREST.INI). This key is Ctrl+M by default.

12.2 ToolSet Tips

1) Adjust the shape of the ToolSet window as desired by dragging its border. Everest automatically shuffles the icons to fit.
2) To customize the initial appearance of objects added to your page via drag-and-drop, simply create a page with the special name @default. Add objects to it, and set their attributes as you prefer. Be sure to save the @default page in your book. From then on, during this session, any new objects you create on other pages will start out life with the attributes you chose for that object class. The next session, to re-enable the custom attributes, simply load the @default page as if you were going to edit it.
3) To change the size of the icons, use the Icon Size feature on the Book Editor window's pull-down menu.

12.3 VisualPage Editor Tips

1) If the Page contains a pull-down menu object, the menu appears in the VisualPage editor window. Select a menu item (just as you would at run time) to learn the event code it generates.
2) Enable a "snap-to" grid via the Settings window (choose Settings from the main File pull-down menu). Example: if you set the grid size to 10 pixels, Everest automatically aligns each object to the nearest 10th pixel after you move it.
3) If your VisualPage editing window is larger than the actual window size for the user, enable a window size guide by entering the width and height of the user window via the Settings window. Everest will then draw a dashed/dotted line in the VisualPage Editor to mark this size.
4) If you are working with overlapping objects (such as Buttons on Frames) while editing, one object may obscure another. To get around this problem, make the offending object invisible. This can be done via the Hide/UnHide feature in the Book Editor's pull-down menu. Everest automatically unhides hidden objects when you Preview the page, or reload it for editing.
5) Select a group of objects by clicking and holding the left mouse button on an open area of the VisualPage editor, and dragging the mouse. A rubber band box will surround the objects, and when you release the button, Everest will highlight the enclosed objects in the Book Editor. If you would like to edit one or more attributes of all these objects at once, simply do so in the Attributes window (press Ctrl+A if it is hidden).
6) To delete an object, drag it back to the Toolset.
7) Nudge (move by one pixel) the currently selected object(s) by pressing Alt+CursorKey.

12.4 Book Editor Tips

1) Change the attributes of a group of objects in one step by highlighting the objects in the Book Editor, then editing the desired attribute(s) in the Attributes window. If the Attributes window is not open or visible, press Ctrl+A to access it.
2) Temporarily disable one or more objects via the Toggle feature in the pull-down menus. During page execution, Everest ignores objects that are toggled off. Everest displays $$ next to the icons of disabled objects. Use Toggle again to re-enable a disabled object.
3) An asterisk (*) next to an icon in the Book Editor indicates you have made a change to that object.
4) Delete an object by dragging it from the Book Editor via the mouse button 2, and dropping it on the Toolset.

12.5 Attribute Editor Tips

1) Control the order in which the attributes are listed via the Alpha Sort item on the pull-down menu named Options.
2) For those attributes that have a limited number of settings, double click on the attribute name to advance it to the next setting.

3) For color attributes, double click on the name to open the Color Dialog box. For FontName and FontSize, double click on the name to open the Font Dialog box.

4) To automatically select the correct event code for a keypress, double click on an xxxActivator or xxxEvent attribute, then press the desired key.

5) To create a BRANCH command, double click on an xxxAction attribute, then double click on the name of the page to which to branch.

6) You can adjust the width of the two columns. Here's how: move the mouse to a location between the words Attribute and Value. When the mouse pointer changes in appearance, hold down the left button and drag in the desired direction.

7) To prevent the attribute cells from auto-scrolling horizontally, avoid moving the mouse while clicking in the Value column. Or, disable horizontal scroll via the option on the pull-down menu named Other.

12.6 Debug Window Tips

1) To speed up project execution, close the Debug window.

2) Experienced authors can fix certain problems (such as improper ZOrder or undimensioned array by using the "Perform A-pex3 command" feature in the Other pull-down menu.

12.7 Programming Tips

1) Everest accesses variables faster than object attributes. If you refer to the same object attribute multiple times (such as inside a loop), it is faster to copy its value into a variable before the start of the loop, then use the variable.

2) Rather than set the Visible, Enabled and ZOrder attributes separately, use the Zev attribute instead (it's faster).

3) Instead of `count=count+1` use `count++`. Similarly, instead of `items=items-1` use `items--`. The ++ and -- operators are much faster. Note: ++ and -- cannot be used with Sysvars or object attributes.

4) Instead of setting Boolean object attributes to Yes or No, set them to -1 and 0. It's more efficient.

5) Everest remembers the name and IDNumber of the last referenced object. So, in a given program, when referring to several attributes of a particular object, omit the object name after the first reference. Doing so produces a faster running program. For example, change:

```
Textbox(which).Visible = 0: Textbox(which).Enabled = 0
```

to

```
Textbox(which).Visible = 0: .Enabled = 0
```

12.8 Technical Support Tips

1) Please be at your computer when you call for technical support. We might ask you to try some procedures in Everest to address your question.
2) If you receive an error message in Everest's Uh-oh window, write down the complete message before you call. These details are very important.
3) We might need to recreate the trouble here. Be ready to describe how to duplicate the problem, starting with an empty, new page. In general, if we can't duplicate the problem, we can't fix it.
4) It is best to describe the problem first, then the situation in which it occurs (rather than vice versa).
5) To duplicate a problem, we may need you to send us a sample page. Mail is fine, but e-mail (intersys@insystem.com) is much faster.
6) If you have a support contract, the technical support voice line telephone number is 410-531-9000. The best times to call are weekdays 9 AM to 12 Noon, and 1 PM to 5 PM, eastern time. If you are not covered by a support contract, you can still obtain support on an On-Demand basis (call for cost details).
7) For the latest information, visit our Web site on the Internet at http://www.insystem.com.

13 Preparing the Project for the User

Your project is nearing completion. You want to prepare a copy of it, with the necessary run time files, for the user. The topics in this chapter tell you how.

13.1 Run a Branch Test

You should always run a branch test on your project before releasing it to the user. A branch test checks branching instructions found in a given book to make sure the pages to which they refer actually do exist.

Everest's branch test feature can be found on the Author window's Utilities pull-down menu.

Note that the branch test feature cannot test branch commands embedded within Flextext objects. Also, the branch test cannot check granular projects (those split into one page per book).

13.2 Print a Hard Copy

For final review and record keeping, consider printing a hard copy of the contents of your project, This can be done via the Print Book feature in the Author window's File pull-down menu.

Spell Checking

Everest currently does not contain a built-in spell check feature, however you can export your book to a text file, and spell check it with your favorite word processor (or other spell checking software).

Exporting Text

To export only the text of your book, first open the desired book, then from the Author window's File pull-down menu, choose Print Book. In the Print Book window, enable the Print Text Only check box.

To output the text into a file, click the Setup button to open the Windows Print Setup window. In the list of specific printers, there should be one named "Generic / Text Only on FILE:."

Choose this printer. If such a printer does not appear in the list, you'll need to configure Windows for it (use the Windows Print Manager to do so).

Finally, click the Print button to begin.

13.3 Package Your Project

At this point, you have checked everything in your project, and now you want to ship it to the end user. What files need to be shipped, and how do you gather all of them together? How does the end user install your project? These and other pertinent questions are answered in this topic.

NOTE: Before you can distribute your projects to others, you must be licensed to do so. The Everest Free-Authoring Software does NOT include such a license. Contact Intersystem Concepts if you need information about obtaining a distribution license.

What to Ship

The files you need to ship to the end user fall into either of two categories: your project, and the Everest run time play back program.

Your Project

At minimum, your project consists of the book (the .ESL file) that contains your pages. If any pages reference external files (i.e. ones not previously embedded into the book), you'll need to ship those files as well. Most projects employ at least a few such files for images, sound, etc. Your project might even be spread across several books.

The Everest Run Time Program

To replay your project on their computer, the end user needs a copy of the Everest ERUN program, plus its support files. A complete list of these files can be found in the next topic. NOTE: You must be licensed by Intersystem Concepts to distribute ERUN; if ERUN.EXE was not supplied by us with your copy of Everest, you are not yet licensed to distribute it.

The Project Packager

You can certainly gather all the necessary files together manually. However, most authors find that Everest's Project Packager makes the task much easier. The Packager can be found on the main Author window's Utilities pull-down menu. The Packager can do the following:

1) Locate all external files referenced by the pages of your book(s), and copy them into one spot.
2) Gather together ERUN plus all the support files it needs.
3) Copy files onto diskettes, and group the files so they fit on the minimum number of diskettes.
4) Compress the run time files and your project into .ZIP form.
5) Split your pages into separate books (for Internet applications). This is called "making a book granular."
6) Configure the most commonly used EVEREST.INI settings.

7) Generate an InstallShield-compatible end-user setup script.

More details about these items can be found below.

Gathering Your Project's Files

The Packager can gather all the files referenced by your project, and copy them into one location. We recommend that you do NOT use the Packager to copy directly to the media you plan to delivery to the end user. Instead, you should tell the Packager to copy everything into what is known as a "staging area." A staging area is nothing more than a convenient location on your hard drive; a brand new subdirectory makes the ideal staging area.

The Staging Area

The staging area is useful because it serves as a testing area, lets you make final manual adjustments to configurations, and provides a master from which to make copies when you are ready.

To create a new staging area, in the Project Packager window, click the Make Dir button, and create a new subdirectory.

Trying It

To run the Packager, at minimum, do the following:

1) In the From column, highlight the book(s) you want to package.

2) Enable the "Copy graphics/external files" check box.

3) In the To column, point to the staging area.

4) Click on OK.

The Packager will scan your book(s) for anything and everything that looks like the name of an external file. It will attempt to copy the file, but if it cannot (perhaps it is unable to find the file), it will prompt you to specify its location. The Packager is very aggressive when scanning your project for file names, and will even find things you entered as comments. If it finds something that is not a valid file, simply skip it when prompted.

When the Packager is done, it tells you how much (how many bytes) it copied. If you look in the staging area (via DOS or the Windows File Manager), you should see the minimum number of files your project needs to run.

Packaging ERUN

To gather together the necessary Everest run time files (ERUN plus support files), do the following:

1) Enable the "Copy Everest run time files" check box.

2) In the To column, point to the staging area.

3) Click OK.

You can perform the packaging of the run time files either separately, or at the same time you are packaging your project's files.

Grouping Files for Diskettes

If you've ever tried to gather a large group of files together, and copy them onto multiple diskettes so they "just fit" you know what a tedious task it can be. You often wind up shuffling the order of the files several times so that you do not waste empty space.

Fortunately, Everest's Project Packager automates this process. To gather files together in diskette sized chunks, do the following:

1) Set the From and To column information as described in the sub-topics above to copy either or both your project and the Everest ERUN files.
2) In the To column, set the "Max chunk size in K bytes" field to the size of your diskettes in kilobytes. For example, if your diskettes are 1.44M in size, you would enter 1440.
3) Click OK.

If you are Packaging to a staging area on your hard disk, Everest will create subdirectories within the staging area that will each contain files up to the chunk size you specified. If a single file is larger than the chunk size, Everest will alert you to the problem. Everest cannot split such a large file across diskettes; you'll need to use the .ZIP feature (described below) or seek other alternatives.

Differently Sized Diskettes

If your project is large enough that it requires several diskettes, you might want to reserve room on the first to hold your installation program, or other important information. To do so, enter a negative number in the "Max chunk size" field.

For example, if you are using 1.44M diskettes, and want to reserve 440K on the first diskette, enter -1000 for the Max chunk size. In this example, Everest will limit what it copies to just 1000K, then will prompt you to enter the size of the next diskette. You can repeat the process as many times as you want. As soon as you enter a positive number, though, Everest assumes you want it to copy the same amount for each subsequent diskette.

Compressing into ZIP Format

To reduce the shipping size of your project, consider storing it in .ZIP format. The .ZIP format is a very popular one. The PkZip product (from PkWare, and downloadable from many Web sites) helps you compress and uncompress .ZIP files.

You do not need PkZip to use Everest's .ZIP features. However, if you compress your project into .ZIP form, the end user will need PkUnZip to uncompress it. You can obtain a license from PkWare to distribute PkUnZip with your project. Visit their Web site at http://www.pkware.com or call 414-354-8699.

The Project Packager can either compress each file into .ZIP form individually, or compress many of the files together into one .ZIP.

Compress Each File into ZIP Format

Enable this check box when you want to produce individual .ZIP files for each of your projects files. Typically, this feature is used only in tandem with the "make granular" feature when you are preparing your project to run via the Internet.

Compress Together into ZIP Format

Enable this check box to make the Packager put multiple files into one (or more) .ZIP files. Use this feature to help minimize the distribution size of your project.

Making a Book Granular

Enable the "Make granular" check box to split the contents of a book into separate books. This helps to streamline the delivery of your project via the Internet. See the Internet and Intranets chapter for details.

Configuring EVEREST.INI

You must supply a copy of the EVEREST.INI file with your project. The EVEREST.INI stores various bits of configuration information about how you want ERUN to run your project. The Project Packager can prepare a copy of the EVEREST.INI for your project, and configure the most commonly used options.

Starting Book

In the Starting Book field, enter the name of the first book (.ESL file). This is where the user will begin running your project. If you are making your project granular for Internet delivery, you should enter @START.ESL in this field.

User Comments File

Enter the name of the file that will store comments entered by the user while running your project. The file name extension must be .ECM.

Enable User Log on and Records

Enable this check box if you want Everest to require to enter identification information prior to the start of your project. You must enable this feature if you want Everest to create a bookmark for the user. The bookmark saves the user's place in your project when he exits, and allows later resumption at the quit point.

User Records File

Enter the name of the file in which to store user records (the bookmarks). Everest can store the bookmarks of up to 32,000 users in one file. Administrators can employ the Everest INSTRUCT program to read and generate reports based on the user records.

Other Items

To configure other items in the EVEREST.INI, you will need to manually edit the copy of the file the Packager puts in the staging area. This can be done with any text editor, or with the INSTRUCT program. A later topic in this chapter describes the configurable items stored in the EVEREST.INI file.

Generating a User Installation Script

Enable the "Generate setup (user install) script) check box if you want the Packager to prepare a file that contains an InstallShield-compatible setup script. The Packager will put the script into a text file with the extension .RUL. You can edit this file with any text editor.

What is InstallShield?

InstallShield (from InstallShield Corporation, formerly Sterling) is a popular tool that helps you build a custom installation program for your project. With InstallShield, you can create a program that automates the installation of your project for the end user. Basically, the end user can run your InstallShield-created SETUP program within Windows, and InstallShield copies it from the media you supplied (diskettes, CD-ROMs, etc.) onto the user's hard drive, and configures Windows appropriately. Visit the InstallShield Web site at http://www.installshield.com or call 847-240-9111.

Other Installers

The software installation and setup area is changing rapidly, especially for Internet-based applications. Please check the README file that comes with Everest for possible enhancements that are too recent to have been included in this document.

How Do I Use the .RUL File?

The Packager creates a .RUL file that you can compile with InstallShield version 2. It should also work with newer versions, but has not been tested (InstallShield has recently released several new versions and permutations of their product).

Limitations

InstallShield 2 compresses files into a proprietary ".Z" format. Everest's Packager does not support this format. Instead, it can create .ZIP format files, which are different, more standard, and more highly compressed.

If you prefer to use the less efficient .Z format, but still want Everest's Packager to gather the files together into diskette size chunks for you, do the following:

1) Prepare the Packager's options normally, except use a smaller "Max chunk size" to adjust for the .Z format's inefficiency. For example, instead of 1440, you might specify 1300.
2) Enable the "Compress together into .ZIP Format" check box.
3) Run the Packager.
4) Exit Everest, and look in the staging area. You should have several .ZIP files, each of which is no larger than the max chunk size you specified in step 1.
5) Use PkUnZip to uncompress one of these files into its constituent files.
6) Use the InstallShield compress utility to compress the constituent files into a .Z file.
7) Repeat for each .ZIP file Everest's Packager created.
8) If you end up with a .Z file that is too large for a single diskette, go back to step 1 and reduce the Max chunk size.

It's a bit tedious, but much easier than trying to arrange your projects files manually to fit on diskettes.

Without InstallShield

It is not required that you employ InstallShield to package your project for distribution. You can, instead, simply distribute the files Everest's Project Packager gathers, and instruct the end user to copy them manually onto their computer.

Displaying a Clickable Windows Icon

After the end user has copied your project's files onto his computer, he can prepare a Program Group and Program Item for it (to tell Windows about your project). For example, this can be done in Windows 3.1 via the following steps:

1) Select New from the Windows' Program Manager's File menu.

2) In the New Program Object window that appears, click on Program Group, then click on OK.

3) In the Program Group Properties window that appears, enter a description (such as "XYZ course" and a group file name, such as "XYZ.GRP"). Click OK when done.

4) Again, select New from the Program Manager's File menu.

5) This time, click on Program Item, then on OK.

6) In the Program Item Properties window that appears, enter a Description (for example "XYZ course"), the Command Line (which must read ERUN, preceded by the path in which it was installed, for example "C:\XYZ\ERUN"), and the Working Directory (which is the directory in which ERUN was installed, for example, "C:\XYZ"). Click OK when done. Upon doing so, Windows should display the ERUN program's icon within the new Program Group's window.

7) Double click on the ERUN icon to begin running your project.

Testing

We recommend that after you package your project, you install it to test that everything is working properly. Try it on several computers. Often your project can be influenced by the default color settings or graphics resolutions on other computers.

Updating

Once an end user has your project installed, you can send updates simply by providing the files that have changed. These might include the .ESL books and any external graphics files. Gathering the appropriate files must be done manually, as Everest has no way to track which files the user already has

13.4 Multiple Projects

If you use the Project Packager described in the prior topic, and copy both your project's files and the Everest ERUN run time player program files at the same time, the Packager copies all of them together. Many authors distribute the files to end users in this fashion.

However, if you will be distributing multiple projects to the same end user, this approach is inefficient. Why? Because each project will have its own copy of ERUN, when a single copy can do the job.

The following information tells you how to make a single copy of ERUN run any number of projects.

Isolating ERUN

The first step is to isolate ERUN and its support files. By isolate, we mean put them into a subdirectory of their own, separate from any project files.

The following files should all go together in one subdirectory:

ERUN.EXE

EVEREST.MSG	
EDLLDRV.EXE	(needed only if using Dll() function)
AAPLAY.DLL	(needed only if displaying .AAS, .FLI or .FLC files)
COMPPLUS.DLL	
DDEML.DLL	(needed only if performing DDE)
EVEREST.DLL	
GVJPEG.DLL	(needed only if displaying .JPG files)
MHRUN200.DLL	
MHRUN300.DLL	
MMSYSTEM.DLL	
OLECLI.DLL	(needed only if using OLE object)
SHELL.DLL	(needed only if using OLE object)
VBRUN300.DLL	
VER.DLL	
CSFORM.VBX	
CSHT.VBX	
DSSOCK.VBX	
GVBOX.VBX	
MCI.VBX	
MHEN200.VBX	(needed only if using the Ply() function)
MHGCHK.VBX	
MHGCMB.VBX	
MHGGAG.VBX	
MHGGRP.VBX	
MHGLBL.VBX	
MHGLBX.VBX	
MHGOPT.VBX	
MHIN200.VBX	
MHTM200.VBX	
MSMASKED.VBX	
MUSCLE.VBX	
OLECLIEN.VBX	
PICCLIP.VBX	
VBPLAY.VBX	

Plus, support files required by snap-on (extensibility) objects, if any (make sure you are licensed to do so).

Your Project's Files

The next step is to place each of your project's files together in different subdirectories.

Also, copy the EVEREST.INI, configured as you wish, in the location of your project.

Linking the Two

The final step is to create a Windows icon that links ERUN with your project. This can be done manually via the Microsoft Windows Program Manager, or via an installation utility, such as InstallShield.

If you view the Program Item Properties for the ERUN icon you created, you will see a Command Line that resembles:

```
C:\EVEREST\ERUN.EXE
```

This varies, of course, based on where ERUN.EXE is located. With such a Command Line setting, ERUN will expect to find EVEREST.INI in the same location. However, your EVEREST.INI is stored with your project, and that's in a different subdirectory.

This is the key: tell ERUN to look elsewhere for the EVEREST.INI. To do so, add /ini=[location] to the Command Line. For example, if you changed the Command Line to resemble:

```
C:\EVEREST\ERUN.EXE /ini=C:\proj1
```

then ERUN would look for EVEREST.INI in the C:\proj1 subdirectory.

Custom .INIs

If you want different .INIs for the same project, simply rename the EVEREST.INI to anything you want (except keep the .INI extension), and include the name on the Command Line. Your Command Line might then resemble:

```
C:\EVEREST\ERUN.EXE /ini=C:\proj1\special.ini
```

13.5 Distributing INSTRUCT

The Everest INSTRUCT program is designed as a tool for administrators at your end user's location. Via INSTRUCT, administrators can modify the user records files, generate reports, view comments, etc.

Many authors do not distribute INSTRUCT with their projects. Others find it an invaluable tools for their end users. If you would like to distribute INSTRUCT, be aware that you must be licensed by Intersystem Concepts to do so.

INSTRUCT Files

The INSTRUCT program needs all of the support files (.VBXs and .DLLs) that ERUN does, plus the following:

INSTRUCT.EXE	(instructor module itself)
MSAJT110.DLL	(INSTRUCT.EXE support file)
VBDB300.DLL	(INSTRUCT.EXE support file)
XBS110.DLL	(INSTRUCT.EXE support file)
MSAES110.DLL	(INSTRUCT.EXE support file)
EURTEMP.DBF	(INSTRUCT.EXE support file)
EUVTEMP.DBF	(INSTRUCT.EXE support file)

SUMCMI

SUMCMI is the CMI database manager. If you have a license to distribute INSTRUCT, at your option, you can distribute SUMCMI. SUMCMI needs the following files.

SUMCMI.EXE	(CMI data reporter)
CMISCRN.DBF	(SUMCMI.EXE support file)
CMISTU.DBF	(SUMCMI.EXE support file)
CMIWORK.DBF	(SUMCMI.EXE support file)

13.6 EVEREST.INI File Details

The EVEREST.INI file contains configuration information for the AUTHOR, ERUN and INSTRUCT programs. The file can be edited with the INSTRUCT program, or with a text editor. Typically, you provide a copy of the EVEREST.INI when you distribute your project.

Unless you tell it otherwise, Everest looks for its .INI file in the same directory as the .EXE being executed. If you want to place EVEREST.INI in a different directory, at the end of command line specify the directory name in the form:

/ini=<inipath>

for example:

```
D:\CBT\ERUN /ini=C:\WINDOWS
```

Contents of EVEREST.INI

The following is a description of the entries in the EVEREST.INI file.

[Version]

Ini=	The version number of the .INI file.
Welcome=	Optional: add this to display a short welcome message within ERUN's start up window while your Internet-based project is initially downloading.

[Settings]

ProjectPath=	The location of the initial book. To use the location in which the currently executing .EXE program is located, set to nothing. Examples: `C:\CBT\` or `http://www.xyz.com/`
ProjectFile=	The file name of the initial book. Example: `BOOK1.ESL`
StarPath=	The location to employ for file paths that use * in place of a disk drive letter. To use the directory in which the currently executing .EXE program is located, set to nothing.
TempPath=	The disk path you want Everest to employ for temporary files. To use the path in the TEMP environment variable, set to nothing.
INetSite=	The location to employ for file paths that use ~ in place of a disk drive letter. Example: `http://www.xyz.com/`
INetTimeOut=	The number of seconds to wait for replies from the Internet/intranet. If this time is exceeded, Everest generates error -317 or -323.
FTPServer=	The name of the FTP server for file uploads. This is the name of the server you want Everest to upload to when you edit on the fly, or use the Rec() function to upload user records files. Example: `xyz.xyz.com`
FTPDirectory=	The name of the directory for FTPServer uploads. Example: `/var/www/xyz/html/`

EquivURL=	This is the URL in http: form of the same server location specified by FTPDirectory. Example: `http://www.xyz.com/html/`
FTPUserName=	The user name to employ for gaining proper access for uploads. Example: `guest`
FTPPassword=	The password for FTP uploads. Example: `opensesame`
CachePath=	The location on the local computer (hard disk or RAM disk) in which to store downloaded files. Example: `D:\cache\`
CacheOption=	Set to 0 if you prefer uploads during edit on the fly be made in actual form, or 1 if you prefer ZIP form.
CacheDate=	Set to the date of the oldest allowable file to reuse from the cache.
MsgPath=	The path to the EVEREST.MSG file. If omitted, defaults to the location of the current .EXE.
UserRecs=	The path to and file name of the user records file on the local computer (must NOT be an Internet URL). To use the ProjectPath and USERREC.EUR filename, set to nothing. Example: `C:\records\user.eur`
Comments=	The path to and file name of the user comments file on the local computer (must NOT be an Internet URL). To use the ProjectPath and USER.ECM, set to nothing.
SaveCMI=	Set to 1 to enable CMI data collection, 0 to disable.
LogOn=	Set to 1 to enable user log on, 0 to disable. Add 2 if you want to maximize the default log on window. Add 4 to force ERUN to check for an @logon page when your book is hosted by an Internet/intranet site. If omitted, LogOn defaults to a value of 1.
Splash=	(Optional). Set to the name of a .BMP file to display as a "splash" image to overlay ERUN's startup/copyright window.
AuthorWindow=	The status of the AUTHOR program's main window.
AuthorHotKey=	The event code of the hot key that brings the AUTHOR program's main window to the top.
EditHotKey=	The event code of the hot key that interrupts the running of a project in AUTHOR to edit the current page on the fly.
ResUpHotKey=	The event code of the hot key that resumes running a project in AUTHOR after it was interrupted to edit a page on the fly. Also saves changes.
IconScale=	The relative size, in per cent, of object icons when displayed in the AUTHOR program (100 = normal).
ToolSetWindow=	The status of the AUTHOR program's ToolSet window.
VisualWindow=	The status of the AUTHOR program's VisualPage editor.
ScriptWindow=	The status of the AUTHOR program's Book Editor window.
AttribWindow=	The status of the AUTHOR program's Attributes window.
AttribDesc=	Set to 1 to display attribute descriptions in the AUTHOR program's Attributes window, or 0 for no descriptions.
ApexWindow=	The status of the AUTHOR program's A-pex3 Program editor window.

MenuWindow=	The status of the AUTHOR program's pull-down menu editor window.
MMWindow=	The status of the AUTHOR program's multimedia finder window.
DebugWindow=	The status of the AUTHOR program's Debug window.
LoadFileWindow=	The status of the AUTHOR program's LoadFile window.
PeekWindow=	The status of the AUTHOR program's Peek window.
SettingsWindow=	The status of the AUTHOR program's Settings window.
ObjMWindow=	The status of the AUTHOR program's Object Manager window.
EmbedWindow=	The status of the AUTHOR program's Embbede File Manager window.
AssistWindow=	The status of the AUTHOR program's A-pex3 Assistant window.
RegNo=	Registration number.
SortAttributes=	Set to 1 to display attributes in alphabetical order in the AUTHOR program's Attributes window, or 0 for functional order.
SkipSyntaxCheck=	Set to 1 to disable automatic syntax checking in the AUTHOR program's A-pex3 Program code editor, or 0 to enable.
AutoSaveINI=	Set to 1 to update the EVEREST.INI file upon exiting the AUTHOR program, or 0 to not do so.

[CustomColors]

These colors appear in the AUTHOR program's Color dialog box.

SaveAsObject=	The background color to employ in the Attributes window while editing an object that has SaveAsObject enabled.
0=	Custom color 0, default = &H0&
1=	Custom color 1, default = &H800000&
2=	Custom color 2, default = &H808000&
3=	Custom color 3, default = &H80&
4=	Custom color 4, default = &H800080&
5=	Custom color 5, default = &H800080&
6=	Custom color 6, default = &H8080&
7=	Custom color 7, default = &HC0C0C0&
8=	Custom color 8, default = &H808080&
9=	Custom color 9, default = &HFF0000&
10=	Custom color 10, default = &HFF00&
11=	Custom color 11, default = &HFFFF00&
12=	Custom color 12, default = &HFF&
13=	Custom color 13, default = &HFF00FF&
14=	Custom color 14, default = &HFFFF&
15=	Custom color 15, default = &HFFFFFF&

[SnaptoGrid]

X=	Horizontal pixels between object placement grid in AUTHOR program's VisualPage editor.
Y=	Vertical pixels between object placement grid in AUTHOR program's VisualPage editor.
On=	0 when grid alignment is off, 1 for TLWH adjustment, 2 for TL adjustment only.

[WindowGuide]

X=	Location of vertical boundary line in AUTHOR program's VisualPage editor, in pixels.
Y=	Location of horizontal boundary line in AUTHOR program's VisualPage editor, in pixels.

[Names]

Author=	The name of the person using the AUTHOR program. This name is saved with pages that are changed.
Extern1=	The full path and file name of an external object driver program (such as ICFILES.EXE). Everest allows up to 16 such programs (Extern2=, Extern3=, ..., Extern16=).
DLLDrv=	The name of the Dll() function driver program. If set to nothing or not present, Everest uses EDLLDRV.EXE located in the same directory as the currently running .EXE program.

Wild Card Disk Paths

For more flexibility in specifying locations for the EVEREST.INI items that accept disk paths, you can employ disk drive wild card characters. These are documented in the technical reference/on-line help for the Pth() function. For example, if you want the user records to be stored on the same drive as the ERUN program, but you do not know in advance which disk drive that will be on the end users computer, you would employ the % wild card in the UserRecs item. This might resemble:

```
UserRecs=%:\records\user.eur
```

Computed Disk Paths

You can also embed expressions for the disk path items. For example, to set CachePath to the location specified via the DOS environment variable named TEMP, you would use:

```
CachePath={env("TEMP")}
```

13.7 Preparing Projects for Compact Discs

A Compact Disc (CD-ROM) is an information storage medium, just as floppy diskettes and hard disks are. Many people ask us if Everest can "make CDs" as if there were something magical about the discs. Everest can make CDs just as well as it can make 5.25" floppy disks, 3.5" floppy disks and hard disks (Everest supplies the tools, and authors supply the content and materials). You can think of a CD-ROM as a big floppy disk...one whose contents cannot be changed once written.

And, that's the only significant fact about a CD-ROM: it is a read-only disc.

Since a CD-ROM cannot be written to (after manufacture) you must be sure your project is fully debugged before disc pressing begins. We recommend you keep your entire project on a hard disk and test it thoroughly.

When ERUN Writes to Disk

There are certain situations in which the ERUN run time program needs to write to disk:

1) to write user records

2) to create a temporary file from one embedded in the book

3) to handle any A-pex3 file writing operations in your project

These situations must be handled with care so that ERUN does not attempt to write to the CD-ROM (which would generate an error message). Suggestions are given below.

Handling User Records

User records are necessary only when your project employs user log on. If you want to disable log on, modify the EVEREST.INI file to include LogOn=0 in the [Settings] section.

If you are employing user records, specify a write-enabled directory and file in which to store the records via the UserRecs entry in the EVEREST.INI file.

Handling Temporary Files

You need to worry about this issue only if you have embedded files into a book (Everest creates a temporary file when it loads such a file). Specify a write-enabled directory in which to store the temporary file via the TempPath entry in the EVEREST.INI file. A RAM disk is ideal.

Alternatively, you can leave TempPath empty, and instruct the end user to set the DOS environment parameter named TEMP to the desired write-enabled directory.

Handling Your A-pex3 Programs

You need to worry about this only if your project writes its own files (such as via the Fyl() function). We recommend that your programs determine a write-enabled directory at run time via system variables (such as Sysvar(149) which can be configured by the end user by modifying the StarPath item in the EVEREST.INI file).

ERUN Locations

You can place ERUN (and its support files) on the CD, or install it on the user's hard drive. The former is easier because no special installation is needed.

ERUN on the CD

If you decide to put ERUN on the CD, we recommend that you place it in the same subdirectory as the starting book of your project. When ERUN starts running, it checks the ProjectPath entry of the EVEREST.INI file to determine where (in which subdirectory) the book is located. You should leave this blank (i.e. "ProjectPath="), to tell ERUN to look for your book in the same subdirectory (i.e. the one in which ERUN is located).

Alternatively, you can place ERUN in the root directory of the CD, and your project in a subdirectory. You can't know in advance the drive letter designation of the end user's CD-ROM drive, so omit it in the ProjectPath entry. Omitting the drive letter (and colon) tells ERUN to use the current drive, For example, if your project is stored in the \WOW subdirectory, put `ProjectPath=\WOW` in the EVEREST.INI.

ERUN on the Hard Disk

If, as part of the installation process for your project, you put ERUN on the end user's hard drive, you should also put the EVEREST.INI file there. Your installation program can adjust the disk path entries in EVEREST.INI to conform to the user's computer system. Alternatively, the end user can employ a text editor to manually modify the EVEREST.INI file as you instruct.

EVEREST.INI Locations

By default, ERUN expects to find the EVEREST.INI file in the same subdirectory (i.e. the one that ERUN is in). If you have several different projects on the CD, you might want to play back each with just one copy of ERUN. This can be done by creating a separate EVEREST.INI file for each project, and telling ERUN where to find it.

To tell ERUN to look for the EVEREST.INI in a certain subdirectory, include the /ini switch in the ERUN command line. For example, in the Windows Program Manager's Program Item Properties window, for Command Line you could enter:

```
C:\WOW\ERUN /ini=D:\MODULE2\
```

This would run the ERUN program from C:\WOW, and tell the program to use the EVEREST.INI found in D:\MODULE2. The EVEREST.INI file can then, in turn, specify the location of the project, user records, etc. Note: the /ini parameter must be the last one on the Command Line.

13.8 Preparing Projects for Networks

Local Area Networks (LANs)

Everest works with NetBIOS compatible networks, and automatically performs the appropriate file sharing and locking for books, user records, user comments and graphics files. This means multiple users on a network can safely run the same project and share the same user records files.

Internet and Intranets

See the chapter devoted to the Internet and intranets.

14 Internet and Intranets

14.1 Internet vs. Intranet

Recent years have witnessed tremendous growth of the Internet and of intranets. Functionally, the Internet and intranets are similar. The main difference is one of scope: the Internet is available world wide and is open to the public, while intranets are typically used by a group, such as a particular corporation, and usually have restricted access.

Communication speeds are also vastly different. As of this writing, most people access the Internet with a so-called "28.8" modem. The number refers to the maximum speed at which data can be transmitted. In the case of "28.8" that means 28,800 bits per second. Since there are 8 bits per byte, 28,800 bits = 3,600 bytes. In real world terms, each character of text fits in 1 byte. So, 3,600 bytes means 3,600 text characters per second. That's slightly higher than the number of characters found on one full page of typewritten text.

Intranet communication speeds are typically much higher, often on the order of megabits per second.

A Page Per Second - What's the Problem?

So, even 28.8 modems can transmit a page per second. Since that's faster that people can read, you might be wondering why the Internet often seems so slow. The answer can be summed up in one word: multimedia.

While text is relatively compact, images, sounds, animation and video are not. A one page document might require only 3 Kbytes to store on your computer. A one page picture will occupy substantially more room, sometimes 1000 Kbytes, or even more. Such huge items take a long time to transmit, and create traffic jams on the entire Internet. Even intranets, which are typically much faster, can struggle under the load of large data transmissions.

The following topics discuss how Everest addresses these problems, and makes possible CBT/interactive multimedia delivery via the Internet. The topics in this chapter apply equally to the Internet and intranets.

14.2 Everest and the Internet

The key to successful delivery of an entire CBT or interactive multimedia project via the Internet is squeezing the most performance out of the slow communications link of typical modems.

Brief History

When we first started developing authoring systems over 10 years ago, the slowest component in the computer was the floppy diskette. Many authors used our DOS-based Summit Authoring System to create interactive projects that ran directly from diskettes. We developed many special techniques for speeding the performance of such projects. Back then, we did not anticipate we would later apply this experience to developing Internet-capable authoring tools.

How Everest Does It

Everest does *not* require that the end user download a copy of your project and then run it from their local computer. Instead, Everest functions much like a Web browser in that it automatically retrieves pieces of your project from the Internet on demand. A user that has Everest's ERUN player program can begin running your project, regardless of its size, within just seconds.

Everest combines three technologies to make this possible, even with slow modems.

Data Compression

Everest compresses every page you create so that it occupies minimal space. Many pages compress to just 2 Kbytes. Graphics bitmaps often compress by a 100:1 ratio. Importantly, this data compression happens transparently a you author. There's no need to perform an extra "data compression" step.

Just-in-Time Compilation

Everest compiles your programs at run time, after they have been transmitted to the user. Compilation means Everest translates your programs into a form more easily and more quickly executed by the user's computer. By compiling after transmission, Everest avoids transmitting the compiled version...this dramatically reduces transmission time, and boosts run time performance by up to 500%.

Predictive Preloading

While the user is viewing a page, Everest predicts what content (text, graphics, animation, sound, video, etc.) the user will need next, and pre-downloads it. When the user proceeds, Everest has already obtained the necessary content, and can display it quickly. Rather than let the communications channel sit idle while the user is busy, this technique makes use of the free bandwidth.

Edit-on-the-Fly

Everest also has something for the author: you can directly modify your Internet-hosted projects in real time. This means that while test running your project on the Internet, if you spot a problem, you can choose "Edit-on-the-fly" from the pull-down menu, make the correction, and save the changes, all in a matter of seconds.

Edit-on-the-fly for the Internet is a big time saver, and Everest is the only authoring system that can do it.

What Are the Limits?

You should not get the false impression that absolutely everything you create with Everest can be played back over the Internet with adequate speed. Large multimedia files, especially motion video and audio, are still large even when compressed, and can be slow to transmit. They may be too large to make Internet delivery of your projects practical at the current time. However, as communication speeds increase in the coming years, this bandwidth limitation should gradually disappear.

How do you determine how well your project runs at various communication speeds? Use Everest's Internet simulator; it is described in a topic below.

14.3 Partial or Full Internet Play Back

With Everest, you can create projects that make no, little or full use of the Internet. The same version of ERUN can play back your project regardless of its storage location: entirely on the user's computer, entirely on an Internet site, or any mix in between.

Partial Internet Use

Perhaps the majority of your project's content will reside on the hard disk of the user's desktop computer, and you want to connect to the Internet only to retrieve a bit of current information. For example, you might want to display the current weather satellite photo, current company news, stock prices, whatever. This is quite easy to do.

Loading Images

To load an image file from an Internet site, simply specify its URL in an attribute that accepts image file names. For example, you might set SPictureFile to:

```
http://www.weather.com/satellite/usa.jpg
```

Loading HTML Files

To display a text file that is stored in HTML format, use a Textbox, and Fyl() function operation 10. For example, inside a Textbox you might enter:

```
{fyl(10, 1, "http://www.xyz.com/news.html", 1)}
```

This will load the contents of the HTML file, and remove the embedded codes from it, leaving plain text.

Loading Plain Text Files

To display a normal (non-HTML formatted) text file, use Fyl() function operation 10. For example, inside a Textbox you might enter:

```
{fyl(10, 1, "http://www.xyz.com/plain.txt")}
```

If the text does not wrap properly, it may be because the Carriage Returns have been stripped from it. Try setting the Extra parameter to 2, for example:

```
{fyl(10, 1, "http://www.xyz.com/plain.txt", 2)}
```

Downloading Other Files

To simply download a file from an Internet site, perhaps for other purposes, such as a database, use Fyl() function operations 41 or 42. Operation 41 waits for the file to be downloaded completely; operation 42 downloads the file in the background. The A-pex3 programming to do so would resemble:

```
ecode = fyl(41, 1, "http://www.xyz.com/prices.dbf")
```

Uploading Files

To upload a file to an Internet site via FTP, use Fyl() function operation 43. Prior to attempting such an upload, you must be sure the various FTP items of the EVEREST.INI file (or Sysvars) are set as desired; see the Settings window topic in the technical reference/on-line help. A-pex3 programming to upload a file would resemble:

```
ecode = fyl(43, 1, "C:\result.htm", "results.html")
```

Full Internet Use

By "full Internet use" we mean that the all (or almost all) of your project is stored on an Internet site. With ERUN, the end user can play back your entire project in real time (i.e. without pre-downloading). The remainder of this chapter discusses how to configure your project to make this possible.

14.4 Internet Simulator

Want to test how well an entire project runs via the Internet or an intranet? Don't have Internet access? Use Everest's Internet Simulator.

Benefits of the Simulator

The toughest part of testing your project for Internet delivery is knowing how well it will perform on the end user's computer. You certainly could try your project on many different computers, and with many different modems. But, Everest's Internet Simulator makes that task unnecessary.

Running the Simulator

You do not need Internet access to use the simulator. You don't even need a modem. To run the simulator, choose it from the main Author window's Run menu. You can choose a simulated modem speed, from 9600 baud, up to the 10 megabit-rate of (currently) fast intranets. Click on the Instructions button if you need help entering items in the fields. Click on OK when ready to begin.

Monitoring Status

While your project is running, you can monitor status of the Net connection (or simulated connection) via the main Author window (the one with the File pull-down menu)...make it a bit taller if you cannot see the status display. You should see messages like: "Sim#1: Attempting to connect for file...". The only thing the simulator does not simulate are delays due to traffic on the Internet.

Boosting Run Time Performance

Computer software can never run too fast. Based on the results from the simulator, if you feel your project needs a speed boost, consider making it "granular." This performance boosting technique, as well as others, are discussed in the following topics in this chapter.

14.5 Maximizing Your Project's Internet Performance

Everest does *not* require that you modify, convert, compile or otherwise change your project in order to make it run from the Internet. The same project files (books, graphics, etc.) you would distribute via diskette or CD-ROM are the ones you can post on an Internet site so that anyone in the world who has the Everest ERUN player program can view them.

Speed Problems?

When a user begins running your project, ERUN's first task is to obtain the book that contains the @start page. The name of this book is stored in the ProjectFile item in the EVEREST.INI file. If this book is large, some time may be required to retrieve it, and users may become impatient, or have a poor first impression. Alternatively, your project might start quickly, but bog down when it loads large multimedia files.

There are two techniques you can use either independently, or together, to improve the situation.

1) Store Your Project's Files in .ZIP Format

If Everest cannot find a file your project needs, such as a picture or video, it automatically searches for the same file name, except with a .ZIP extension. In case you are not already familiar with them, .ZIP files are highly compressed forms of other files. Several programs can compress files into .ZIP form, the most popular being PKZip from PKWare.

For example, perhaps your project uses a file named BIGIMAGE.BMP. At run time, if Everest cannot find the file, it then checks for BIGIMAGE.ZIP. If it finds BIGIMAGE.ZIP, it automatically uncompresses it and re-attempts to load BIGIMAGE.BMP.

You don't need to do anything in your project to enable this "auto-ZIP check" feature. However, you do need to compress each of your individual project files into its own .ZIP file. You can do so manually (tedious), or let Everest's Project Packager (described below) do the job for you.

2) Make the Book Granular

The .ZIP compression technique will not help much with books, because Everest already compresses the data stored within. However, if your book is large, you can virtually eliminate the delays associated with retrieving it by making it granular. Making a book "granular" means dividing it so each page is stored in a book of its own. Everest's Project Packager utility has a feature that will automatically do this for you.

Using the Project Packager

The Project Packager readies your project for distribution via any media (diskette, CD-ROM, Internet, etc.). Its main purpose is to gather together all the files your project needs, and copy them into a "staging area" (a subdirectory) on your local computer's hard disk drive.

.ZIPping and Making Granular

To make a book granular and .ZIP all its files, from the main Author window's Utilities menu, choose Project Packager. Choose the source and destination (it may be best to use the Make Dir button to create a new subdirectory on your hard to act as the destination).

Be sure to enable the Make Granular check box. You should also enable both the "Copy graphics/external files" and "Compress each file into .ZIP format" check boxes. Disable the other check boxes. Finally, click OK and let the Packager do its work.

When it creates a book from a single page, the Packager assigns the book a name based on the first 8 letters in the page name.

Trying the Granular Book

Use the Internet Simulator to test run your project after your have made it granular. As the source location, specify the staging area into which you told the Project Packager to put your book. If your original book contains a page named @start (which every book should have as its first page), tell the simulator to start the test run at the `@start.esl` book and `@start` page.

With the .ZIP and granular improvements, you should observe a significant improvement in the download and execution speed of your book.

Caveats about Granular Books

There are a few things you must be careful about when making a book granular.

@next and @back Branching

Do not use BRANCH @next and BRANCH @back commands in a book you will make granular. The reason is once the book has been divided into individual pages, Everest does not know their original order, and therefore cannot handle such branching commands. The Project Packager checks the Wait objects of your pages, and will warn you if they contain such a command.

GOSUB Commands

The Packager will gather all Program objects that have SaveAsObject enabled, and put them into a book of their own called @GOSUBS.ESL. At run time, Everest knows to automatically check the @GOSUBS.ESL book for any Program object it cannot find within the current page.

Embedded Graphics Files

If your original book contains embedded files, the Project Packager will copy the necessary ones into the appropriate granular books. All will work fine. However, if several pages reference the same embedded file, it is better if you do *not* embed the file in the book. Why? If you embed a file, the Packager must put a copy into each granular book that needs it...that makes the granular books larger (and slower to download). If you leave the file external, the granular books will be smaller, and the graphics file will be downloaded into the cache once: only the first time a page needs it.

Multiple Books

If you make multiple books granular, you should probably keep the granular form of each book in its own subdirectory. This is to avoid conflicts if two (or more) books contain a page of the same name.

Editing

In a granular book, if you edit an object that has SaveAsObject enabled, be aware that Everest does not automatically update that object in any other book. If you are making extensive changes, it is probably better if you modify the original book, then re-generate the granular form when done.

@restart and @preempt

Under normal conditions, you need to put an @restart and/or @preempt page into each book in which you want their features to be active. However, this would impose significant overhead for granular books. Therefore, it is only necessary to put the @restart or @preempt page into the first book (normally that's @START.ESL).

14.6 Putting Your Project on the Internet

OK, with help from the Project Packager, you've put your project in a staging area on your hard disk. You've tested it extensively on your desktop computer. Perhaps you've even tried it via Everest's Internet Simulator. You are ready to post it on the Internet.

Uploading Your Project

Basically, all you need to do is upload your project's files from the staging area to the desired location on the Internet or intranet. Everest does not do this for you; you will need "FTP uploading" software or the equivalent. You use the same approach you would to post HTML pages. Contact your system administrator, or your Internet Service Provider if you need instructions.

Using Multiple Directories

Some authors isolate a project's graphics (or other) files into a subdirectory of their own. For example, you might have designed your project to expect the books (.ESL files) in C:\project and the graphics in C:\project\graphics. In a page, the PictureFile attribute of one of your Picture objects might be set to graphics\cat.pcx. To load that image at run time, Everest knows to look for C:\project\graphics\cat.pcx.

If your project employs multiple directories like this example, then create the same directory structure on your INet site. So, for example, if the books reside in http://www.xyz.com, when Everest sees a request to load graphics\cat.pcx, it will look for http://www.xyz.com/graphics/cat.pcx. (Notice how Everest automatically converted the DOS-style \ directory character to a UNIX-style / directory character.)

There are a few things that require care. In DOS, case in directory names is not significant (i.e. graphics\cat.pcx is the same as GRAPHICS\cat.pcx). However, in UNIX, case can be significant. We recommend you stick with lower-case letters.

Also, when your project uses multiple directories, note that Everest transfers all such files into a single cache location on the user’s local computer. Consequently, for example, if it has already transferred graphics\cat.pcx into the cache, it may not recognize that images\cat.pcx might be a different image. To avoid confusion, do not store identically named files in different directories.

What About ERUN?

For end users to play back your project, they will need to run a copy of the Everest ERUN project player program on their desktop computer. The ERUN program is not a plug-in; instead, it operates much like a Web browser.

If you are licensed to distribute the ERUN program, there are several ways you can provide a copy to the end users. You can post it (together with its support files as described in the Preparing Your Project for the User chapter) on the Internet and allow people to download it. You might also simply send them a copy on diskette or CD-ROM.

When the end user receives a copy of ERUN, he will need to install it. That can be as simple as copying it into a subdirectory on his hard disk drive. You might consider including a SETUP program to help the user perform the installation.

User's Computer Configuration

For ERUN to operate correctly, the end user's computer must already be set up for Internet or intranet (INet) access. Here is the minimum configuration:

80386 or better computer

4M RAM

VGA display capability

Windows 3.1, 95 or compatible

operational modem (or other INet connection)

WINSOCK (or other compatible INet connection software)

INet service (i.e. *not* a custom on-line service, such as Compuserve or America On-Line)

The bottom line: if the end user's computer can access the INet via a browser such as Netscape Navigator or Microsoft Internet Explorer, then ERUN should also be able to do so.

14.7 Testing Your Project on the Internet

Once you have uploaded your project to the desired Internet/intranet site, you can test run it from within AUTHOR.

Testing Your Project

You will need to know your project's URL. A URL resembles

```
http://www.insystem.com
```

Contact your system administrator if you do not know the URL for where you uploaded your project.

Direct Button

From the main Author window's Run menu, choose Start at. In the window that appears, click the Direct button. The Direct button lets you type the specific location (URL, hard drive, whatever) of your project. You must also include the name of the book as well as the name of the page at which to start running; use a semicolon to separate the book and page. For example, you might enter something that resembles:

```
http://www.insystem.com/evdemo/@start;@start
```

Everest will then attempt to connect to the site, and begin running your project. The Direct button uses exactly what you type, so be sure to employ upper and lower case letters as needed.

Problems?

For this to operate correctly, your computer must already be set up for direct Internet or intranet access. Everest will not work via custom on-line service providers such as CompuServe or America On-Line.

To test if your computer is properly set up, try running a standard Web browser such as Netscape Navigator or Microsoft Internet Explorer. If these browsers work, then Everest should also.

Make sure the URL you entered is correct. Case is significant in URLs. Try the URL above, which, as of this writing, is the location of the Everest demonstration disk.

Internet History

If you have connection problems or errors, the Internet History window can often reveal the location of the trouble. From the main Author window, choose INet History. This opens a window that contains the most recent messages between Everest and the server. Features on the pull-down menu let you print or save the messages to disk.

The ERUN program has the same INet History window, though it is normally hidden from the user. If an end user is having trouble running your project, instruct him to start ERUN with `/history` on the command line. Alternatively, inside your project, use the following A-pex3 programming to open the window

```
dummyvar = ext(126,1)
```

14.8 Editing Your Project on the Fly

If you spot a problem with a page while you are test running your project, from the main Author window's Run menu, choose Edit on the fly. Even if your project is hosted on the Internet, you can make changes in real time; Everest will automatically upload the changes when you save them.

Edit-on-the-fly is probably the single most valuable authoring system feature because it saves the most time.

FTP Settings

If you are thinking ahead, you might wonder if anyone on the Internet with a copy of Everest could modify your project via the Edit-on-the-fly feature. The answer is no...not unless they have the necessary FTP information and uploading passwords.

In fact, even you won't be able to save your Edit-on-the-fly changes until you tell Everest certain FTP (file transfer protocol) information. You should enter this information in the Settings window (from the main Author window's File menu, choose Settings).

You will need to enter the proper FTP information in the Internet/intranet area of the Settings window. Click the Instructions button for examples and help for each field.

FTP Upload Password

On the Settings window, for security reasons, we recommend that you leave the FTP Upload Password field empty. If you do choose to enter your password, please note that it will be stored in an unsecure format in the EVEREST.INI file. If you leave the upload password field empty, Everest will prompt you to enter the it immediately prior to performing an upload, and will not store it in the EVEREST.INI file.

Other Protection

Since your Internet/intranet site is probably designed so that only authorized users will be able to upload files into the area in which you install your project, it is unlikely someone would be able to edit your project against your will. Still, you may wish to employ additional measures to discourage unauthorized users from modifying your project.

Book Passwords

You can apply an editing password to each book. The AUTHOR program will prompt for the password before it opens any passworded book for editing. To apply a password to an open book, use the Book menu and choose Password.

The Project Packager also lets you specify a password (if you make a book granular).

Editing Lock

Both the Project Packager and the Copy Page utility let you lock individual pages to prevent further editing by anyone, including yourself! Be sure you have a master un-locked copy of any pages you are considering locking.

14.9 Configuring ERUN for Internet Play Back

As mentioned previously, the end user needs a copy of the Everest ERUN program in order to play back your projects. Before you can distribute ERUN, you must be licensed to do so. Contact Intersystem Concepts to obtain such a license.

ERUN and EVEREST.INI

ERUN looks in the EVEREST.INI file to determine what project to run, as well as what configuration to apply (for example, whether or not to collect user records). By default, ERUN looks for the EVEREST.INI in the same location (i.e. subdirectory) from which it (ERUN) is being executed. For Internet play back, this default approach is usually undesirable. Why? Let's look at an example.

The Problem

Perhaps at the moment you have only one project you wish to deliver via the Internet. In the future, it is likely you will have more. The greater the number of projects you have, the greater the chance is you will want to execute them with different configurations. You might, for example, want to collect user records for the first project, but not the second.

Such different configurations require different settings in EVEREST.INI. Since, by default, ERUN looks for the EVEREST.INI in its (ERUN's) location, having different EVEREST.INI files would mean you'd need to install a copy of ERUN for each project. That's inefficient, and wastes hard disk space on the end user's computer.

The Solution

There are several possible solutions to this problem, however, the one outlined by the steps below is the one we recommend.

1) When you provide the end user with the ERUN program, do *not* also provide the EVEREST.INI.
2) Instead, keep the EVEREST.INI stored with the other files on the Internet that make up your project.
3) When you setup ERUN on the end user's computer, create an icon in Windows to launch it. This can be done manually via the Microsoft Windows Program Manager, or via an installation utility, such as InstallShield.
4) If you view the ERUN icon's Program Item Properties, you will see a Command Line that resembles:

   ```
   C:\EVEREST\ERUN.EXE
   ```

   (The directory varies, of course, based on where ERUN.EXE is located.) With this default Command Line setting, ERUN will expect to find EVEREST.INI in the same location.
5) This is the key: tell ERUN to look elsewhere for the EVEREST.INI. To do so, add /ini=[location] to the Command Line. For example, if you changed the Command Line to resemble:

   ```
   C:\EVEREST\ERUN.EXE /ini=C:\proj1
   ```

   then ERUN would look for EVEREST.INI in the C:\proj1 subdirectory. If you keep EVEREST.INI with your project on your Internet site, as we recommend, the Command Line might then resemble:

   ```
   C:\EVEREST\ERUN.EXE /ini=http://www.xyz.com/proj1
   ```

The Benefits

The benefits to this approach are many. First, you can create many such program item icons in Windows that all point to the same copy of ERUN, but which use different EVEREST.INI configuration files. Therefore, one copy of ERUN can execute any number of different projects.

Second, if you need to change the EVEREST.INI file, you can do so on your Internet site, and everyone who uses your project will instantly have the updated configuration.

Some Possible Issues

If your project's EVEREST.INI is stored on your site, each user will reference the same copy. However, there is no guarantee that each user's computer will be configured identically, and therefore you must use care when choosing the settings of various items, especially those that involve directories.

Where's the Cache?

Of primary concern is the location of the download cache ERUN will employ while retrieving files from your site. This is defined in the EVEREST.INI file via the CachePath item. You might set it to something like `C:\evcache`, but there is no guarantee that such a drive or subdirectory exists on the user's computer. Fortunately, there's an easy solution.

Here's what to do: set the CachePath entry to employ Everest's %: drive letter wildcard. This would, for example, resemble:

```
CachePath=%:\evcache
```

The %: symbol is Everest's wildcard for the "path to the currently executing program." In this case, that's ERUN, and %: will therefore be a disk drive that exists on the end user's computer. So, for example, if ERUN is located in C:\EVEREST, then the setting above will result in a CachePath of C:\EVEREST\EVCACHE.

If you do not want the cache located underneath the directory that contains ERUN, then omit the \, as in:

```
CachePath=%:evcache
```

So, if ERUN is located in C:\EVEREST, then the setting above will result in a CachePath of C:\EVCACHE.

But, you wonder, what about the \EVCACHE directory? What if it does not exist? The answer is, if the CachePath directory does not exist, ERUN automatically creates it for you, provided it is not more than one level deep. Wait. There's one more issue about the cache.

If you are paying very close attention, you'll notice there's a chicken-and-the-egg type problem here. CachePath is configured in the EVEREST.INI, so when ERUN retrieves the EVEREST.INI from the Internet, it does not yet know what the CachePath is. So, what does it do with the copy of EVEREST.INI it downloads? It keeps it entirely in memory...problem solved.

For more information about the cache, see the topic named "Cache Options" later in this chapter.

Where are the user records and comments?

Just as with CachePath, we recommend you use the %: wildcard to configure the UserRecs and Comment items in the EVEREST.INI. ERUN will create the subdirectories for each, provided they are no more than one level deeper than the currently existing subdirectory.

What's the version?

It is also possible that the user is attempting to run your project with a version of ERUN that is older than the one your project needs. The first page of your project can check the value returned by the Ext(23) function...that will be ERUN's version number. Your project can also check the string returned by the Ext(123) function, which is the date and time that ERUN.EXE was last modified. If ERUN is too old, display a polite message to this effect, then `BRANCH @exit`.

Different Configurations of the Same Project

You might want to allow different types of access to your project. For example, you might want to configure the EVEREST.INI without user records for some people (perhaps to allow them to run a demonstration of your project), and with records for others.

You certainly could store two separate copies of your entire project on your Internet site, with the only difference being the EVEREST.INI configuration. However, there is a much better way: simply invent a different name for the EVEREST.INI.

For example, you might rename the EVEREST.INI that supports user records to SAVE.INI, and the one that does not NOSAVE.INI. (The file name extension must be .INI.) Then, simply include the .INI name when you configure ERUN's program icon in Windows. For example, the Command Line in the Windows Program Item Properties might resemble:

```
C:/PRO/ERUN.EXE /ini=http://www.xyz.com/proj1/nosave.ini
```

Increasing Start Performance

First impressions, however unfair they might be, do make an impact. If your project is slow to start, it may be perceived as slow throughout. When ERUN starts, its first task is to obtain the EVEREST.INI file. There is little you can do to speed up this part of the process; if the EVEREST.INI is on the INet site (as we recommend), ERUN will download it from there, and establishing the connection to do so will take a few moments.

Without Log on

If your project does not employ user records (i.e. no log on), ERUN will next start running the book specified in the ProjectFile item in the EVEREST.INI. If this book is large, and ERUN needs to download it, there will be a delay. To minimize the delay, you should consider making the book granular. Alternatively, you might instead create a small book that contains nothing more than a title page. Such a title page would display a "One moment, please" type message, then branch to the larger book that contains the bulk of your project. This approach quickly gives the user something to look at while your project is getting started.

With Log on

If you are employing a custom log on (an @logon page), you must set the LogOn item in the EVEREST.INI to a value of 5, otherwise, in the interest of expediency, ERUN will skip the check for @logon.

If you are using ERUN's default log on window, set the LogOn item to a value of 1. This tells ERUN you are not employing an @logon page, so it can quickly display the default log on window. Plus, while the user is entering his/her log on information, ERUN will be busy in the background downloading the initial files needed.

Splash Screen

When ERUN first starts, and while it is obtaining the EVEREST.INI file, it displays a small Everest copyright window. You can overlay this window with an image of your choice. To do so, set the Splash item in the Settings section of the EVEREST.INI to the name of the desired .BMP file. For example:

```
Splash=C:\graphics\corplogo.bmp
```

This file need not be on the local computer...you can also specify a URL for the image, except, of course, it will take a moment for ERUN to download it (perhaps defeating the purpose of the splash screen). The image must be in .BMP format (or a .ZIPped .BMP).

14.10 User Records and the Internet

As described in the Record Keeping chapter, you can configure your project so that Everest automatically stores user records and a user bookmark. Everest keeps this information on the user's computer in the location you specify via the UserRecord item in the EVEREST.INI.

In many situations, particularly those involving Computer-Based Training, you will want to monitor the user's progress, generate reports, etc. When the users are running your projects locally, this is easy to do...you simply use the Everest INSTRUCT program to examine the user records files.

However, when the users are remote, such as somewhere on the Internet, monitoring their progress requires an extra step: you must obtain a copy of their records/bookmark file.

Transferring User Records

One way you could obtain a copy of the user's record is for them to manually send a e-mail message to you with the records file attached. The drawback to this approach is that it requires a manual step for the user.

A better approach is for your project to ask ERUN to upload a copy of the user records to your FTP site. This is easy to do. Here are the steps:

1) In your book, create a page named @finish. Everest automatically runs the @finish page immediately after a user quits (or your project executes the `BRANCH @end` command). The @finish page is normally the place you tell the user something like "Thank you for using this application. Remember, you can restart later and resume where you left off."
2) On the @finish page, put a Program object
3) In this Program object, place the following A-pex3 programming:

```
ecode = rec(12)
IF ecode # -1 THEN
  dummyvar = mbx("Error " + ecode + " uploading.")
ENDIF
```

That's it. Operation 12 of the Rec() function uploads the user records file to the FTP site you've specified via the various FTP configuration items in the EVEREST.INI file.

CMIFile

To transfer the CMI data collection file, use Rec(13) in place of Rec(12) in the example above.

Security Issues

For ERUN to be able to upload to your FTP site, it will need to know the correct password. You can specify this via the EVEREST.INI file, or via programming in the project itself. If you will not be giving end users a copy of the EVEREST.INI file, and will instead be keeping it on your site (as we recommend), then the most secure location for this password is in the FTPPassword item in the EVEREST.INI.

If you do intend to provide copies of the EVEREST.INI file to the end users, then the more secure location for your password is in the project itself. Immediately prior to uploading the records, set Sysvar(186) to your password. The Program in the @finish page might then resemble:

```
sysvar(186) = "opensesame"
ecode = rec(12)
sysvar(186) = ""
IF ecode # -1 THEN
  dummyvar = mbx("Error " + ecode + " uploading.")
ENDIF
```

Name Conflicts

If multiple users from various locations upload their records file to your site, won't each subsequent upload overwrite and replace the previous? The answer is no, not if you use the Rec(12) function. The reason is because Everest generates a random file name to assign the uploaded file. Everest remembers this name in the Sysvar(197) variable, and re-uses it each time the user logs off. The file name has the extension ".eur". Theoretically, it is possible Everest would randomly generate the same random file name for two different users, but since there are over 200 billion possible names, this is extremely unlikely.

If you prefer to upload the file and assign it the name of your choice, use the Fyl(43) function instead of Rec(12).

Examining Uploaded Records Files

To examine the records files that have been uploaded to your site, first download them from your FTP site to your local computer (using software of your choice), then open the files with the Everest INSTRUCT administrator program. With INSTRUCT you can, if you desire, combine the individual files into one master database, and generate reports.

14.11 Limiting Access to Your Projects

If you place your projects on the World Wide Web, then anyone who has a copy of ERUN will be able to play them back. However, you may want to limit access to your project. Perhaps you want to restrict it to only people in your organization, or only those that have a certain password, or only those who have paid a fee. This topic tells you how.

Passwords

Everest offers several types of passwords you can use to help guard against unauthorized access to your project. The general approach is to ask the user to enter a password in order to continue. You might request the password near the beginning of your project; the @start page is a good place. There are several password techniques you can use.

Adequate Protection

The simplest password approach involves using Everest's Ibx() input box function to requesting the user to enter the password; then comparing it to a string. An A-pex3 Program to do would resemble:

```
pw = ibx("Please enter the password")
IF pw = "opensesame" THEN
```

```
  pw = mbx("That's correct.  Click OK to proceed.")
  BRANCH page2
ELSE
  pw = mbx("Incorrect password.  Click OK to exit.")
  BRANCH @exit
ENDIF
```

In the example above, the correct password (opensesame) has been hard-coded into the Program. That makes it relatively easy to crack, and means the password is the same for all users. Of course, you can edit your page and change the hard coded password as frequently as you like.

Better Protection

A more secure password is one that changes at a regular interval. Perhaps you'd like a password that changes periodically, and is not hard coded in the project. That's exactly what the Ext(115), Ext(116) and Ext(117) functions do. They generate a seemingly random password based on the current year, month or day (respectively) and a character string you specify. The password always contains 9 digits. Here's an example:

```
correct = ext(116, "seedstring")
pw = ibx("Please enter the password")
IF pw = correct THEN
  pw = mbx("That's correct.  Click OK to proceed.")
  BRANCH page2
ELSE
  pw = mbx("Incorrect password.  Click OK to exit.")
  BRANCH @exit
ENDIF
```

In the example above, the Ext(116) function generates a password consisting of 9 digits. The user might obtain the password by calling you, requesting it via e-mail, etc. Since the password is not hardcoded, but is instead generated at run time, there is no way for the user to obtain it by breaking into your page.

You will need to know the password, though, to provide to authorized users. You can generate it quickly in the AUTHOR program; from the Utilities menu, choose Evaluate Expression. Then enter the Ext() function as it exists in your Program; in the case of the example above, you would enter `ext(116, "opensesame").`

Given a particular seed string, the Ext(116) function will return the same password for the entire calendar month. Use Ext(115) or Ext(117) if you want the password to be valid for a different period.

Getting Fancy

Nine digits might be difficult to enter or keep track of. Sometimes adding hyphens can make the number easier to remember, or describe over the telephone, etc. This is easily done with help from the Fmt() function. Here's the same example as before, except using the Fmt() function to break the 9 digit number into 3 sections of 3 digits each:

```
correct = fmt(ext(116, "seedstring"), "000-000-000")
pw = ibx("Please enter the password")
IF pw = correct THEN
  pw = mbx("That's correct.  Click OK to proceed")
  BRANCH page2
ELSE
  pw = mbx("Incorrect password.  Click OK to exit.")
  BRANCH @exit
ENDIF
```

Best Protection

The previous example still uses the same password for all people running your project. Perhaps you are charging a fee for the user to access your project. In this case, you can instruct them to telephone you for the password, and "have their credit card ready" when they do. You'll probably want to generate an individualized password, so that others will not be able to use your project without also calling you.

The way to make this work is to specify a seed string that is unique to the user...for example, their log on name. The user's log on name (last name) is available in Sysvar(132). Here's an example:

```
correct = fmt(ext(116, sysvar(132)), "000-000-000")
pw = ibx(sysvar(132)+", please enter the password")
IF pw = correct THEN
  pw = mbx("That's correct.  Click OK to proceed.")
  BRANCH page2
ELSE
  pw = mbx("Incorrect password.  Click OK to exit.")
  BRANCH @exit
ENDIF
```

This example is the almost the same as the previous, except, now, Sysvar(132) is the seed string. Note that it is also displayed in the password window. When the user telephones for the password, ask them to tell you the name displayed in the password input window. To determine their password, use AUTHOR's Evaluate Expression feature, and type it in yourself.

Access Lists

Still, passwords can be broken (or obtained) by a very determined person. For maximum security, consider using an access list. With an access list, you can limit use of your projects to those people you designate by log on name.

Creating an Access List

The INSTRUCT program helps you create access lists. The process is as follows:

1) Add to a user records file the people you wish to authorize to use your project. The INSTRUCT program lets you specify the log on last name, first name, group, ID and a password.
2) When you have added all the people you wish, use the Generate Access List option on the INSTRUCT program's pull-down menus. Assign the file a name that matches that of the User Records file that will be indicated by the UserRecs item in the EVEREST.INI, except employ a file name extension of .EAL (for Everest Access List). For example, if your UserRecs item in the EVEREST.INI is USERRECS.EUR, then the access list must be named USERRECS.EAL.
3) Upload this .EAL file to the Internet/intranet location containing your project.

Checking the Access List

There is one more important step. Your project must check the access list. To do so, employ the Rec(14) function early in the project, such as on the @start page. An A-pex3 program to do so might resemble:

```
IF rec(14) = 0 THEN
  dummyvar = mbx("You are not authorized", 16)
  BRANCH @exit
ELSE
  BRANCH page2          $$ ok to continue
ENDIF
```

When the Rec(14) function returns 0, it means ERUN could not find the user's log on information in the access list. Rec(14) will also return 0 if the user bypassed the log on process (i.e. is running your project without logging on).

14.12 Cache Options

As ERUN runs your project, it obtains the appropriate files (your project's books, graphics, etc.) from the Internet/intranet site. It immediately copies these downloaded files into a location on the local computer (usually a subdirectory on the user's hard disk). This location is called the cache (pronounced "cash"). As described in a prior topic in this chapter, you tell ERUN the location of the cache via the CachePath item in the EVEREST.INI file.

The configuration of the cache can have a significant influence on the apparent execution speed of your project. This topic describes how you can configure the cache for optimal performance.

Cache Defaults

The following defaults apply to the cache:

ERUN will use the current DOS default directory on the end user's computer as the cache. Rarely is this desirable, so you should set CachePath as described earlier in this chapter.

ERUN will also ignore any existing files in the cache. If it downloads a file from the Internet that has the same name as one already in the cache, the one in the cache will be overwritten.

When the ERUN program stops running, it cleans up (deletes) any files in the cache that it downloaded during the session.

Customizing Cache Handling

You can change how ERUN treats the cache. For example, you may want ERUN to not clean up the cache when done. This can be particularly handy if the user will be logging on again in the near future: the files won't have to be downloaded again, and your project will appear to run much faster.

You can control how ERUN treats the cache by changing the value of CacheOption in the AUTHOR program's Settings window. This sets the value of the CacheOption item in the EVEREST.INI file. You can also edit the EVEREST.INI file directly, or even change Sysvar(187) at run time via A-pex3 programming code within your project.

CacheOption controls several things at once. Add up the numeric value from the list below to determine what setting is right for your project.

1 Indicates most of your project files will be in .ZIP form, rather than in their uncompressed, original form. ERUN will look for a .ZIP file first.

2 If the uncompressed, original file cannot be found for downloading, do not also check for the same file in .ZIP form; instead, report that the file cannot be found.

4 Do NOT clean up the cache when done; leave downloaded files in the cache.

8 Check the cache first before downloading. If the required file already exists in the cache (such as from a download during a prior session, or from an initial installation), then use it rather than downloading another copy.

So, if you want to enable options 1, 4 and 8 above, in your EVEREST.INI file, you would put:

```
CacheOption=13
```

because 13 is 1 + 4 + 8.

Outdated Files in the Cache

Let's say you set CacheOption to 13. That tells ERUN to look for .ZIP files first, to not clean up the cache when done, and to always check the cache for a file that it may already have (and therefore not need to download).

There's a possible problem here. Let's say that after people have begun using your project, you correct something in it, and post the corrected file(s) to the site from which people download. If CacheOption is set so ERUN first looks in the cache, people who have used your project before will have the old version stored in their cache. ERUN won't know to download the new version!

CacheDate to the Rescue

Fortunately, there is an easy solution. In the EVEREST.INI file, set the CacheDate item to the date of the most recent modification to your project. Important: use YYYY/MM/DD format. For example:

```
CacheDate=1997/01/31
```

ERUN compares the CacheDate to the dates of each existing cache file it is considering using. If the file in the cache is older than CacheDate, ERUN downloads a new copy.

15 Converting From Summit

This chapter is intended for Summit for DOS (S/D) authors who are learning Everest and/or importing S/D courseware into Everest.

15.1 Comparing S/D and Everest

In S/D, screen libraries contain up to 2000 screens. Screen library 01 is stored in disk files SCNAME01.DAT, SCNUMS01.DAT and SCREEN01.DAT.

In Everest, the terminology is different: screens are called pages, and screen libraries are called books. The number of pages in a book is limited to approximately 4000. You can choose the first 8 characters of the disk file name of the book. The file name extension is always .ESL.

The overhead of Windows and the object-oriented design of Everest increases the disk space consumed by books. Books in Everest typically occupy two times as much disk space as in the equivalent content in S/D.

According to PC Magazine benchmarks, Windows applications run from 7 to 10 times slower than similar DOS applications, and Everest is no exception. The performance hit is the price we all pay for a graphical user interface. If you find Everest too slow, increase your computer's processing power by a notch. For example, upgrading from a '386 to a '486 should offset most of the performance overhead of Windows. Windows is the best friend the computer hardware manufacturers ever had.

15.2 Comparing Page Design

In S/D, when editing a presentation screen with AUTHOR, you can press <F3> to step through several editing pages of data about the screen: Screen Attributes Page, the text window (Screen Editor Page), the Calculation & Branching Page, and the Final Branching Page.

In Everest, these "pages" are handled as objects, and a page in your book simply becomes a collection of objects. For example in Everest, a presentation type page just like the one described above for S/D would have one of each of the following classes of objects: Layout, Textbox, Program and Wait.

Each new page you create begins its life without any objects. You add only those objects you want.

In Everest, your page is displayed both visually (much like XGRAPH), and as what is called an Book Editor (similar in concept to XGRLINE). Icons are used to represent each object in the Book Editor. The Book Editor becomes a flowchart of the execution order of the objects on your page.

You can arrange the objects to execute in any order. For example, if you want to perform calculations prior to displaying text, you simply drag a program object icon to the top. No need for Special Option 13.

You add objects to the VisualPage Editor or Book Editor by dragging their icons from the ToolSet. Icons dropped on the VisualPage Editor are appended to the bottom of the Book Editor.

To move an object on the VisualPage Editor to a new location, first move the focus to it (if necessary) by clicking on it with the left mouse button, then point to one of its sizing handles, press the right side mouse button and drag. To resize instead, point to a sizing handle, press the left button and drag.

To change an object's place in the Book Editor, point to it, hold down the right mouse button, and drag it to the desired location in the Book Editor.

All objects have attributes. Attributes specify things such as the object's display location and color, mode of operation, name, etc.

Objects are assigned a default name automatically. The name is made up of <page name>_<object name>_<next available letter>. You can modify the name as you wish. Names can be up to 19 characters in length.

Each object has an identification number attribute. This number is assigned automatically by Everest when you create the object. In most cases, on a given page, each object in a particular class (Textbox, Picture, etc.) has a unique IDNumber.

For special purposes, you can change the IDNumber to match that of another object in the same class. When the Book Editor of your page is executed, an object that has a unique IDNumber will be ADDED to the page display. An object that has the same IDNumber as a previous object will REPLACE that object on the page. The IDNumber must be between 1 and 99, inclusive.

If desired, you can load a previously created object (say, one you built for another page) into the current page. This creates an "instance" of the original object. Think of an instance as a subroutine call. This powerful feature lets two (or more) pages refer to the same object...therefore reducing the disk space needed to hold your project. It also means you can modify the object in one location, and have the changes automatically reflected in all other places the object is used.

Some object instancing is available in S/D. By specifying the same name in the Graphics Underlay field, two lesson screens can refer to the same Xgraphics screen. If you modify that Xgraphics screen, the changes appear in both lesson screens.

Everest gives you even more power. Certain object attributes are known as "instance" or "local" attributes. These can be changed locally (i.e. within the particular page) without the modifications appearing elsewhere. The object's display location coordinates are the most common local attributes.

15.3 Comparing Programming

In Everest, variables are not prefixed by { in programs. You can employ an @define Program object to create author defined variables, but you are not required to do so. To embed a variable for display within text, surround it with { }; for example: Today's date is {today}.

Everest's programming language is named A-pex3. In Everest, A-pex3 is easier to read, and more powerful than in S/D.

Other differences in Everest: string constants must be surrounded with quotes. Variable names can contain digits (first character must be a letter) and the underline character (ASCII 95). "IF" is not implied (you must type the word IF to start a condition). Use a colon to separate one calculation or command from another on the same line.

In IFs with multiple conditions, you can use the & symbol to represent AND. For example, in S/D:

```
{c1>0{c2>0@{c3>0{c4>0:|tworight
```

becomes in Everest:

```
IF c(1)>0 & c(2)>0 @ c(3)>0 & c(4)>0 THEN BRANCH tworight
```

The & is not required, but is recommended for readability.

All system variables are now found in an array named Sysvar(). They are read-only, unless otherwise indicated elsewhere in the documentation. All S/D system variables are not considered system variables in Everest. The functionality of most S/D system variables can be obtained by translating them as indicated below:

{a1-{a4	Mse() function, Sysvar(120) to Sysvar(124)
{c#	JudgeVar attribute
{q1	Sysvar(5)
{q2	Sysvar(6)
{q5	Rnd() function
{q6	Period attribute
{q7	Tim() function
{q8	not supported
{q9-{q11	SCALE command
{q13	not supported
{q15	Position attribute
{q16-{q19	not supported
{q20-{q30	Button objects
{q31	Obj() function
{q32	not supported
{r#	ResponseVar, AdjustResponse attributes
{x#	ResponseVar, AdjustResponse attributes
{y1	Sysvar(9)
{y2	Sysvar(10)
{y3	Sysvar(11)
{y4	Tries attribute
{y5	Sysvar(12)
{y6	Sysvar(56)
{y7	Sysvar(131)
{y8	Sysvar(132)
{y9	Sysvar(133)
{y10	Sysvar(134)
{y11	Sysvar(139)
{y12	Sysvar(140)
{y13	Sysvar(135)
{y14	Sysvar(137)
{y15	Sysvar(138)
{y16	Sysvar(58)
{y17-{y24	not supported
{y25-{y32	Sysvar(61) to Sysvar(68)

{z1	Dat() function
{z2	Tim() function
{z3	Sysvar(16)
{z4	Sysvar(18)
{z5	Sysvar(17)
{z6	JudgeVar attribute
{z7	Sysvar(1)
{z8	Sysvar(130)
{z9	not supported
{z10	Gdc() function
{z11	Ini() function
{z12	not supported
{z13	Mse() function
{z14-{z15	not supported
{z16	Ini() function
{z17	Sysvar(16)
{z18-{z19	not supported
{z20-{z25	not supported
{z26	Sysvar(58), Sysvar(100)
{z27	not supported
{z28	not supported
{z29-{z30	JudgeActivator1, JudgeActivator2 attributes
{z31	Sysvar(81) to Sysvar(88)
{z32	Sysvar(71) to Sysvar(78)

Most functions operate identically in S/D and Everest. Here is a list of those that have differences (for new and changed functions, refer to the Technical Reference for details):

Arr()	new (number of elements in an array)
Cvi()	new (converts Mki() string back to number)
Dat()	same, plus new dat(5) that returns day of week number (1 = Sunday, 7 = Saturday)
Dde()	new (Microsoft Windows Dynamic Data Exchange)
Dll()	new (call DLL routines)
Ext()	the following ext() functions are supported without change: -26 to 0, 2, 6, 7, 8, 13, 22, 23, 36, 39, 42, 43, 44, 45
	the operation of the following ext() functions has been changed (details in tech ref): 1, 3, 4, 5, 9, 10, 11, 12, 14, 15, 16, 17, 18, 19, 24, 25, 26, 27, 28, 29, 41, 51
	the following ext() functions are not supported: 20, 21, 30, 31, 32, 33, 34, 35, 37, 38, 40, 46 to 50, 52, 53
Fmt()	new (fancy format for numbers and strings)
Fnt()	new (font information)
Fre()	changes
Fyl()	new (read/write disk files)
Gdc()	new (Microsoft Windows GetDeviceCaps)
Gsm()	new (Microsoft Windows GetSystemMetrics)
Hex()	new (hexadecimal)
Hlp()	new (link to Microsoft Windows help system)

Ibx()	new (simple user input box)
Ini()	new (read/write .INI files)
Key()	new (keyboard features)
Lod()	use on arrays
Mbx()	new (simple user message box)
Mci()	new (Microsoft Windows Media Control Interface)
Mid()	new (substring parsing)
Mki()	new (converts integer to 2-byte character string)
Mse()	new (mouse)
Msg()	reads from the EVEREST.MSG file
Obj()	new (object information)
Ply()	new (play musical notes; substitute for Special Option 11)
Pop()	not supported
Pth()	new (file name parsing)
Rec()	new (user records; substitute for Special Options 15 and 26)
Reg()	new (region string)
Rpl()	new (string replacement)
Scl()	not supported
Scn()	same, but value returned is not unique to page
sfl()	new (shuffle attributes randomly)
Shl()	changes
Srt()	use on arrays
Stf()	changes
Sum()	use on arrays
Typ()	new (storage format of variable contents)
Var()	new (variable exists)
Vfr()	Position attribute

15.4 Comparing Interaction

You can (and should) design your Everest project to be "event driven." This means you specify the events (such as a mouse click) that you want a page to respond to. You also specify the actions to take (such as "branch to a menu") when those events occur.

Each event has a unique numeric event code. You invent and assign the codes to the events as you wish. For example, many objects have a ClickEvent attribute. A click event occurs when a user clicks the mouse on the object. For example, you can assign the ClickEvent attribute of a button an event code of 12345. Any number from -32000 to 32000 is allowed.

Every time a user presses a key or clicks the mouse, an event is generated. As in S/D, every key has a numeric keycode (event code). Thanks to Microsoft, most of the keycodes are different from those in S/D (*sigh*).

Interaction is handled somewhat differently in Everest. Each question field is an independent object: Input, Check, Button, Option, etc.

On pages with multiple fields, the order in which the focus (highlight) moves through the fields (the Tab order) is dictated by the order in which you arrange the fields in the Book Editor.

IMPORTANT: the question fields you place on your page are not enabled for user input until Everest encounters a Wait object in your page. For example, the typical interactive page contains one or more question field objects, followed by a Wait object.

User responses are not judged for accuracy until a Judge object is encountered in the page. After judging, execution of the page continues at the object that follows the Judge object. That's a good place for feedback or branching.

Pages can contain up to 99 of any object class, therefore you can have 99 input fields, 99 check boxes, etc.

Anticipated answers are specified individually with each field. There is no "Anticipated Answers Page" that has all answers for all fields. You enter your specifications in the Answers attributes for the field.

With each field you can specify the variable in which to store the user's response, and the variable in which to store the answer judgment. Unlike S/D, you are not forced to use the {r# or {c# variables. As a matter of fact, if you don't care to save the response or judgment, simply don't specify a variable for the ResponseVar or JudgeVar attributes.

15.5 Comparing Branching

In S/D, a branch command is represented by the | character. In Everest, use the word BRANCH in place of the |.

In S/D, most branching is of the "go to" variety. In Everest, you can also use the CALL command, which is a "go sub" type branching command. Use the RETURN command to return from a CALL.

In S/D, you suffix the name of the screen library when branching across libraries; for example: |across;02 (this opens the SCxxxx02.DAT files). In Everest, you prefix the book name; for example: BRANCH lib02;across (this opens the LIB02.ESL file).

Some special branching keywords are different:

Purpose	In S/D	In Everest
Branch to previous page	{p	@prev
Branch to recent menu	{m	@menu
Return from subroutine	{r	RETURN command
Save recs, & branch to @finish	end	@finish
Save recs & end project	@end	@end
Exit project (don't save recs)	@exit	@exit

15.6 Comparing Execution

In S/D, the execution order of the elements of each screen is (almost) cast in concrete...first the Special Option, then the graphics underlay, then the text window, etc. In Everest, you define the order in which the elements execute by the placing them in the Book Editor.

In S/D, each screen overlays the prior one, obscuring a portion or all of it. Everest works in a similar fashion.

Pages and the objects they contain are displayed within a window. IMPORTANT: an object you add to the window (say, a Textbox) remains active and enabled in that window until one of the following:

1) execution of an Erase object in the Page

2) execution of an Erase command in an A-pex3 program

3) the object is replaced by another in the same class with the same IDNumber

4) the object is deleted via the Destroy attribute

This means that while your project branches from one page to another, by default the objects stay where they are.

15.7 Special Option Substitutes

Everest has no "Special Option" as is found on the Final Branching Page of S/D. The functionality of most Special Options can be reproduced in S/D. Here are suggestions:

Special Option #	Substitute
0	put a Special object at the top of the Page
1	InputTemplate attribute
2	none
3	Shl() function
4	none
5	TabStop attribute
6	Rec() function
7	Rec() function
8	NextActivator attribute
9	none
10	Preset attribute
11	Ply() function
12	none
13	put a Program object where needed in Page
14	SetFocus attribute
15	Rec() function
16	Mci() function
17	SpecialEffect attribute
18	none
19	LPRINT command
20	none
21	none
22	none

23	remove Textbox from Page
24	none
26	Rec() function
28	Enabled attribute

15.8 Importing from S/D via EXLATE

The quickest and easiest way to convert your S/D courseware into Everest format is to start with the EXLATE program. EXLATE.EXE is a DOS program that you should copy to and run from the subdirectory that contains your copy of S/D.

EXLATE converts approximately 80% of the content of most S/D courseware. Specifically, here is what it does:

1) Converts Screen Attributes Page information to a Layout object.
2) Converts the text window to a Textbox or Flextext object. In a Textbox, the foreground color of individual characters is not preserved (i.e. all text is the same color). In a Flextext object, the foreground color is preserved.
3) Converts questions fields to their Window equivalent (as closely as possible).
4) Converts the Calculation and Branching Page to a Program object. Commands are translated into A-pex3 language equivalents.
5) Converts the Final Branching Page to a Wait object. Keypress codes are translated into their Everest equivalents.
6) Converts Xgraphics to a Program object. Commands are translated into A-pex3 equivalents. Those that cannot be converted are stored in their original form, prefixed with $$*.
7) Assembles a page for each lesson screen.

EXLATE is not a one-step, do-it-all, complete conversion utility. It is simply a tool to help you move programming, graphics and text from S/D to Everest.

EXLATE can convert more than half of the S/D Xgraphics commands. Those it cannot: Palette, Get Image, Put Image, Font, Animate, Load charset, Load fontset, Load binary (except .BFT), EGA Palette, Student Interaction Point, Videodisc, Input.

The functionality of most Xgraphics commands that do not translate can still be found in Everest. You need to manually convert such commands into their substitutes.

S/D Xgraphics Command	Everest Substitute
Palette	Picture object to load .DIB file
Get Image	none
Put Image	Picture object
Font	A-pex3 FONT command
Load fontset	A-pex3 FONT command
Animate	Animate object
Load charset	A-pex3 FONT command
Load binary	Picture object
EGA Palette	Picture object to load .DIB file

Student Interaction Point	Wait object
Videodisc	Media object or Mci() function
Input	Input object

The functionality of certain other commands is changed by EXLATE:

S/D Xgraphics Command	Notes
Arc	Converts to a Circle command with arc parameters. Microsoft changed how the arc drawing feature operates...your lesson might be impacted.
Load binary	Converts Load .BFT Bitfont commands into FONT command to load similar looking Windows fonts; handles all S/D fonts found in \SAMPLES except OE.BFT.
Transform	Converts to a loop; you can restore the Transform functionality only by recoding your graphics commands to employ variables.
Round box	Microsoft's change to the arc drawing feature might impact the appearance of your rounded boxes, especially very small (thin and/or short) ones.
Bitfont	Size and effects (underline, italics, etc.) are not translated.

Appendices

Appendix A - Keypress Codes

The following chart shows the event codes that are generated by various key presses. (For the proper codes to use with the Chr() and Asc() functions, refer to the ASCII codes table found later in this topic.)

Backspace	8
Tab	9
Center 5	12
Enter	13
Shift alone	16
Ctrl alone	17
Alt alone	18
Pause	19
CapsLock	20
Escape	27
Space	32
PgUp	33
PgDn	34
End	35
Home	36
Left	37
Up	38
Right	39
Down	40
PrintScreen	44
Ins	45
Del	46
0	48
1	49
2	50
3	51
4	52
5	53
6	54
7	55
8	56
9	57
a	65
b	66

c	67
d	68
e	69
f	70
g	71
h	72
i	73
j	74
k	75
l	76
m	77
n	78
o	79
p	80
q	81
r	82
s	83
t	84
u	85
v	86
w	87
x	88
y	89
z	90
numeric 0	96
numeric 1	97
numeric 2	98
numeric 3	99
numeric 4	100
numeric 5	101
numeric 6	102
numeric 7	103
numeric 8	104
numeric 9	105
numeric *	106
numeric +	107
numeric -	109
Decimal	110
numeric /	111
F1	112
F2	113
F3	114
F4	115
F5	116
F6	117
F7	118
F8	119
F9	120
F10	121
F11	122
F12	123
NumLock	144
ScrollLock	145
;	186
=	187
,	188

-	189
.	190
/	191
`	192
[	219
\	220
]	221
'	222
Shift	add 1000
Ctrl	add 2000
Alt	add 4000
Key Up	add 8000 (enable by setting Sysvar(170) = 8)

Examples:

Key	Code Generated
a	65
Shift+A	1065
Ctrl+A	2065
Alt+A	4065
Ctrl+Alt+A	6065

Use the key(5, EventCode) function to convert an event code to its ASCII equivalent. For example, key(5, 65) returns 97 (the ASCII code of the lower-case letter a). If an ASCII equivalent does not exist, the Key() function negates and returns the EventCode parameter.

When editing any of the xxxEvent or xxxActivator attributes, you can have Everest automatically generate the event code for a particular key. To do so, double click on the attribute name in the Attributes window; when the "Generate Keypress Event Code" window appears, press the desired key.

ASCII CODES

The following table shows the codes used by the Chr() and Asc() functions:

Line Feed	10
Enter	13 (also known as Carriage Return)
Escape	27
Space	32
!	33
"	34
#	35
$	36
%	37
&	38
'	39
(	40
)	41
*	42
+	43
,	44

-	45
.	46
/	47
0	48
1	49
2	50
3	51
4	52
5	53
6	54
7	55
8	56
9	57
A	65
B	66
C	67
D	68
E	69
F	70
G	71
H	72
I	73
J	74
K	75
L	76
M	77
N	78
O	79
P	80
Q	81
R	82
S	83
T	84
U	85
V	86
W	87
X	88
Y	89
Z	90
[	91
\	92
]	93
^	94
_	95
`	96
a	97
b	98
c	99
d	100
e	101
f	102
g	103
h	104
i	105
j	106
k	107

l	108
m	109
n	110
o	111
p	112
q	113
r	114
s	115
t	116
u	117
v	118
w	119
x	120
y	121
z	122
{	123
|	124
}	125
~	126
Others	127 to 255

Appendix B - Bounds

Windows

Max windows open simultaneously at run time	8
Max memory for objects per window	64000 bytes
Max page size	32000 bytes

Attributes

Max length of Text attribute	32000 chars
Max length of other string attributes	250 chars

Variables

Max unique names for author defined variables	2500
Max number of variables and array elements	32000
Max memory for author defined variables	Windows memory
Max memory for author defined variables at user log off	64000 bytes
Max memory for system variables	Windows memory
Max memory for system variables at user log off	64000 bytes
Max string length	32000 chars
Min numeric value	-3.4 x 10^38
Max numeric value	3.4 x 10^38

A-pex3 Program Code

Max program size	32000 chars
Max line length	250 chars
Max function and parentheses nesting	8 levels

External Objects

Max objects per external file	8
Max external files loaded at once	16

Appendix C - Error Messages

There are three types of error messages you might encounter in Everest: 1) GPF (global protection fault), 2) Everest specific error messages, and 3) Microsoft error messages.

GPF Errors

Global protection fault errors (better known as GPFs) are the result of bugs in Windows, other Microsoft software, or other third-party software. They can result from unexpected values in memory, as well as a host of other causes. Video display drivers are notoriously buggy and are often the cause of GPFs; you might try running Windows in a different resolution and/or color depth to see if that cures the GPF. Since all GPFs are caused by bugs in software external to Everest, there's little we can do to correct them. If you can repeat a GPF at will, let us know how: we'll need to know step-by-step instructions for creating a page from scratch that repeats the problem. If we can also duplicate it, there's a small possibility that we can adjust Everest to work around it.

Everest Specific Errors

Everest specific error messages are represented by numeric codes less than 0. These error messages are usually caused by mistakes in your project, such as attempting to branch to a page that you have not yet created.

-098 Renamed application
    Do not change the name of the Everest .EXE files.

-099 Contact tech support
    Report this error to technical support.

-112 Insufficient memory for operation
    There is not enough memory available to perform an action.

-114 File not found
    A file was not found on disk.

-119 Internal error
    A problem occurred in Everest. Please report this error to tech support.

-123 Page name missing
    Everest expected you to enter the name of a page.

-131 Trouble reading current directory
    Everest had trouble reading the files on disk. This could indicate a disk problem.

-134 Illegal or undefined variable name
    Variable names must start with a letter, and may contain letters, numbers and the underline character (ASCII 95).

-135 Illegal variable number
    You attempted to reference an array variable without specifying an element number.

-136 Variable not defined as array

You attempted to use a non-array variable where only an array is allowed.

-137 Out of array bounds
You attempted to reference an element outside the bounds of an array. For example, xyz(15) when the xyz array contains only 10 elements. Check your DIM and REDIM commands. Also, use the Variables window to determine the number of elements currently in the array.

-138 Expression conclusion ("then") missing
The "THEN" part of a conditional expression is missing.

-139 Illegal character
In a page or object name, you attempted to use a character that is not allowed. Or, in an expression, Everest expected an operator such as = or +, but found some other character.

-142 Closing quote mark missing
Everest found an open quotation mark, but no closing one before the end of the expression. If you need to put a quotation mark in a string, use (34).

-143 Range error
A value was outside an acceptable range. Example: char = "Everest"\-1 produces this error because you cannot parse the leftmost -1 characters of a string.

-144 = sign or space expected
Everest thinks you have entered the name of a variable. Be sure that you use = to assign a value to a variable. This error can also occur if you forget to enter the desired command, such as BRANCH or IF.

-145 Bad function or mismatched ()
This indicates an expression does not have the same number of open and closing parentheses. It can also indicate that functions are nested more than 8 levels deep.

-146 Relational operator expected
Everest expected a relational operator in an expression, but did not find one.

-147 Constant or variable expected
An expression ended sooner than Everest expected.

-149 Unexpected character
Everest did not expect to find this character in this context. For example, using & in the non-conditional portion of an expression will produce this error.

-154 Mismatched parentheses
This indicates an expression does not have the same number of open and closing parentheses.

-156 Nested IF...THEN block not allowed
Currently, IF...THEN blocks may not be used inside other IF...THEN blocks. You can use a single-line IF expression inside an IF...THEN block.

-158 Page not found
A page you attempted to load or branch to does not exist. Check the page name and current book. In the course of creating your project, this error is normal during test runs when you have not yet created the page to which the project is attempting to branch.

-160 Book is full

There is no more room in the book. Note that special versions of Everest (such as the working model and evaluation package) have an artificial maximum limit on the number of pages in a book. The actual version of Everest is limited only by memory and disk space in the computer.

-161 Not available on your hardware

This operation is not available on this computer (due to hardware limitations).

-162 Illegal math operation

Everest detected an illegal operator in an expression.

-163 IF...THEN block missing ENDIF

The ENDIF command is missing.

-164 IF...THEN block not allowed here

Block-style IF...THEN expressions are not allowed here.

-165 Page name too long

Page names are limited to 8 characters. Object names can contain 19 characters.

-167 System help file not found

Everest was not able to locate the EVEREST.HLP file.

-168 System help page not found

Everest could not locate a topic in the help file.

-173 Expression too lengthy to evaluate

A-pex3 expressions are limited to 250 characters in length.

-174 WARNING: Page name mismatch

Each time Everest loads a page from disk, it double checks the name of the page that was loaded. This message indicates the name of the page loaded from disk does not match the name Everest was expecting. This error usually indicates a damaged book (.ESL file) due to a disk error. You should probably work from a backup copy of the book, as other pages may also be damaged.

-181 This item may not be edited

This page or object may not be changed.

-185 Process terminated by user

An informational error that indicates you stopped a process before it ran to completion.

-199 Page is empty

This means either the page contained no information, or was not found. Check for proper branching; make sure all necessary pages and books exist.

-201 Exceeded author defined var slots

There is no more room to hold author defined variables or array elements.

-202 Improperly initialized user record

This user's record was not properly initialized. Delete it via the INSTRUCT program, and recreate it.

-203 Duplicate user record
Two records in the user records database are identical. Please report this to tech support.

-204 No record in database
Everest expected to find a user's record in the database, but did not. Please report this to tech support.

-205 Source and destination must be different
You may not perform this operation (such as copying a book) within a given disk subdirectory.

-208 Bad internal message number
Everest used an incorrect message number. Please report this to tech support.

-209 Entry required here
Everest was expecting an entry in a field, but found none.

-210 Item deleted at another station
Before an item could be retrieved, it was deleted at another station (on the network).

-211 Available in Everest Professional only
This feature is not available in this version of Everest.

-212 DOS 3.0 is required
Everest requires DOS 3.0 as a minimum.

-213 GOTO label not found
A LABEL for a GOTO command was not found. A LABEL with the name specified in the GOTO command must exist in the same Program object.

-214 Reserved word
The names of variables may not be the same as the names of A-pex3 commands or functions.

-215 Too many parameters for function
This function does not accept this many parameters.

-216 Nested DO...LOOP not allowed
Currently, DO...LOOPs may not be used inside other DO...LOOPs.

-217 (RE)LOOP without DO
A LOOP or RELOOP command was detected outside a DO...LOOP.

-218 No such operation
This action does not exist.

-219 Insufficient region coordinates
The Reg() function or =T= operator did not find the expected number of X-Y coordinates (typically a single or double pair of coordinates is required).

-220 Missing LOOP
A DO command did not find a corresponding LOOP command.

-221 Illegal char in var name

Everest found an illegal character in the name of a variable.   Variable names must start with a letter, and may contain letters, numbers and the underline character (ASCII 95).

-222 Decimal point already used
In a numeric expression, Everest found more than one decimal point.

-223 Label not allowed inside IF or DO
The LABEL command is not allowed inside an IF...THEN c block or DO...LOOP construct.

-224 Label must be at leftmost edge of line
Do not indent LABEL commands.

-225 Undefined variable
This variable has not yet been defined.

-226 Undefined attribute
This attribute is unknown.

-227 No such object name
This object class name is unknown, or does not exist in this window.

-228 No such attribute name
This attribute is unknown.

-229 Parameter list must be enclosed with ()
The list of parameters for A-pex3 commands must be enclosed with parentheses.

-230 Max 20 rows on pull-down menu
Pull-down menus are limited to 20 rows.

-231 Max 8 columns on pull-down menu
Pull-down menus are limited to 8 columns.

-232 Bad object number
An object with this number does not exist.   Please report this to tech support.

-233 JLabel of JUMP not found
A JLabel object for the target of a JUMP was not found in this page.   Use the Name attribute of the JLabel object in the JUMP command.

-234 HyperHelp Error
The Windows help system returned an error.

-235 Includes nested too deeply
Include objects can be nested up to 8 levels deep.

-236 Wait object not allowed in included page
A Wait object may not appear in a page that is used by an Include object.

-237 MCI error detected
The Windows Media Control Interface detected an error.   Usually, additional error information is displayed.

-238 Exceeded max # of variable names

The limit on the number of unique variable names has been reached.

-239 Exceeded max # objects in one class
The limit on the number of objects in a given class has been reached.

-240 Cut & paste buffer is full
There is no more room in the cut & paste buffer.

-241 Backup stack is empty (can't do @prev branch)
A BRANCH @prev command was encountered, but there is no page in the list of previous pages (the backup stack). Consider making the BRANCH command conditional, such as IF Len(sysvar(71)) > 0 THEN BRANCH @prev.

-242 Menu stack is empty (can't do @menu branch)
A BRANCH @menu command was encountered, but there is no page in the list of menu pages (the menu stack). Consider making the BRANCH command conditional, such as IF Len(sysvar(81)) > 0 THEN BRANCH @menu.

-243 Call stack is empty (can't do return)
A RETURN command was encountered, but no corresponding CALL command had been used previously.

-244 Book stack is corrupted
The list of books has been damaged.

-245 Incorrect create syntax
The syntax of the Create attribute is incorrect.

-246 Bad IDNumber (must be from 1 to 99)
The IDNumber attribute must range from 1 to 99.

-247 Page is missing objects
Everest was not able to load all objects from disk.

-248 Can't find object in Page
Everest could not find an object in the Page.

-249 Unable to load page from disk
The page could not be loaded. This can indicate a damaged book.

-250 Feature no longer available
This feature is not available in this version of Everest.

-251 No such object class
The class name is bad.

-252 Unable to load object from disk
The object could not be loaded.

-253 .ESL has bad file format
The book file appears to be damaged. You may need to work from a back up copy.

-254 Too much data for linklist
The linked list database can handle records up to 32,000 bytes.

-255 Page too large to save

This page contains too much text to save in its current form. Change the SaveAsObject attribute of one or more objects (particularly large Program objects) to Yes, and retry.

-256 Window number must be from 1 to 8
Everest encountered a window number outside of the allowed range.

-257 CALL stack is corrupt
The list of pages on the CALL stack is damaged.

-258 Error during RTF file load
Everest was not able to load a Rich-Text Format file.

-259 No project running (for DDE request)
The Dde() function attempted a conversation with an application that is not currently running.

-260 BRANCH missing name of page
Enter the name of the desired page after the BRANCH command.

-261 Conditional DO or LOOP needs IF
A conditional expression in a DO or LOOP command must start with IF.

-262 Comment file name extension must be .ECM
Use .ECM as the file name extension.

-263 Records file name extension must be .EUR
Use .EUR as the file name extension.

-264 External reply timed out
An external application did not return a reply within the allotted time period. The external application may not be functioning correctly.

-265 No external object with this name
This indicates that a page refers to an external object, but that an external application that supports the object has not been loaded. Check that the proper external applications have been loaded either manually, or via the EVEREST.INI file's Extern# entries.

-266 Too many external files
Everest supports a maximum of 16 external applications with up to 8 objects each.

-267 Same external file loaded previously
This external application has already been loaded into memory.

-268 Book file name extension must be .ESL
Everest can save pages to books with .ESL file name extensions only.

-269 Exceeded IF block nesting level
IF blocks can be nested up to 8 levels deep.

-270 Exceeded DO...LOOP nesting level
DO...LOOPs can be nested up to 8 levels deep.

-271 Block IF without ENDIF
Block IF structures must end with an ENDIF command.

-272 DO without LOOP
DO...LOOP structures must end with a LOOP command.

-273 Not an Everest add-on module
The external application you are attempting to load does not contain an Everest identification signature. It is probably not suitable for use via Everest's extensibility features.

-274 Embedded file not found
Everest was unable to find the file embedded within the .ESL.

-275 Windows Help error
The Windows Help system reported an error. This could be caused by low memory or resources.

-276 .WMF too large
Due to a Windows bug, Everest must pre-process .WMF files. This limits the size of the file to 32K bytes. You can resolve this error message by embedding the .WMF into the book.

-277 String too long
Everest encountered a string that was too long. Strings are limited to 32K bytes.

-278 User record number mismatch
The user record number did not agree with the copy read from the user records file. This could indicate a damaged user records file, or unauthorized tampering with the user records.

-279 Please highlight (select) something before using this feature
This feature requires that you first choose the item or text on which to act. Do so, then retry.

-280 Newer version needed
You will need to upgrade to a newer version of Everest in order to edit this page.

-281 Cannot be edited with this version
The page you are attempting to load may not be edited with the version of Everest you are using.

-282 Protection failure 0
Make sure the key is securely attached to the parallel port.

-283 Protection failure 1
Make sure the key is securely attached to the parallel port.

-284 Protection failure 2
Make sure the key is securely attached to the parallel port.

-285 Improper spacing
Blank spaces are rarely significant in A-pex3 programming, but you've found a place they are. This can be caused by a space between an object and attribute name. The proper syntax is <object>.<attribute> with no blank spaces in between.

-286 Page already exists
A page by this name already exists. Choose a different name.

-287 Expected attribute name

In A-pex3 programming, object references are followed by a period and an attribute name. This error can be caused by a blank space or other illegal character between the period and attribute name.

-288 List full; if possible, narrow search

The size of a list exceeded 64K. If possible, narrow the search parameters so that the list will be shorter.

-289 Unexpectedly unable to find object

An object that should have been present in the Page was not found. Please report the conditions under which this occurs.

-301 A-pex3 Compiler framing error

The compiler encountered an illegal character in the compiled code. This could indicate a problem with the compiler. Please report the line of programming that triggers this error.

-304 Command not allowed here

The A-pex3 command you are using is not allowed in this location.

-307 Expected attribute

The compiler expected an attribute name, but found none. This could happen if you enter the name of an object without an attribute. Please report the line of programming that triggers this error.

-308 Unexpected attribute

The compiler encountered an unexpected reference to an object's attribute. Please report the line of programming that triggers this error.

-309 Can't find object

Everest was unable to find an object you referenced in your programming. This error can be caused by an attempt to GOSUB to a Program object that does not have SaveAsObject enabled. Also, check that you have correctly spelled the name of the Program.

-310 Problem with object counter

Internal error. Please report how to duplicate this to tech support.

-311 Unable to open/find book

Everest was unable to locate the book to open. Please report how to duplicate this to tech support.

-312 Name must start with a letter

The name of this item must start with a letter (a to z).

-313 Page order invalid

The order of the pages in this book has been lost or damaged. Everest will attempt to recover as much as possible, placing "lost" pages at the end of the book. In rare cases, you may need to work from a backup copy of the book.

-314 Unable to branch to @next or @back

Everest was unable to locate the page to branch to for a BRANCH @next or BRANCH @back command. This could happen if, for example, you use BRANCH @next from the last page in a book.

-315 Socket exception error
An unrecoverable error occurred during Inter/intranet communication.

-316 Socket close error
An error occurred while closing an Inter/intranet socket connection.

-317 Socket timeout
Everest waited to hear a reply from the Inter/intranet site, but none arrived; the communication channel might be blocked or disconnected. To tell Everest to wait longer, set INetTimeOut in the EVEREST.INI file, or Sysvar(181) to the desired number of seconds to wait.

-318 Maximum sockets already in use
Too many concurrent Inter/intranet connections have been established. Please report to tech support the conditions under which this error occurs.

-319 Bad host name
The name of the Inter/intranet host is unacceptable.

-320 File not found on host
Everest attempted to download the file you wanted from the host, but no such file was found. Check the name, and verify the host has that file for downloading.

-321 Error during FTP upload to host
An error occurred during the attempt to upload a file to the host. Your FTP settings may be incorrect in the Settings window.

-322 File to upload is missing or empty
Everest attempted to upload a file, but could not find it, or the file was empty (length 0).

-323 Upload timeout
Everest waited to hear a reply from the FTP upload server, but none arrived; the communication channel might be blocked or disconnected. To tell Everest to wait longer, set INetTimeOut in the EVEREST.INI file, or Sysvar(181) to the desired number of seconds to wait.

-324 Incompatible file
This file cannot be loaded because Everest cannot understand its contents. This error can occur, for example, if you attempt to display a graphics file stored in a format that is not supported.

-325 Download queue error
While a file is being downloaded, if a request comes in to download another, Everest will sometimes add it to a list for downloading momentarily. This error indicates something was wrong with the list.

-326 Incorrect book password
The password you entered is not the correct one.

-327 Bad link in file (incomplete data)
There is bad data in the .ESL or .EUR file. You may need to work from a backup. You can also try copying the contents of the file into a new file; for books, use the Copy Pages utility.

-328 Bad link in file (expected more data)
See error -327.

-329 Bad data in file (corrupted?)
See error -327.

-330 Cache and source location must be different
The location of the cache and the source cannot be identical. Use a different location for one or both.

-331 Unable to make directory
Everest could not create a subdirectory. Remember that Everest can only create subdirectories that are one level below existing directories.

-332 This book cannot be used with this version
Your version of Everest cannot use this book. For example, the Everest Free-Authoring Software cannot be used to modify or view books created with other versions.

-333 Choose an item from the list
Before you can proceed, you must choose (highlight) one of the items in the list displayed.

-334 Bad link in file (bad first pointer)
See error -327.

-335 Cannot paste there
You cannot paste icons at that location in the Book Editor. Try moving the pointer to another location (you can drag the pointer with the mouse).

-336 Please specify a page name
The operation you requested needs to know the name of the page to use. You may have omitted the page name, or left empty a field that requires a page name.

Microsoft's Error Messages

Microsoft's error messages are represented by positive numeric codes. Sometimes they can be caused by problems in Everest itself. Other times they are caused by limitations of the computer, or by your project.

001 NEXT without FOR
002 Syntax error
003 RETURN without GOSUB
004 Out of DATA
005 Illegal function call
This is Microsoft's "catch-all" message for any error that does not fit in another category. In order to provide advice about this error, we will need very specific details on how to reproduce it from scratch at our location.
006 Overflow
007 Out of memory
Usually this is caused by too many objects in a window. Another possible cause is insufficient available RAM in the computer.
008 Label not defined

009 Subscript out of range
010 Duplicate definition
011 Division by zero
012 Illegal in direct mode
013 Type mismatch
014 Out of string space

> This typically results from too much data stored in system and author defined variables.

016 String formula too complex
017 Cannot continue
018 Function not defined
019 No RESUME
020 RESUME without error
021 Unprintable error
022 Missing operand
023 Line too long
024 Device timeout
025 Device fault
026 FOR without NEXT
027 Out of paper
028 Out of stack space
029 WHILE without WEND
030 WEND without WHILE
033 Duplicate LABEL
034 Disk Full
035 Subprogram not defined
036 Subprogram already in use
037 Argument-count mismatch
038 Array not defined
039 CASE ELSE expected
040 Variable required
044 Disk is write-protected
045 File is locked
046 Volume is locked
047 File is busy (delete)
048 Error in loading DLL
049 Bad DLL calling convention
051 Microsoft internal error
052 Bad filename or number
053 File not found
054 Bad file mode
055 File already open
056 FIELD statement active
057 Device I/O error

> Usually the result of a disk drive error.

058 File already exists
059 Bad record length
061 Disk full
062 Input past end of file

> Fyl() function action 11 returns this error code after it has read the entire file. If you are looping and reading records from the file, this error is to be expected; simply exit the loop upon encountering this error,

063 Bad record number
064 Bad filename
067 Too many files
068 Device unavailable

069 Communication-buffer overflow
070 Permission denied

> This error code is returned when you attempt to open a file that is locked by another station on the network.   Retry opening the file until it becomes available (i.e. until no error code is returned).

071 Disk not ready
072 Disk-media error
073 Feature unavailable
074 Rename across disks
075 Path/File access error
076 Path not found
080 Feature removed
081 Invalid filename
082 Table not found
083 Index not found
084 Invalid column
085 No current record
086 Duplicate value for unique index
087 Invalid operation on null index
088 Database needs repair
091 Object variable not set
092 FOR loop not initialized
093 Invalid pattern string
094 Invalid use of Null
095 Cannot destroy active form instance
260 No timer available
280 DDE channel not fully closed
281 No more DDE channels
282 No foreign application responded to a DDE initiate
283 Multiple applications responded to a DDE initiate
284 DDE channel locked
285 Foreign application won't perform DDE method or operation
286 Timeout while waiting for DDE response
287 User pressed Esc during DDE operation
288 Destination is busy
289 Data not provided in DDE operation
290 Data in wrong format
291 Foreign application quit
292 DDE conversation closed or changed
293 DDE method invoked with no channel open
294 Invalid DDE Link format
295 Message queue filled: DDE message lost
296 PasteLink already performed on this object
297 Cannot set LinkMode; invalid LinkItem or LinkTopic
298 DDE requires DDEML.DLL
320 Cannot user character device names in filenames
321 Invalid file format
340 Control array element does not exist

> This error can result from referring via A-pex3 programming to an object that does not (yet) exist in the window.   Remember that at run time the Page is executed in order from top to bottom.   Be sure to position Program objects in the Page AFTER the objects to which their A-pex3 programming refers.   Also, check that you are employing the correct ID number.

341 Invalid object array index
342 Not enough room to allocate control array
343 Object not an array

344 Must specify index for object array
345 No more objects allowed in this window
360 Object already loaded
361 Cannot load or unload this object
362 Cannot unload controls created at design time
363 Custom control not found
364 Object was unloaded
365 Unable to unload within this context
366 No MDI form available to load
380 Invalid attribute value
381 Invalid attribute array index
382 Attribute cannot be set at run time
383 Attribute is read-only
384 Attribute cannot be modified when window minimized or maximized
385 Must specify index when using property array
386 Attribute not available at run time
387 Attribute cannot be set on this object
388 Cannot set Visible attribute from a parent menu
389 Invalid key
390 No defined value
391 Name not available
392 MDI child windows cannot be hidden
393 Attribute cannot be read at run time
394 Attribute is write-only
395 Cannot use separator bar as menu name
400 Window already displayed
401 Cannot show non-modal window when modal window is already displayed
402 Must close or hide topmost modal window first
403 MDI windows cannot be shown modally
404 MDI child windows cannot be shown modally
420 Invalid object reference
421 Method not applicable for this object
422 Attribute not found
423 Attribute or control not found
424 Object required
425 Invalid object user
426 Only one MDI window allowed
460 Invalid Clipboard format
461 Specified format does not match that of data
480 Cannot create AutoRedraw image
481 Invalid picture
482 Printer error
520 Cannot empty Clipboard
521 Cannot open Clipboard

Inter/intranet related

20000 Connection closed
20001 Memory allocation error
20002 Socket is already closed
20003 Socket is aready listening
20004 No port number or service name specified
20005 Socket is already connected
20006 UDP Socket is already active
20010 Winsock DLL not found
20011 Socket is not closed

21004 Interrupted system call
21009 Bad file number
21013 Permission denied
21014 Bad address
21022 Invalid argument
21024 Too many open files
21035 Operation would block
21036 Operation now in progress
21037 Operation already in progress
21038 Socket operation on non-socket
21039 Destination address required
21040 Message too long
21041 Protocol wrong type for socket
21042 Bad protocol option
21043 Protocol not supported
21044 Socket type not supported
21045 Operation not supported on socket
21046 Protocol family not supported
21047 Address family not supported by protocol family
21048 Address already in use
21049 Can't assign requested address
21050 Network is down
21051 Network is unreachable
21052 Net dropped connection or reset
21053 Software caused connection abort
21054 Connection reset by peer
21055 No buffer space available
21056 Socket is already connected
21057 Socket is not connected
21058 Can't send after socket shutdown
21059 Too many references, can't splice
21060 Connection timed out
21061 Connection refused
21062 Too many levels of symbolic links
21063 File name too long
21064 Host is down
21065 No Route to Host
21066 Directory not empty
21067 Too many processes
21068 Too many users
21069 Disc Quota Exceeded
21070 Stale NFS file handle
21071 Too many levels of remote in path
21091 Network SubSystem is unavailable
21092 WINSOCK DLL Version out of range
21093 Successful WSASTARTUP not yet performed
22001 Host not found
22002 Non-Authoritative Host not found
22003 Non-Recoverable errors: FORMERR, REFUSED, NOTIMP
22004 Valid name, no data record of requested type

Appendix D - File Naming Scheme

The following table will help you identify files created by Everest:

.EAL	Everest access list. Create via INSTRUCT program to limit Internet access to your projects to authorized users.
.ECM	Everest comments file. Create via BRANCH @comment. View/edit via the INSTRUCT program.
.ESL	Everest book (originally "Everest screen library). Create/edit via the AUTHOR program.
.EUR	Everest user records. Create/edit via the INSTRUCT or ERUN programs.
.EP?	Temporary file to hold images between user log off and log on.
.SP?	Temporary file to hold images between user log off and log on.
EVEREST.INI	Everest configurations file. Can be edited with the INSTRUCT program or text editor.
CMIFILE.DAT	Temporary holding file for data collected by the CMIData attribute and Rec() function. Generate reports with SUMCMI.EXE.

Index